



Barman - User Guide

Release 3.14.1

EnterpriseDB UK Limited

Jul 28, 2025

Contents

1	Concepts	3
1.1	Introduction	3
1.2	General backup concepts	4
1.3	Postgres backup concepts and terminology	6
1.4	Barman concepts and terminology	8
2	Installing	11
2.1	System requirements	11
2.2	RHEL-based distributions	12
2.3	Debian-based distributions	13
2.4	SLES-based distributions	14
3	Quick start	15
3.1	Configuring your first server	15
3.2	Taking your first backup	19
3.3	Restoring a backup	20
4	Architectural backup designs with barman	21
4.1	Where to install Barman	21
4.2	One Barman, many Postgres servers	23
4.3	Backup strategies	23
4.4	WAL archiving strategies	23
4.5	Two typical scenarios for backups	24
4.6	Geographical redundancy	28
4.7	Cloud snapshot backups	29
5	Pre-requisites	31
5.1	Postgres users	31
5.2	Postgres connection	32
5.3	Postgres client tools	32
5.4	Postgres streaming replication connection	33
5.5	SSH connections	34
5.6	WAL archiving via <code>archive_command</code>	35
6	Barman check	37
6.1	Usage	37
6.2	Understanding the output	38

7	Backups	41
7.1	Overview	41
7.2	Incremental Backups	42
7.3	Managing Bandwidth Usage	42
7.4	Network Compression	43
7.5	Backup Compression	43
7.6	Backup Encryption	45
7.7	Immediate Checkpoint	46
7.8	Streaming Backup	46
7.9	Backup with Rsync through SSH	47
7.10	Concurrent Backup of a Standby	48
7.11	Managing external configuration files	49
7.12	Using an immutable storage for backups	49
7.13	Cloud Snapshot Backups	50
8	WAL archiving	59
8.1	WAL compression	59
8.2	WAL encryption	60
8.3	Synchronous WAL streaming	60
9	Catalog information	63
9.1	Purpose	63
9.2	Usage	64
10	Recovery	67
10.1	Local recovery	68
10.2	Remote Recovery	68
10.3	Tablespace Remapping	68
10.4	Point-in-Time Recovery	69
10.5	Fetching WALs from Barman	70
10.6	Recovering Encrypted Backups	71
10.7	Recovering Compressed Backups	72
10.8	Recovering block-level incremental Backups	72
10.9	Limitations of .partial WAL files	73
10.10	Managing external configuration files	73
10.11	Recovering from Snapshot Backups	73
11	Retention policies	77
11.1	Overview	77
11.2	Key Components of Retention Policies	77
11.3	Key Objectives of Retention Policies	78
11.4	Minimum redundancy safety	78
11.5	Scope of retention policies	79
11.6	Use cases	79
11.7	How retention policies are enforced	80
11.8	Configurations and Syntax	80
11.9	Retention policy for block-level incremental backups	80
11.10	Retention policy for Cloud Backups	80
12	Diagnose and troubleshooting	83
12.1	Barman status	83
12.2	Barman show servers	84
12.3	Barman replication status	84
12.4	Barman diagnose	84

13 Configuration Reference	85
13.1 Usage	85
13.2 Options	86
13.3 Configuration Models	105
14 Commands Reference	107
14.1 barman	107
14.2 Shortcuts	109
14.3 Exit Statuses	109
14.4 Sub-Commands	109
14.5 barman-cli commands	130
15 Geographical Redundancy	135
15.1 Sync information	135
15.2 Configuration	136
15.3 Node synchronization	136
15.4 Manual synchronization	136
16 Hook Scripts	137
16.1 Before and after creating a backup	138
16.2 Before and after deleting a backup	138
16.3 Before and after a WAL is archived	139
16.4 Before and after a WAL file is deleted	139
16.5 Before and after restoring a backup	140
16.6 Using barman-cloud-* scripts as hooks in barman	140
17 Barman for the cloud	141
17.1 Barman cloud client package	142
17.2 Installation	142
17.3 Commands Reference	142
18 Glossary	165
Index	167

Barman is an open-source administration tool for disaster recovery of Postgres servers written in Python. It allows your organisation to perform remote backups of multiple servers in business critical environments to reduce risk and help DBAs during the recovery phase.

Barman is distributed under GNU GPL 3 and maintained by [EnterpriseDB](#), a platinum sponsor of the [PostgreSQL](#) project.

Creating a disaster recovery plan can be challenging, especially for those unfamiliar with the various concepts involved in backup management. There are many different methods for taking backups, each with its own advantages, disadvantages, and technical requirements. The choice of the right approach will depend on your resources, environment and technical knowledge. Knowing that not everyone might be well-grounded in this context, this section is dedicated to explaining the most fundamental concepts regarding database backups, particularly in the context of Postgres and Barman.

If you are already familiar with the concepts of backups, logical and physical backups in Postgres, feel free to skip to the [Barman concepts and terminology](#) section.

1.1 Introduction

In a perfect world, backups wouldn't be necessary. However, it is important, especially in critical business environments, to be prepared for when the unexpected happens. In a database scenario, the "unexpected" could take any of the following forms:

- Data corruption.
- System failure (including hardware failure).
- Human error.
- Natural disaster.

In such cases, any *ICT* manager or *DBA* should be able to fix the incident and recover the database in the shortest time possible. We normally refer to this discipline as disaster recovery, and more broadly as business continuity.

Within business continuity, it is important to familiarize yourself with two fundamental metrics, as defined by Wikipedia:

- Recovery Point Objective (RPO): the maximum targeted period in which data might be lost from an IT service due to a major incident.
- Recovery Time Objective (RTO): the targeted duration of time and a service level within which a business process must be restored after a disaster (or disruption) in order to avoid unacceptable consequences associated with a breakage in business continuity.

In a few words, RPO represents the maximum amount of data you can afford to lose, while RTO represents the maximum down-time you can afford for your service.

Understandably, we all want RPO=0 (zero data loss) and RTO=0 (zero down-time, utopia), even if it is our grandmother's recipe website. In reality, a careful cost analysis phase is required to determine your business continuity requirements.

Fortunately, with an open source stack composed of Barman and Postgres, you can achieve RPO=0 thanks to synchronous streaming replication. RTO is more the focus of a High Availability solution, like Patroni or repmgr. Therefore, by integrating Barman with any of these tools, you can dramatically reduce RTO to nearly zero.

In any case, it is important for us to emphasize more on cultural aspects related to disaster recovery, rather than the actual tools. Tools without human beings are useless. Our mission with Barman is to promote a culture of disaster recovery that:

- Focuses on backup procedures.
- Focuses even more on recovery procedures.
- Relies on education and training on strong theoretical and practical concepts of Postgres crash recovery, backup, Point-In-Time-Recovery, and replication for your team members.
- Promotes testing your backups (only a backup that is tested can be considered to be valid), either manually or automatically (be creative with Barman's hook scripts!).
- Fosters regular practice of recovery procedures, by all members of your devops team (yes, developers too, not just system administrators and *DBAs*).
- Solicits regularly scheduled drills and disaster recovery simulations with the team every 3-6 months.
- Relies on continuous monitoring of Postgres and Barman, and that is able to promptly identify any anomalies.

Moreover, do everything you can to prepare yourself and your team for when the disaster happens, because when it happens:

- It is going to be a Friday evening, most likely right when you are about to leave the office.
- It is going to be when you are on holiday (right in the middle of your cruise around the world) and somebody else has to deal with it.
- It is certainly going to be stressful.
- You will regret not being sure that the last available backup is valid.
- Unless you know how long it approximately takes to recover, every second will seem like forever.

In 2011, with these goals in mind, 2ndQuadrant started the development of Barman, now one of the most used backup tools for Postgres. Barman is an acronym for "Backup and Recovery Manager".

Be prepared, don't be scared.

1.2 General backup concepts

While each database system may have its own terminology, there are fundamental backup principles that are consistent across all relational databases. This section provides an overview of the core concepts necessary to understand how backups work.

1.2.1 Physical and logical backups

In the context of relational databases, a logical backup is nothing more than a series of operations that, when executed, recreate your data in the exact state as when the backup was taken. To put it simple, it is a sequence of SQL statements to reconstruct the database structure and data. This method is not dependent on specific environment specifications such

as database version or system architecture, making it a suitable choice for migrating data across incompatible systems. It also offers more flexibility by allowing the restoration of specific database objects or tables.

However, logical backups can be time-consuming, potentially taking several hours or days, depending on the size of the database. Also, because the backup reflects the database's state at the start of the backup process, any changes made during the backup are not captured, introducing potential windows for data loss. As a result, this method is typically recommended for smaller and less complex databases. In Postgres, logical backups are implemented via the `pg_dump` and `pg_dumpall` utilities.

A physical backup, on the other hand, works by copying the database files directly from the file system. Therefore, this method is usually tied to environment specifications. There are different approaches to taking physical backups, ranging from using basic Unix tools like `cp` to more sophisticated solutions such as using backup managers, like Barman. Backup management tools can play a vital role in physical backups, as ensuring the files represent a consistent state of the database, while also keeping the server running normally, can be challenging if done manually.

Physical backups can be much faster than logical backups, not only during the backup process but especially during recovery, since they do not require a complete replay of operations in order to recreate the database. It also enables the possibility of incremental backups, which significantly reduces time and storage usage, allowing for more frequent backups. Finally, one of the greatest advantages of this approach is the ability to perform point-in-time recovery (PITR), which allows you to restore your database to any specific point in time between the current time and the time of the backup. This feature is only possible when transaction logs are archived alongside your physical backups.

As noticed, physical backups are more robust but also more complex. For this reason, auxiliary backup management tools, like Barman for Postgres, play an important role in ensuring this process is handled effectively and reliably in your disaster recovery plan.

1.2.2 Backup types

Regarding physical backups, they can essentially be divided into three different types: full, incremental and differential.

A full backup, often also called base backup, captures all your data at a specific point in time, essentially creating a complete snapshot of your entire database. This type of backup contains every piece of information needed to restore the system to its exact state as when the backup was taken. In this sense, a recovery from a consistent full physical backup is the fastest possible, as it is inherently complete by nature.

Incremental backups, on the other hand, are designed to capture only the changes that have occurred since a previous backup. A previous backup could be either a full backup or another incremental backup. Incremental backups significantly reduce the time and storage usage, allowing for more frequent backups, consequently reducing the risks of data loss. Generally, Incremental backups are only possible with physical backups as they rely on low-level data structures, such as files and internal data blocks, to determine what has actually changed. A recovery from an incremental backup requires a chain of all backups, from the base to the most recent incremental.

Lastly, differential backups are similar to incremental backups in that they capture only the changes made since a previous backup. However, the key difference is that a differential backup always records changes relative to a full backup, never being relative to an incremental or another differential backup. In fact, every differential backup is an incremental backup, but not every incremental backup is a differential backup. A recovery in this case only requires the most recent differential backup and its related base backup.

1.2.3 Transaction logs

Transaction logs are a fundamental piece of most relational databases. It consists of a series of contiguous files that record every operation in the database before they are executed. As a result, they possess every change that happened in the database during a period of time. It primarily ensures that databases can effectively recover from crashes by being able to replay any operations that were not yet flushed to disk before the crash, thus preventing data loss. It is also a key component of many implementations of database replication.

Transaction logs are recycled after all its operations are persisted. However, if we are able to archive these logs in advance, we essentially retain a complete record of all changes made to the database that can be replayed at any time.

Having a base backup along with transaction logs archival enables for continuous backups. This is particularly valuable for large databases where it is not possible to take full backups regularly. You might notice that it achieves a similar goal as differential backups, but with even more capabilities as it also enables more robust features such as point-in-time recovery.

1.2.4 Point-time Recovery

Point-in-time recovery enables you to restore your database to any specific moment from the end-time of a base backup to the furthest point covered by your archived transaction logs. By maintaining a continuous archive of transaction logs, you have the ability to replay every change made to the database up to the present moment. This is done by replaying all transaction logs on top of a base backup, also providing you with the ability to stop the replay at any point you want based on a desired timestamp or transaction ID, for example. PITR allows for a precision of less than seconds, a huge advantage over standard full backups, which are usually executed daily.

This feature is especially valuable in situations where human error or unintended changes occur, such as accidental deletions or modifications. By restoring the database to the exact state it was just before the unwanted event, PITR significantly reduces RPO. It provides a powerful safeguard, ensuring that critical data can be quickly and accurately recovered without reverting the database to an earlier full backup and risk losing all subsequent legitimate changes.

It does not mean, however, that PITR is the solution to all problems. Replaying transaction logs can still take a long time depending on how far they go from the base backup. Therefore, the optimal solution is actually a combination of all strategies: full backups with frequent incremental backups along with transaction log archiving. This way, restoring to the most recent state is a matter of restoring the most recent backup followed by a replay of subsequent transaction logs. Similarly, restoring to a specific point in time is a matter of restoring the previous backup closest to the target point followed by a replay of subsequent transaction logs up to the desired target.

1.3 Postgres backup concepts and terminology

This section explores backup concepts in the context of Postgres, its implementations and specific characteristics. The content is mainly based on the [Backup and Restore](#) section from the [Postgres official documentation](#), so we strongly recommend you read that if you want more detailed explanations on how Postgres handles backups.

1.3.1 pg_dump vs pg_basebackup

There are essentially two main tools for taking backups in Postgres: `pg_dump` and `pg_basebackup`. The difference between them is essentially the difference between logical and physical backups. Namely, `pg_dump` (`pg_dumpall` included) takes logical backups while `pg_basebackup` takes physical backups.

Note

Barman does not make use of `pg_dump` or `pg_dumpall` in any way as it does not operate with logical backups. `pg_basebackup` is used by Barman depending on the backup method configured.

`pg_basebackup` essentially copies all files from your Postgres cluster to a destination directory, including tablespaces, if any, using the [streaming replication protocol](#). It can only backup the entire cluster, not being able to backup specific databases or objects. `pg_basebackup` is responsible for putting your database server in and out of backup mode as well as making sure all required transaction logs for consistency are stored along with the base backup. For that reason, unlike `pg_dump`, a backup taken with `pg_basebackup` also includes changes that happened while the backup was in progress, which is a huge advantage for databases under frequent heavy load. You can read more about `pg_basebackup` in its [dedicated section in the official documentation](#).

Note

In reality, a physical backup in Postgres is only complete/self-contained if it also has at least the transaction logs (WALs in Postgres) that were generated during the backup process. Otherwise the backup itself is insufficient to restore and start a new Postgres instance.

It is also possible to accomplish a similar result as `pg_basebackup` using the [Postgres low-level backup API](#), which is yet another way of taking physical backups in Postgres. The low-level API is used in cases where you want to take physical backups manually using alternative copying tools. In this scenario, you are responsible for putting the database server in and out of backup mode manually as well as making sure all transaction logs required for consistency are archived correctly.

Note

Barman uses the Postgres low-level API depending on the backup method configured e.g. `backup_method = rsync`.

1.3.2 Write-ahead logs

Write-ahead logs (WAL) is how Postgres (and other databases) refer to transaction logs. In Postgres, each WAL file supports 16 MB worth of changes (configurable). WAL files are written sequentially, one after another, and are maintained simultaneously until a checkpoint is performed. A checkpoint in Postgres is the act of persisting all changes to disk so that WALs can be recycled afterwards. A checkpoint usually happens every five minutes or after 1 GB of WAL files are generated, both options are configurable. WAL not only helps with crash recovery, database replication and PITR, but it's also an important component to ensure good performance, as otherwise changes would need to be synced to disk after each transaction commit, resulting in huge I/Os. With WAL, changes can be postponed to a checkpoint-time since it is sufficient to ensure database consistency.

1.3.3 WAL archiving and WAL streaming

Transaction log archiving is known as "continuous archiving" or "WAL archiving" in Postgres. WAL archiving essentially means being able to store WAL files somewhere else before they are recycled. In Postgres, the traditional way of doing that is via the `archive_command` parameter in the server configuration.

The `archive_command` accepts any shell command as a value, which will be executed for each WAL file once completely filled. Such a command is responsible for making sure each file is copied safely to the desired destination. This provides a lot of flexibility in the sense that Postgres does not make any assumptions on how or where you want to store these files, thus allowing you to use any command or library you want. This command must return a zero exit status, indicating success, otherwise Postgres understands that the archiving has failed and will not recycle those files until they can be successfully archived. While this is helpful for ensuring safety it can also become a nightmare if your command starts failing for some reason as WAL files will continue to pile up until it works again or you run out of disk space. You can read more about [WAL archiving in the official documentation](#).

An alternative way of archiving WALs is by using `pg_receivewal`, a native Postgres utility used to transfer WAL files to a desired location using the streaming replication protocol. A huge advantage of this method, commonly known as WAL streaming, compared to the traditional `archive_command` method is that files are transferred in real time, meaning that it doesn't need to wait for a WAL segment to be completely filled in order to start transferring it, significantly reducing the chances of data loss.

Unlike the `archive_command`, by default this method alone does not ensure that WAL files are archived successfully before being recycled. This means that WAL files can be recycled before being archived, essentially having its logs lost forever. For this reason, the use of replication slots is extremely recommended in this scenario. Replication slots are primarily used in the context of database replication to ensure that the primary server will retain WAL files needed by

its following replicas until they are successfully received, providing an extra safety in case a replica goes offline or gets disconnected. It achieves the same goal when used with `pg_receivewal` i.e. making sure WAL files are not recycled until successfully transferred to the receiver.

1.3.4 Recovery

The recovery process in Postgres depends on the backup type. With logical backups, this process is as simple as running `pg_restore` or simply executing all SQL commands from the backup file, depending on the backup file format. With physical backups, however, the process is a bit more complex.

To successfully recover from a physical backup, you need both the cluster files and its WAL archive. It is necessary to have at least the WAL files that were generated during the backup process. If the backup was taken with `pg_basebackup`, the required WAL files will already be included in the output directory, unless specified otherwise. If taken manually, however, using the Postgres low-level API, it is your responsibility to make sure all required WAL files are available during recovery.

To prepare for recovery, you need to follow a few steps. This includes specifying a few parameters in the configuration file of the backup cluster directory, such as a command to get the WAL files from the WAL archive as well as a target point, in case performing *PITR*, among others. For a detailed explanation of this process, refer to the [Postgres official documentation](#). If everything is correct, you should then be able to start a new instance from the backup and Postgres will make sure all required WALs are applied.

If the recovery involves Postgres incremental backups, you will then need to first combine all the backups using `pg_combinebackup`. It will generate a synthetic full backup, which can be used for recovery in the same way as a standard full backup.

1.4 Barman concepts and terminology

This section offers an overview of important Barman concepts as well as demonstrates how Barman utilizes some of the concepts explained in earlier sections.

1.4.1 Server

Barman can manage backups of multiple database servers simultaneously. For this reason, a logical separation of your backup servers becomes necessary. In Barman, a backup server, or simply server, represents the backup context of a specific database server. It defines how Barman interacts with the database instance, how its backups are managed, their retention policies, etc. Each server has its own dedicated directory where all backups and WAL files are stored as well as a unique name which must be supplied in most Barman commands to specify in which context it should run.

1.4.2 Backup methods

As outlined in *Postgres backup concepts and terminology*, there are multiple ways to back up a Postgres server. In the context of Barman, these are referred to as backup methods. Barman supports various backup methods, each relying on different Postgres features, with its own set of requirements, advantages, and disadvantages. The desired backup method can be specified using the `backup_method` parameter in the server's configuration file.

Note

It is highly recommended to use a single backup method when managing your Barman server. If you need to switch backup methods, it's advisable to set up a new Barman server.

1.4.3 Rsync backups

Backups taken with `backup_method = rsync`. When using this backup method, Barman uses the Postgres low-level API and Rsync to manually transfer cluster files over an SSH connection. Rsync is a powerful copying tool which allows you to synchronize files and directories between two locations, either on the same host or on different hosts over a network. Barman utilizes the low-level API to put the server in and out of backup mode while using Rsync to copy all relevant files to the server's designated directory on Barman. At the end of this process, Barman forces a WAL switch on the database server to ensure that all required WAL files are archived. Finally, integrity checks are performed to verify that the backup is consistent.

1.4.4 Streaming Backups

Backups taken with `backup_method = postgres`. When using this backup method, Barman invokes `pg_basebackup` in order to back up your database server. Barman will map all your tablespaces to the server's dedicated directory on Barman. At the end of this process, Barman forces a WAL switch on the database server to ensure that all required WAL files are archived. Finally, integrity checks are performed to verify that the backup is consistent.

1.4.5 Snapshot Backups

Snapshot backups can be performed either by setting `backup_method = snapshot` or by directly using the Barman's cloud CLI tools. These backups work by integrating Barman with a cloud provider where your database server resides. A snapshot of the database's storage volume is then taken as a physical backup. In this setup, Barman manages your backups in the cloud, acting primarily as a storage server for WAL files and the backups catalog.

1.4.6 File-level incremental backups

File-level incremental backups are possible when using *rsync* backups. It uses Rsync native features of deduplication, which relies on filesystem hard-links. When performing a file-level incremental backup, Barman first creates hard-links to the latest server backup available, essentially replicating its content in a different directory without consuming extra disk space. Rsync is then used to synchronize its contents with the contents of the Postgres cluster, copying only the files that have changed. You can also have file-level incremental backups without using hard-links, in which case Barman will first copy the contents of the previous backup to the new backup directory, essentially duplicating it and consuming extra disk space, but still copying only changed files from the database server.

1.4.7 Block-level incremental backups

Block-level incremental backups are possible when using *streaming backups*. It leverages the native `pg_basebackup` capabilities for incremental backups, introduced in Postgres 17. This features requires a Postgres instance with version 17 or newer that is properly configured for native incremental backups. With block-level incremental backups, any backup with a valid `backup_manifest` file can be used as a reference for deduplication. Block-level incremental backups are more efficient than file-level incremental backups as deduplication happens at the block level (pages in Postgres).

1.4.8 WAL archiving via `archive_command`

This is one of the two ways of transferring WAL files to Barman. Commonly used along with *rsync* backups, this approach involves configuring the `archive_command` parameter in Postgres to archive WAL files directly to the server's dedicated directory on Barman. The command can be either an Rsync command, where you manually specify the server's WAL directory on the Barman host, or the `barman-wal-archive` utility, which only requires the server name, with Barman handling the rest. Additionally, `barman-wal-archive` provides added safety by ensuring files are fsynced as soon as they are received.

1.4.9 WAL streaming

This is one of the two ways of transferring WAL files to Barman. Commonly used along with *streaming backups*, this approach relies on the `pg_receivewal` utility to transfer WAL files. It is much simpler to configure, as no manual configuration is required on the database server. As mentioned in *WAL archiving and WAL streaming*, replication slots are recommended when using WAL streaming. You can create a slot manually beforehand or let Barman create them for you by setting `create_slot` to `auto` in your backup server configurations.

1.4.10 Hook Scripts

Barman enables developers to execute hook scripts along specific operations as pre- and/or post-operations. This feature provides developers with the flexibility to implement tailored and diverse behaviors. You can utilize a post-backup script to generate a manifest for the backup when using `rsync` or you can create a hybrid distributed architecture that allows you to copy backups to cloud storage as well by combining post-backup *hook scripts with barman-cloud* commands.

For a more in-depth exploration of this topic, please refer to the main section on *hook scripts*.

1.4.11 Restore and recover

In Barman, recovery is the process of restoring a backup along with all necessary WAL files in a new location, effectively preparing a Postgres instance for recovery.

As outlined in *Recovery*, the recovery process in Postgres consists of several steps, from preparing the base directory to starting the server itself. Barman is able to perform all the steps required to prepare your backup to be recovered, a process known as "restore" in Barman's terminology. In this case, completing the recovery is usually just a matter of starting the server so that Postgres can apply the required WALs and go live.

CHAPTER 2

Installing

Barman official packages are provided by *PGDG*. These packages use the default version of Python 3 that comes with the operating system.

There are three packages that make up the suite of Barman features: `barman`, `barman-cli` and `barman-cli-cloud`.

- `barman` is the main package and it must be installed.
- `barman-cli` is an optional package that holds the `barman-wal-restore` and `barman-wal-archive` utilities. This package is mandatory if you plan to use those utilities as the `archive_command` or `restore_command`. It must be installed on each Postgres server that is part of the Barman cluster.
- `barman-cli-cloud` is an optional package that holds the `barman-cloud-*` client scripts that you can use to manage backups in a cloud provider. It must be installed on the Postgres servers that you want to back up directly to a cloud provider, bypassing Barman.

Note

Barman packages can be found in several different repositories. We recommend using PGDG repositories because it ensures compatibility, stability and access to the latest updates.

Warning

Do not upgrade Barman using different repositories. By doing so you risk losing your configuration as each source repository provides different packages, which use different configuration layouts.

2.1 System requirements

The minimal system requirements needed to run a Barman server are the following:

- Linux operating system (Debian, Ubuntu, RHEL, Rocky, Fedora, etc.) or UNIX-like operating system (FreeBSD, OpenBSD, etc.)

- Python 3.8 or higher
- Python modules:
 - `psycopg2` >= 2.4.2: Required to connect to the Postgres server
 - `python-dateutil`
 - `setuptools`
 - `argcomplete` (optional)
- PostgreSQL client tools: Required to interact with the Postgres server
- PostgreSQL server >= 13
- `rsync` >= 3.1.0: Required for recovery and Rsync backups
- `boto3` >= 1.29.1: Required when using `backup_method = snapshot` together with the snapshot lock feature on AWS
- `file` POSIX command, generally provided by the `file` package

Deprecated since version 3.14: Support for versions 3.6 and 3.7 of Python has been deprecated. It is known that Barman 3.14 does not work with Python 3.6. It may work with Python 3.7, but it's not being tested, nor supported for versions of Python prior to 3.8.

Note

Users of RedHat Enterprise Linux, RockyLinux and AlmaLinux are required to install the *Extra Packages Enterprise Linux (EPEL) repository* <<https://fedoraproject.org/wiki/EPEL>>

2.2 RHEL-based distributions

You can install `barman`, `barman-cli` and `barman-cli-cloud` using *RPM* packages on *RHEL* systems as well as on similar RHEL-based systems like AlmaLinux, Oracle Linux and Rocky Linux.

To begin with the installation, first install the *PGDG RPM repository*.

Important

The `barman-cli-cloud` scripts are part of the `barman-cli` package for RHEL-based distributions from *PGDG*. Therefore, you only need to install `barman-cli` to use the cloud scripts.

2.2.1 barman

To install the `barman` package. Run as **root**:

```
dnf install barman
```

2.2.2 barman-cli

To install the `barman-cli` package, run as **root** in the Postgres server:

```
dnf install barman-cli
```

Note

If you want to use the barman-cloud utilities as *hook scripts*, you will need to install the barman-cli package in the Barman server.

2.3 Debian-based distributions

You can install barman, barman-cli and barman-cli-cloud using *DEB* packages on Debian systems as well as on Debian-based systems like Ubuntu.

To begin with the installation, install the PGDG APT repository. This depends on your system:

- For Debian: [PGDG Debian repository](#).
- For Ubuntu: [PGDG Ubuntu repository](#).

Important

The barman-cli-cloud package is included among the recommended packages when you install barman-cli.

Before starting the installation, it's essential to evaluate your use case. If you don't plan to use the barman-cloud client scripts, such as barman-cloud-backup, you can skip installing barman-cli-cloud as a recommended package when installing barman-cli. However, if you only intend to use the barman-cloud client scripts, you can install the barman-cli-cloud package on its own.

2.3.1 barman

To install the barman package. Run as **root**:

```
apt install barman
```

2.3.2 barman-cli

To install the barman-cli package, run as **root** in the Postgres server:

```
apt install barman-cli
```

2.3.3 barman-cli-cloud

To install the barman-cli-cloud package, run as **root** in the Postgres server:

```
apt install barman-cli-cloud
```

Note

If you want to use the barman-cloud utilities as *hook scripts*, you will need to install this package in the Barman server.

2.4 SLES-based distributions

You can install barman on *SLES* systems by utilizing the packages provided in the [PostgreSQL Zypper Repository](#).

To begin installation, you will need to add the appropriate repository by following the detailed instructions available on the [PGDG SLES Repository Configuration](#).

The current supported version for installation is SLES 15 SP6.

Important

The `barman-cli-cloud` utilities are part of the `barman-cli` package for SLES-based distributions from *PGDG*. Therefore, you only need to install `barman-cli` to use the cloud scripts.

2.4.1 barman

To install the barman package. Run as **root**:

```
zypper install barman
```

2.4.2 barman-cli

To install the `barman-cli` package, run as **root** in the Postgres server:

```
zypper install barman-cli
```

Note

If you want to use the barman-cloud utilities as *hook scripts*, you will need to install this package in the Barman server.

As it is stated in *Architectural backup designs with barman*, we recommend setting up Barman in a dedicated host. That said, the examples in this tutorial assume the following hosts:

- **pghost**: The host where Postgres is running.
- **barmanhost**: The host where Barman will be set up.

Assuming Barman is already installed in **barmanhost** as per *installation*, you can continue through the next steps.

3.1 Configuring your first server

Barman supports different backup and WAL archive strategies. Here you can find simple recipes to set up two of the most commonly used architectures. Choose the one that best suits your needs and proceed to the next sections.

3.1.1 Streaming backups with WAL streaming

This strategy uses the Postgres streaming replication protocol for both backups and WAL archiving, so you only need native Postgres connections between **pghost** and **barmanhost** in order to implement it. A key advantage of this approach is that no SSH connection is required, making it a simpler option to set up in comparison with other strategies.

It relies on the `pg_basebackup` utility for backups and `pg_receivewal` for transferring the WAL files. It is therefore required to have both tools installed on **barmanhost** beforehand. Check the *Postgres client tools* section if you need further details on how to install these tools.

1. As a first step, let's create the required users you will need on your Postgres server. On **pghost**, execute the following commands:

```
createuser -s -P barman
```

This command creates a new Postgres superuser called **barman**, which will be used by Barman for maintenance tasks on your Postgres server. Alternatively you can create a user without superuser privileges, but with the necessary permissions to perform the needed operations by following the recipe in *Postgres users pre-requisite*.

```
createuser -P --replication streaming_barman
```

This command creates a new Postgres user called **streaming_barman** with replication privileges, which will be used by Barman when invoking `pg_receivewal` and `pg_basebackup` to transfer files to your Barman server.

Both `createuser` commands prompt you for a password, which you are then advised to add to a [password file](#) named `.pgpass` under your Barman home directory on `barmanhost`. Check the [pre-requisites](#) section if you need further details on how to configure streaming connections.

From now on, this section assumes you already have both Postgres users created as well as a Postgres server to be backed up. Also, we assume a database named `postgres` is available, so Barman can connect to the Postgres server through that database.

2. Now make sure to allow access to the previously created users from your `barmanhost`. On `pghost`, add these HBA rules to your `pg_hba.conf` file:

```
# allows access to the barman user from barmanhost
host    all    barman    barmanhost/32    md5
# allows access to the streaming_barman user from barmanhost
host    replication    streaming_barman    barmanhost/32    md5
```

Then, reload your Postgres configuration so the new HBA rules take effect. On `pghost`, run:

```
psql -c "SELECT pg_reload_conf();"
```

3. Still on `pghost`, make sure your Postgres server is properly configured for WAL streaming. On its `postgresql.conf` file, assert that `wal_level` is set to `replica` or `logical`:

```
wal_level = replica
```

If changes were made to the `wal_level` configuration value, then restart your Postgres server for the changes to take effect.

4. Now let's configure your first backup server on Barman. On `barmanhost`, create a file at `/etc/barman.d/streaming-backup-server.conf` with this content:

```
[streaming-backup-server]
description = "Postgres server using streaming replication"
streaming_archiver = on
backup_method = postgres
streaming_conninfo = host=pghost user=streaming_barman dbname=postgres
slot_name = barman
create_slot = auto
conninfo = host=pghost user=barman dbname=postgres
```

Where:

- `[streaming-backup-server]` is a name of your choice for your backup server on Barman.
- `description` is a description text for your backup server.
- `streaming_archiver = on` tells Barman that WAL files of this backup server are transferred from Postgres to Barman using streaming replication.
- `backup_method = postgres` tells Barman that this server uses `postgres` as its backup method, which in essence means taking backups using `pg_basebackup`.

- `streaming_conninfo` is a connection string for a *libpq* connection to your Postgres server. This is the connection `pg_receivewal` and `pg_basebackup` use to transfer files to your Barman server.
- `slot_name` is the name of the physical replication slot in Postgres which is used by this backup server to stream WALs through `pg_receivewal`.
- `create_slot = auto` tells Barman that it should create the replication slot automatically in Postgres, not requiring a manual creation beforehand.
- `conninfo` is a connection string for a *libpq* connection to your Postgres server which Barman uses for maintenance purposes.

On `barmanhost`, run:

```
barman list-servers
```

You should see an output with all configured backup servers on Barman, which confirms that it's now aware of your new server:

```
streaming-backup-server - Postgres server using streaming replication
```

5. Once finished with the configuration of both Barman and Postgres servers, you should be ready to go! Execute the following command on `barmanhost` to check that everything is OK with your server:

```
barman check streaming-backup-server
```

If you see a failed check related to WAL archive, don't worry. It just means that Barman has not received any complete WAL file yet, probably because no WAL segment has been switched on your Postgres server since the server was first created. You can force a WAL switch from `barmanhost` with this command:

```
barman switch-wal --force streaming-backup-server
```

Also, if you see failed checks related to replication slot and `pg_receivewal`, run the following command.

```
barman cron
```

This command starts a background process that performs maintenance tasks on your Barman servers. These tasks includes the creation of the replication slot in Postgres, if `create_slot = auto`, as well as starting up of `pg_receivewal` process.

Run the check command again and make sure no failed checks are shown:

```
barman check streaming-backup-server
```

This server is now ready to take backups and receive WAL files from your Postgres server. You may go to the *taking your first backup* section now.

3.1.2 Rsync backups with WAL archiving

This strategy relies on Rsync and SSH connections for transferring backup and WAL files to your Barman server.

Since it depends on SSH connections, it is therefore required that you have a two-way passwordless SSH connection between `pghost` and `barmanhost`. For further instructions on how to set this, please refer to the *pre-requisites* section.

1. As a first step, let's create the required user you will need on your Postgres server. On `pghost`, execute the following command:

```
createuser -s -P barman
```

This command creates a new Postgres superuser called **barman**, which will be used by Barman for maintenance tasks as well as for issuing backup commands using the Postgres low-level API. Alternatively you can create a user without superuser privileges, but with the necessary permissions to perform the needed operations by following the recipe in *Postgres users pre-requisite*.

The `createuser` command prompts you for a password, which you are then advised to add to a `password` file named `.pgpass` under your Barman home directory on `barmanhost`.

From now on, this section assumes you already have this Postgres user created as well as a Postgres server to be backed up. Also, we assume a database named `postgres` is available, so Barman can connect to the Postgres server through that database.

2. Now make sure to allow access to the previously created user from your `barmanhost`. On `pghost`, add this HBA rule to your `pg_hba.conf` file:

```
# allows access to the barman user from barmanhost
host    all    barman    barmanhost/32    md5
```

Then, reload your Postgres configuration so the new HBA rule takes effect. On `pghost`, run:

```
psql -c "SELECT pg_reload_conf();"
```

3. Still on `pghost`, make sure your Postgres server is properly configured for WAL archiving. On its `postgresql.conf` file, assert the following parameters are properly set:

```
wal_level = replica
archive_mode = on
archive_command = 'barman-wal-archive barmanhost rsync-backup-server %p'
```

Note

Since Barman 2.6, the recommended way of archiving WAL files via the `archive_command` is by using the `barman-wal-archive` utility, as in the example above. For this utility to be available, make sure to also have the `barman-cli` package installed on `pghost`. Check the *WAL archiving via archive_command* section for further details and for alternative command options.

4. Now let's configure your first backup server on Barman. On `barmanhost`, create a configuration file at `/etc/barman.d/rsync-backup-server.conf` with this content:

```
[rsync-backup-server]
description = "Postgres server using Rsync and WAL archiving"
archiver = on
backup_method = rsync
reuse_backup = link
backup_options = concurrent_backup
ssh_command = ssh postgres@pghost
conninfo = host=pghost user=barman dbname=postgres
```

Where:

- `[rsync-backup-server]` is a name of your choice for your backup server on Barman.
- `description` is a description text for your backup server.

- `archiver = on` tells Barman that WAL files of this backup server are transferred from Postgres to Barman using the `archive_command` configured in Postgres.
- `backup_method = rsync` tells Barman that this backup server uses `rsync` as its backup method, which in essence means copying over cluster files with `Rsync`.
- `reuse_backup = link` tells Barman that you want to have data deduplication by reusing files of the previous backup, saving storage and network resources whenever taking new backups for this server. Check [rsync backups](#) section for more details.
- `backup_options = concurrent_backup` indicates that Barman is going to issue non-exclusive backup commands on your Postgres server when taking backups.
- `ssh_command` is the SSH command to be used to connect from `barmanhost` to `pghost`. Replace this configuration value accordingly.
- `conninfo` is a connection string for a [libpq](#) connection to your Postgres server which Barman uses for maintenance purposes.

On `barmanhost`, run:

```
barman list-servers
```

You should see an output with all configured backup servers on Barman, which confirms that it's now aware of your new server:

```
rsync-backup-server - Postgres server using Rsync and WAL archiving
```

5. Once finished with the configuration of both Barman and Postgres servers, you should be ready to go! Execute the following command on `barmanhost` to check that everything is OK with your server:

```
barman check rsync-backup-server
```

If you see a failed check related to WAL archive, don't worry. It just means that Barman has not received any WAL files yet, probably because no WAL segment has been switched on your Postgres server since the server was first created. You can force a WAL switch from `barmanhost` with this command:

```
barman switch-wal --force rsync-backup-server
```

Then execute the following command, which starts a background process that performs maintenance tasks on your Barman servers:

```
barman cron
```

Run the check command again and make sure no failed checks are shown:

```
barman check rsync-backup-server
```

This server is now ready to take backups and receive WAL files from your Postgres server. You may go to the [taking your first backup](#) section now.

3.2 Taking your first backup

Regardless of which strategy you choose for your backup server, once completed with the previous steps, you should be all set. You can run this command to take a backup:

```
barman backup --name first-backup <server_name>
```

Once the command finishes, you can list all backups of your backup server with this command:

```
barman list-backups <server_name>
```

And show the details of a specific backup with this command:

```
barman show-backup <server_name> first-backup
```

3.3 Restoring a backup

If you ever need to restore a backup, you can do so with this command:

```
barman restore <server_name> first-backup /path/to/restore
```

If restoring to a remote server, a passwordless SSH connection from the Barman host to the destination host is required and its SSH command must be specified using the `--remote-ssh-command` option:

```
barman restore --remote-ssh-command="ssh user@host" <server_name> first-backup /path/to/  
↪ restore
```

Architectural backup designs with barman

An effective disaster recovery plan begins with a carefully designed architecture. Key decisions involve determining where to host your backup server and how to transfer backups and WAL files, among other considerations. With that in mind, this section explores a few different setups for deploying and managing backups with Barman.

4.1 Where to install Barman

One of the foundations of Barman is the ability to operate remotely from the database server, via the network.

Theoretically, you could have your Barman server located in a data center in another part of the world, thousands of miles away from your Postgres server. Realistically, you do not want your Barman server to be too far from your Postgres server, so that both backup and recovery times are kept under control.

Even though there is no "one size fits all" way to set up Barman, there are a couple of recommendations that we suggest you abide by, in particular:

- Install Barman on a dedicated server.
- Do not share the same storage with your Postgres server.
- Integrate Barman with your infrastructure monitoring tools.
- Test everything before deploying to production.

A good way to start modeling your disaster recovery architecture includes:

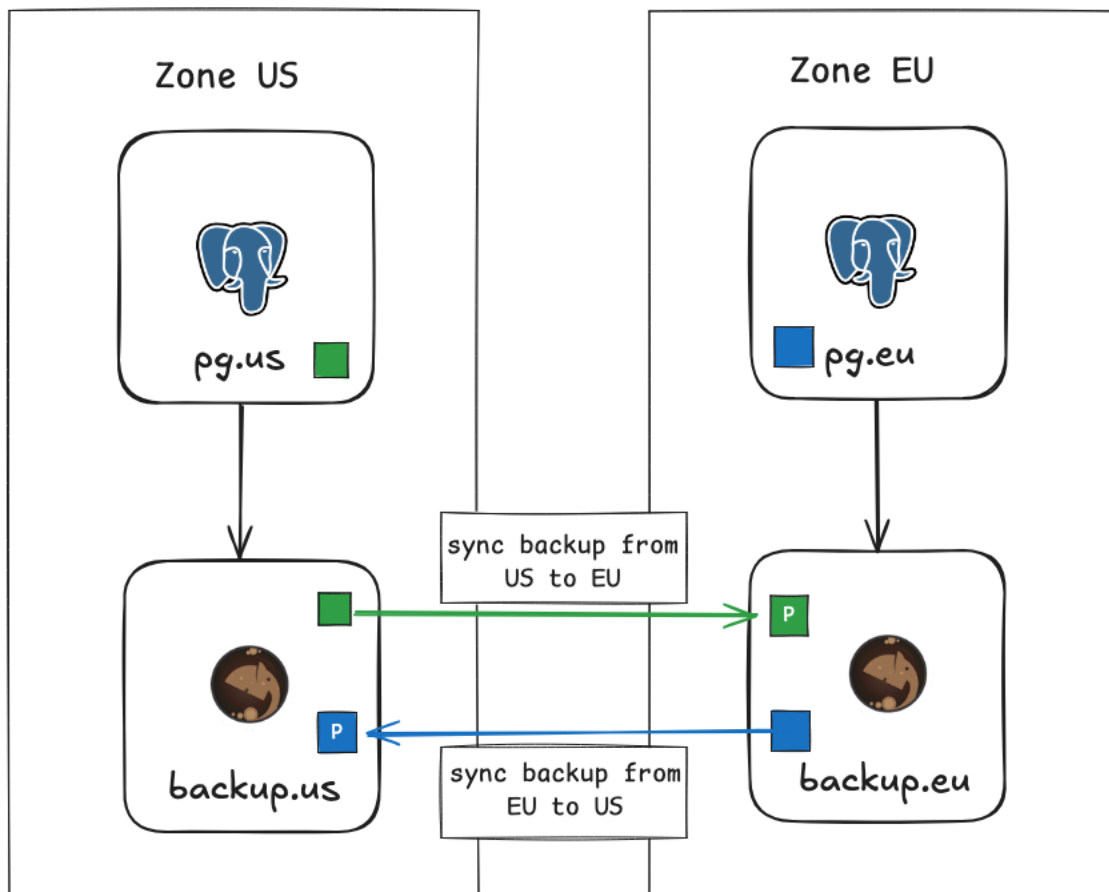
- Design a couple of possible architectures in regards to Postgres and Barman locations, such as:
 1. Same data center.
 2. Different data centers in the same metropolitan area.
 3. Different data centers in different metropolitan areas.
- Elaborate the pros and the cons of each hypothesis.
- Evaluate the *SPOF* of your system, with a cost-benefit analysis.
- Make your decision and implement an initial solution.

With that in mind, a very common setup for Barman is for it to be installed in the same data center where your Postgres servers are, in which case the *SPOF* is the data center itself. Though the impact of such *SPOF* can be significantly alleviated with features such as *geographical redundancy* (introduced in Barman 2.6) and *Hook Scripts*.

With geographical redundancy, you can rely on a Barman instance that is located in a different data center/availability zone to synchronize the entire content of the Barman server co-located in the same data center as the Postgres node. Also, given that geo-redundancy can be configured in Barman not only at global level, but also at server level, you can create hybrid installations of Barman where some servers are directly connected to the local Postgres servers, and others are backing up subsets of different Barman installations (cross-site backup). Figure below shows two availability zones (one in Europe and one in the US), each with a primary Postgres server that is backed up in a local Barman installation, and relayed on the other Barman server (defined as passive) for multi-tier backup via Rsync/SSH. Further information on geo-redundancy is available in the *geographical redundancy* section.

Backup architecture for Postgres

Multi-location geographical redundancy



Thanks to *Hook Scripts*, backups of Barman can be exported to different media, such as tape via tar, or locations, like an object storage bucket in a cloud provider.

Remember that no decision is forever. You can start one way and adapt over time to the solution that suits you best. However, try and keep it simple to start with.

4.2 One Barman, many Postgres servers

Another relevant feature first introduced by Barman is the support for multiple servers. Barman can store backup data coming from multiple Postgres instances, even with different versions, in a centralized way. As a result, you can model complex disaster recovery architectures, forming a "star schema", where Postgres servers rotate around a central Barman server.

Every architecture makes sense in its own way. Choose the one that resonates with you, and most importantly, the one you trust, based on real experimentation and testing.

4.3 Backup strategies

The choice of a backup strategy will also play a vital role in your setup. Barman is able to take backups using either Rsync, which uses SSH as a transport mechanism, or `pg_basebackup`, which uses Postgres streaming replication protocol. Choosing one of these two methods is a decision you will need to make, however for general usage we recommend using streaming replication for all currently supported versions of Postgres.

Note

Because Barman makes use of `pg_basebackup` when using streaming backups, features such as parallel backup are currently not available. In this case, bandwidth limitation has some restrictions - compared to the traditional method via Rsync. In Postgres versions prior to 17, incremental backups are also not available when using this method.

Backup using Rsync/SSH is recommended in cases where `pg_basebackup` limitations pose an issue for you.

The reason why we recommend streaming backup is that, based on our experience, it is easier to set up. Also, streaming backup allows you to backup a Postgres server on Windows, and makes life easier when working with Docker.

4.4 WAL archiving strategies

Recovering a Postgres backup relies on replaying transaction logs (also known as xlog or WAL files). It is therefore essential that WAL files be stored by Barman alongside the base backups so that they are available at recovery time. This can be achieved using either WAL streaming or standard WAL archiving to copy WALs into the Barman server.

1. WAL streaming involves transferring WAL files from the Postgres server with `pg_receivewal` using the Postgres streaming replication protocol. With WAL streaming, WALs are transferred while they are still being generated, which means that Barman doesn't have to wait for WAL segments to be completely filled in order to receive them. Such mechanism makes WAL streaming able to significantly reduce the risk of data loss, bringing *RPO* down to near zero values. It is also possible to add Barman as a synchronous WAL receiver in your Postgres cluster and achieve zero data loss (*RPO*=0). With the use of replication slots, we can also assure that no WAL file is recycled before being successfully received by Barman.

Refer to the *pre-requisites for wal streaming* for more information on how to install `pg_receivewal`.

Note

When using WAL streaming, it is recommended to always stream from the primary node. This is to ensure that all WALs are received by Barman, even in the event of a failover.

2. Barman also supports standard WAL file archiving, which is achieved using the Postgres `archive_command`, either using Rsync/SSH or `barman-wal-archive` from the `barman-cli` package. With this method, WAL files are archived

only when Postgres switches to a new WAL file, which normally happens every 16MB worth of data changes. This approach offers more flexibility by allowing you to pick a tool of your choice for transferring the WAL files.

It is required that either WAL streaming or WAL archiving be configured. It is optionally possible to configure both WAL streaming and standard WAL archiving - in such cases Barman will automatically de-duplicate incoming WALs. This provides a fallback mechanism so that WALs are still copied to Barman's archive in the event that WAL streaming fails.

For general usage we recommend configuring WAL streaming only.

Note

Previous versions of Barman recommended that both WAL archiving and WAL streaming were used. This was because Postgres versions older than 9.4 did not support replication slots and therefore WAL streaming alone could not guarantee all WALs would be safely stored in Barman's WAL archive. Since all supported versions of Postgres now have replication slots, it is sufficient to configure only WAL streaming.

4.5 Two typical scenarios for backups

In order to make life easier for you, in this section we summarize the two most typical scenarios for a given Postgres server in Barman. Bear in mind that this is a decision that you must make for every single server that you decide to back up with Barman. This means that you can have heterogeneous setups within the same Barman server.

We will be using `pg` and `backup` to refer to a Postgres and Barman servers respectively. However, in real life, your architecture will most likely contain other technologies such as `repmgr`, `pgBouncer`, `Nagios/Icinga`, and so on.

4.5.1 Scenario 1: Backup via streaming protocol

As stated in *Streaming Backups*, this approach uses the Postgres streaming protocol for transferring cluster files to your Barman server. This is done with the use of the `pg_basebackup` utility. In Barman, this method can be set by having `backup_method = postgres` in your Barman server configurations.

With this approach, you can leverage from *block-level incremental backups* support provided by `pg_basebackup`, available in Postgres 17 or later. Block-level incremental backups tend to be much more efficient than *file-level incremental backups* provided by Rsync strategies in terms of deduplication ratio.

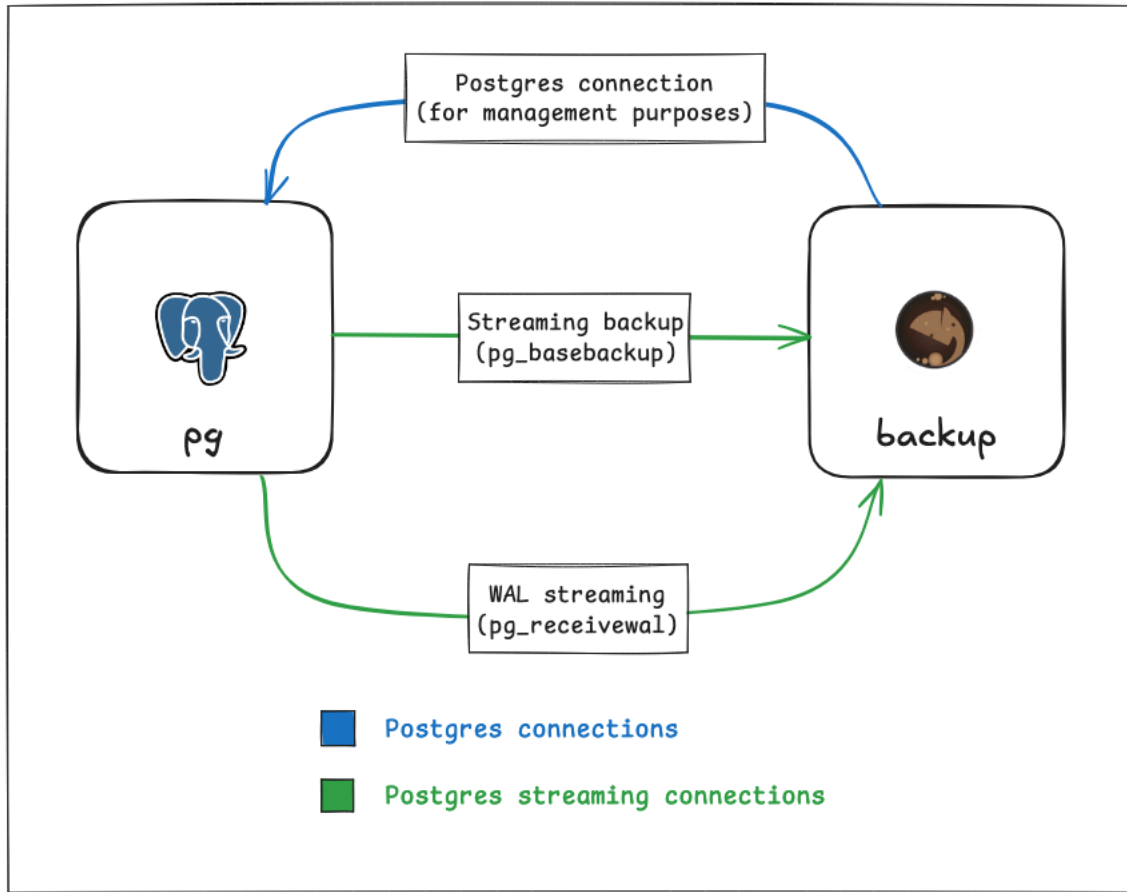
This method is used in conjunction with WAL streaming for WAL files. In Barman's terminology, this setup is known as streaming-only setup as it does not use any SSH connection for backup and archiving operations. This is particularly suitable and extremely practical for Docker environments and highly regulated environments, for example.

The streaming backup method is usually the recommended approach for most use cases.

The figure below illustrates how this setup would function in practice.

Backup architecture for Postgres

Full streaming architecture



In order to configure it, you need:

1. A standard connection to Postgres, for management, coordination, and monitoring purposes.
2. A streaming replication connection to be used by both `pg_basebackup` (for base backup operations) and `pg_receivewal` (for WAL streaming).

4.5.2 Scenario 2: Backup via rsync/SSH

As stated in *rsync backups* concepts, this approach relies on Rsync to transfer backup files to your Barman server. This is done by putting your server in backup mode and transferring your cluster files using Rsync.

A key advantage in this approach is the possibility of using *parallel jobs* when running backup operations, which can significantly decrease the overall time to take backups. It also provides the ability to take *file-level incremental backups*, which reuses files of a previous backup for deduplication. File-level incremental backups can be more flexible than *block level incremental backups* as each backup is completely independent of the others, which means you can delete a root backup without affecting its incremental backups in any way.

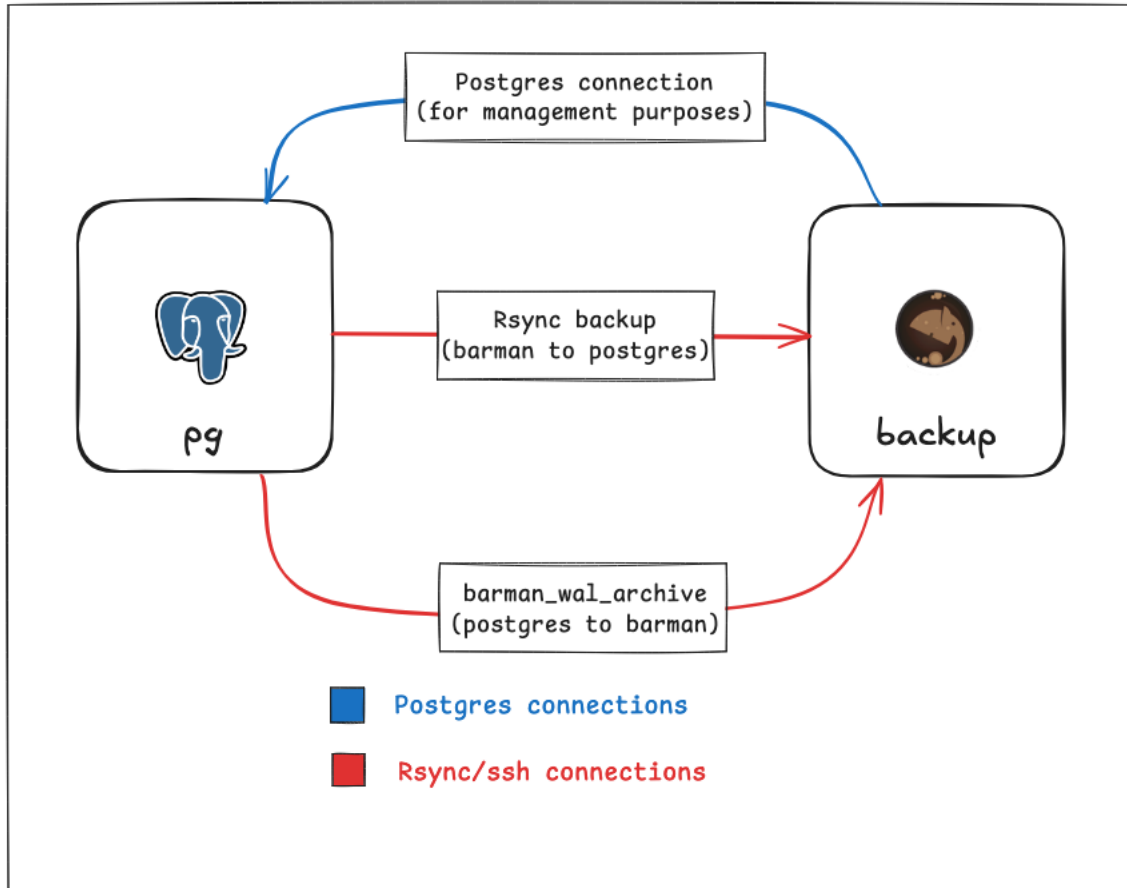
Another advantage of this method is that it allows for a finer control over bandwidth usage, including on a per-tablespace

basis. You can check *Managing Bandwidth Usage* for further details.

The figure below illustrates how this setup would function in practice.

Backup architecture for Postgres

Remote copy architecture



In order to configure it, you will need:

1. A standard connection to Postgres for management, coordination, and monitoring purposes.
2. An SSH connection to be used by Rsync for base backup operations that allow the **barman** user on the Barman server to connect as the **postgres** user on the Postgres server.
3. An SSH connection for WAL archiving to be used by the `archive_command` in Postgres that allows the **postgres** user on the Postgres server to connect as **barman** user on the Barman server.

4.5.3 Hybrid scenario

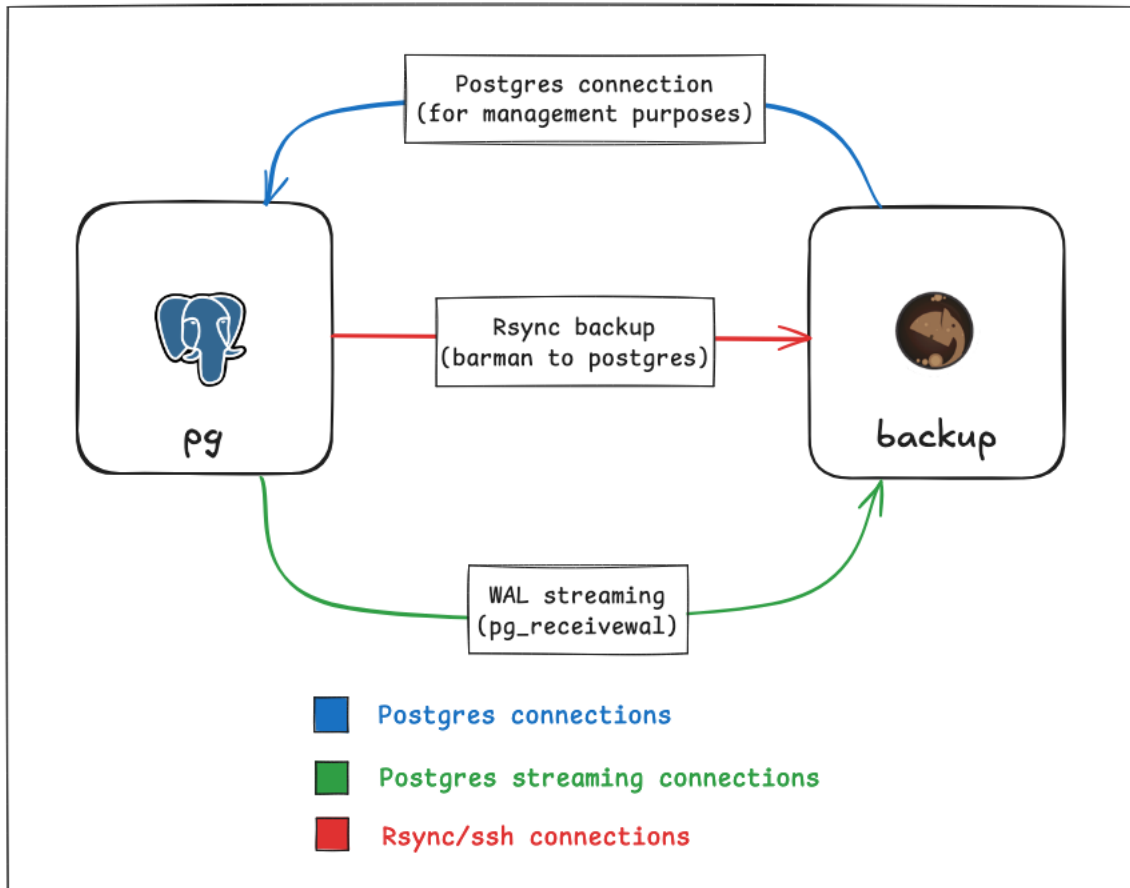
It is also possible to use a hybrid approach, combining rsync backups with WAL streaming in order to achieve results that have the advantages of rsync backups (file-level incremental backups, parallel jobs, etc) together with WAL streaming ones (more efficient WAL transfer, optional RPO zero).

The figure below illustrates how this setup would function in practice.

Backup architecture for Postgres

Frame

Rsync backup with WAL streaming architecture



To accomplish this, you will need to configure the `backup_method` as `rsync`, and set `streaming_archiver` to `on` in your Barman server configuration. You will also need to have a streaming replication connection to be used by `pg_receivewal` for WAL archiving and an SSH connection to be used by Rsync for base backup operations.

4.5.4 WAL archiving fallback redundancy

With Barman, you can configure WAL archiving in addition to WAL streaming in order to have a fallback mechanism in case WAL streaming fails. This can be done with either of the `backup_method` configurations described above.

1. When using the streaming-only setup, described in the [Scenario 1](#), you can also configure WAL archiving via SSH in addition to WAL streaming. In such scenarios, WAL archiving would act as a fallback mechanism in case WAL streaming failed.
2. When using the Rsync backup method, described in the [Scenario 2](#), you can also configure WAL streaming instead of using the `archive_command` in order to have a lower *RPO*. You can also opt for configuring WAL streaming in addition to WAL archiving and have both options.

In either cases, Barman will automatically verify that the WAL files are not duplicated in the archive, and will only store them once.

4.6 Geographical redundancy

A very common setup for Barman is to have it installed in the same data center where your PostgreSQL servers are. In this case, the single point of failure is the data center. Fortunately, the impact of such a *SPOF* can be alleviated thanks to two features that Barman provides to increase the number of backup tiers:

1. geographical redundancy (introduced in Barman 2.6)
2. hook scripts

With geographical redundancy, you can rely on another Barman instance that is located in a different data center/availability zone, and synchronize the entire content of the primary Barman server.

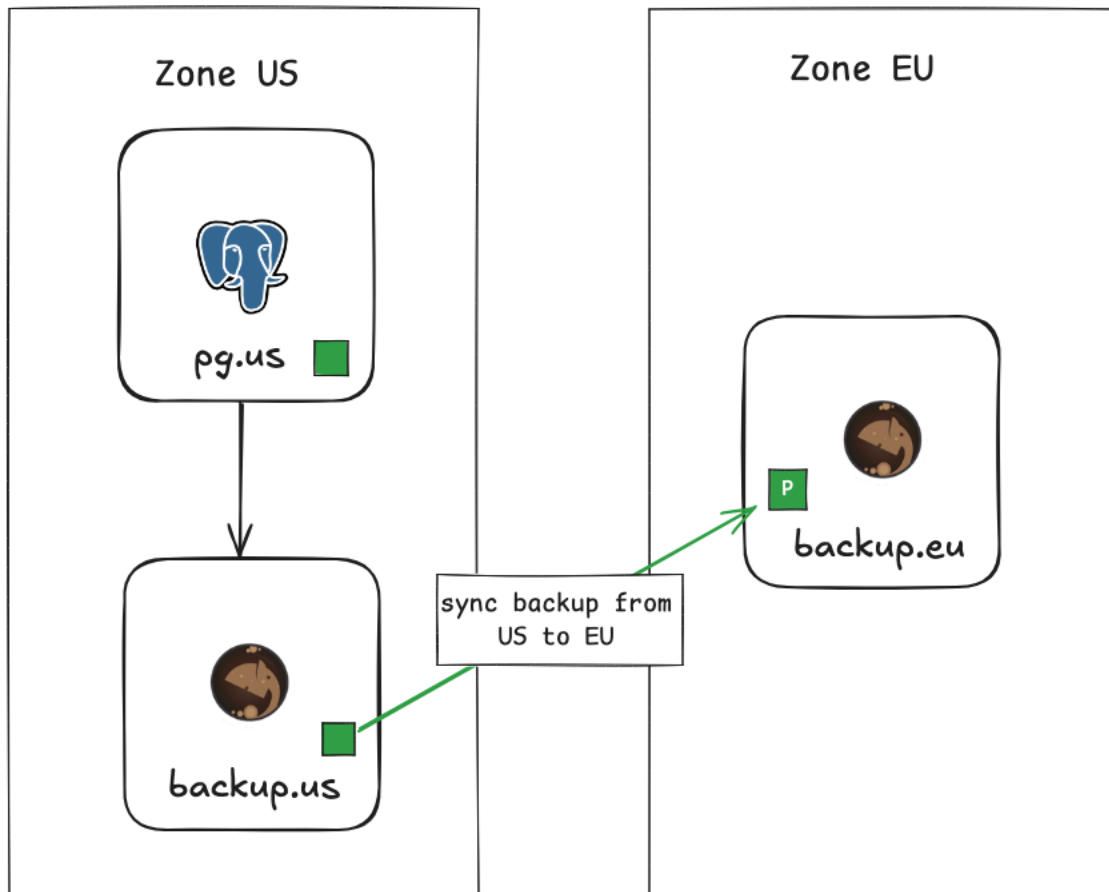
There's more: given that geo-redundancy can be configured in Barman not only at global level, but also at server level, you can create hybrid installations of Barman where some servers are directly connected to the local PostgreSQL servers, and others are backing up subsets of different Barman installations (cross-site backup).

Figure below shows two availability zones (one in Europe and one in the US), each with a primary PostgreSQL server that is backed up in a local Barman installation, and relayed on the other Barman server (defined as passive) for multi-tier backup via rsync/SSH. Further information on geo-redundancy is available in the *Geographical Redundancy* section.

The image below illustrates how this setup would function in practice.

Backup architecture for Postgres

Geographical redundancy



4.7 Cloud snapshot backups

Barman also supports cloud snapshot backups, which takes a snapshot of the storage volume where your Postgres server resides in the cloud. Barman currently supports this method on Azure, Google, and AWS. The prerequisites for this method will depend on which cloud provider where your Postgres server resides, so we recommend checking the [Cloud Snapshot Backups](#) section for further details.

CHAPTER 5

Pre-requisites

This section details some requirements and configurations necessary to set up a Barman environment depending on your use case.

Throughout this section, we assume the following hosts:

- **pghost**: The host where Postgres is running.
- **barmanhost**: The host where Barman will be set up.

5.1 Postgres users

Barman requires a connection to your Postgres instance to gather information about the server. The recommended way to set up this connection is to create a dedicated user in Postgres named **barman**. This user should have the necessary privileges.

Note

The **createuser** commands executed below will prompt you for a password, which you are then advised to add to a [password file](#) named **.pgpass** under your Barman home directory on **barmanhost**. Additionally, you can choose the client authentication method of your preference among those offered by Postgres. Check the [official documentation](#) for further details.

To create a superuser named **barman** in Postgres, run the following command on **pghost**:

```
createuser -s -P barman
```

Or, in case you opt for a user with only the required privileges, follow these steps:

1. On **pghost**, run this command to create a user named **barman** in Postgres:

```
createuser -P barman
```

2. On **pghost**, in the **psql** interface, run the following statements:

```
GRANT EXECUTE ON FUNCTION pg_backup_start(text, boolean) TO barman;
GRANT EXECUTE ON FUNCTION pg_backup_stop(boolean) TO barman;
GRANT EXECUTE ON FUNCTION pg_switch_wal() TO barman;
GRANT EXECUTE ON FUNCTION pg_create_restore_point(text) TO barman;
GRANT pg_read_all_settings TO barman;
GRANT pg_read_all_stats TO barman;
```

In the case of using Postgres version 14 or a prior version, the functions `pg_backup_start` and `pg_backup_stop` will have different names and signatures. You will therefore need to replace the first two lines in the above block with:

```
GRANT EXECUTE ON FUNCTION pg_start_backup(text, boolean, boolean) TO barman;
GRANT EXECUTE ON FUNCTION pg_stop_backup() TO barman;
GRANT EXECUTE ON FUNCTION pg_stop_backup(boolean, boolean) TO barman;
```

Note

In Postgres version 13 and below, the `--force` option of the `barman switch-wal` command does not work without a superuser. In Postgres version 15 or above, it is possible to grant the `pg_checkpoint` role to use it without a superuser by executing the statement `GRANT pg_checkpoint TO barman;`.

5.2 Postgres connection

A connection to your Postgres instance is required regardless of which backup method you are using. This connection is required by Barman in order to coordinate its activities with the database server, as well as for monitoring purposes.

Make sure that `barmanhost` can connect to the database server as superuser or with a user with the required privileges. You can find detailed information about setting up Postgres connections in the [Postgres Client Authentication](#).

With your user created, run the following command on `barmanhost` to assert that it can connect to your Postgres instance:

```
psql -c 'SELECT version()' -U barman -h pghost postgres
```

`postgres` can be any available database through which Barman can connect to the Postgres instance.

If the above command succeeds, it means that it can successfully connect to the database server. Remember the connection parameters above as they are the ones you need to write on your server's configuration file, in the `conninfo` parameter. In this context, this parameter would be as follows:

```
[my-server]
; ...
conninfo = host=pghost user=barman dbname=postgres application_name=myapp
```

`application_name` is an optional parameter.

5.3 Postgres client tools

The Postgres client tools are required to interact with the Postgres server. The most commonly used tools by Barman are `pg_basebackup` and `pg_receivewal`. They are provided by the Postgres client package.

To install the Postgres client package on Debian or Ubuntu run the following command on the `barmanhost`:

```
sudo apt-get install postgresql-client
```

Alternatively, if the barmanhost is using RHEL, Rocky Linux, Alma Linux, follow this recipe:

```
sudo dnf install postgresql
```

5.4 Postgres streaming replication connection

If you plan to use streaming backups or streaming of WAL files, you need to setup a streaming connection. Additionally, you also need to have the Postgres client tools installed, as shared in [pre-requisites](#) section.

We recommend creating a dedicated user in Postgres named `streaming_barman`. You can do so with the following command:

```
createuser -P --replication streaming_barman
```

Note

The `createuser` commands executed below prompt you for a password, which you are then advised to add to a [password file](#) named `.pgpass` under your Barman home directory on `barmanhost`. Additionally, you can choose the client authentication method of your preference among those offered by Postgres. Check the [official documentation](#) for further details.

You can verify that the streaming connection works through the following command:

```
psql -U streaming_barman -h pghost -c "IDENTIFY_SYSTEM" replication=1
```

If the connection is working, you should see a response containing the system identifier, current timeline ID and current WAL flush location, for example:

systemid	timeline	xlogpos	dbname
7139870358166741016	1	1/330000D8	

(1 row)

You also need to configure the `max_wal_senders` parameter in Postgres. The number of WAL senders depends on the Postgres architecture you have implemented. In this example, we are setting it to 2:

```
max_wal_senders = 2
```

This option represents the maximum number of concurrent streaming connections that Postgres is allowed to manage.

Another important parameter is `max_replication_slots`, which represents the maximum number of replication slots that Postgres is allowed to manage. This parameter is relevant if you are planning to use the streaming connection to receive WAL files over the streaming connection:

```
max_replication_slots = 2
```

The values proposed for `max_replication_slots` and `max_wal_senders` must be considered as examples, and the values you use in your actual setup must be chosen after a careful evaluation of the architecture. Please consult the Postgres documentation for guidelines and clarifications.

5.5 SSH connections

If you plan to use Rsync backups or WAL archiving via `archive_command`, then SSH connections are required.

SSH is a protocol and a set of tools that allows you to open a remote shell to a remote server and copy files between the server and the local system. You can find more documentation about SSH usage in the [article "SSH Essentials" by Digital Ocean](#).

SSH key exchange is a very common practice that is used to implement secure passwordless connections between users on different machines, and it's needed to use Rsync for WAL archiving and backups.

5.5.1 SSH configuration of postgres user

Unless you have done it before, you need to create an SSH key for the **postgres** user. Log in as **postgres** on **pghost** and run:

```
ssh-keygen -t rsa
```

As this key must be used to connect from hosts without providing a password, no passphrase should be entered during the key pair creation.

5.5.2 SSH configuration of barman user

You also need to create an SSH key for the **barman** user. Log in as **barman** on **barmanhost** and run:

```
ssh-keygen -t rsa
```

Again, no passphrase should be entered.

5.5.3 From Postgres to Barman

The SSH connection from **pghost** to **barmanhost** is needed to correctly archive WAL files using the `archive_command`.

To successfully connect from **pghost** to **barmanhost**, the **postgres** user's public key has to be stored in the authorized keys of the **barman** user on **barmanhost**. This key is located in the **postgres** user home director in a file named `.ssh/id_rsa.pub`, and its content should be included in a file named `.ssh/authorized_keys` inside the home directory of the **barman** user on **barmanhost**. If the `authorized_keys` file doesn't exist, create it using `600` as permissions.

The following command should succeed without any output if the SSH key pair exchange has been completed successfully:

```
ssh barman@barmanhost -C true
```

5.5.4 From Barman to Postgres

The SSH connection between from **barmanhost** to **pghost** is used for the traditional backup using Rsync.

To successfully connect from **barmanhost** to **pghost**, the **barman** user's public key has to be stored in the authorized keys of the **postgres** user on **pghost**. This key is located in the **barman** user home directory in a file named `.ssh/id_rsa.pub`, and its content should be included in a file named `.ssh/authorized_keys` inside the home directory of the **postgres** user on **pghost**. If the `authorized_keys` file doesn't exist, create it using `600` as permissions.

The following command should succeed without any output if the SSH key pair exchange has been completed successfully:

```
ssh postgres@pghost -C true
```

5.6 WAL archiving via `archive_command`

As stated in the [WAL archiving strategies](#) section, there are two options to archive wals with Barman. If you wish to use the streaming replication protocol to archive WAL files, refer to the [WAL streaming](#) concepts and [Quick start](#) section, specifically the Streaming backups with WAL streaming sub-section. Otherwise you can configure WAL archiving using the `archive_command` with `commands-barman-cli-barman-wal-archive` or with Rsync/SSH.

5.6.1 Using `barman-wal-archive`

Starting from Barman 2.6, the recommended approach for securely archiving Write-Ahead Log files is to utilize the `barman-wal-archive` command from the `barman-cli` package. Refer to the [installation](#) section on how to install this package.

Using `barman-wal-archive` instead of traditional methods like `rsync` or `SSH` minimizes the risk of data corruption during the transfer of WAL files to the Barman server. The conventional methods lack a guarantee that the file's content is properly flushed and fsynced to disk at the destination.

The `barman-wal-archive` utility directly interacts with `commands-barman-put-wal` command. This command ensures that the received WAL file is fsynced and stored in the correct incoming directory for the respective server. The only parameter required for the `archive_command` is the server's name, reducing the likelihood of misplacement.

Note

The `barman-wal-archive` client is designed to minimize failures and prevent WAL files from accumulating on the PostgreSQL server. If a duplicate WAL file is sent to the Barman host, the command will exit with a success code (0) and log a message.

If the content of the duplicate file matches the one already stored on the Barman host, the file is ignored. However, if the content differs, the file is moved to the errors directory on the Barman server. This will cause future `barman check` executions to report a failure due to the mismatch.

To verify that `barman-wal-archive` can connect to the Barman server and that the Postgres server is correctly configured to accept incoming WAL files, execute the following command:

```
barman-wal-archive --test backup pg DUMMY
```

Here, `backup` refers to the Barman host, `pg` is the Postgres server's name as configured in Barman, and `DUMMY` is a placeholder for the WAL file name which is ignored when using the `-t` option.

If the setup is correct, you should see:

```
Ready to accept WAL files for the server pg
```

Since the utility communicates via `SSH`, ensure that `SSH` key authentication is set up for the `postgres` user to log in as `barman` on the backup server. If your `SSH` connection uses a port other than the default (22), you can specify the port using the `--port` option.

Refer to the [Rsync backups with WAL archiving](#) to start working with it.

5.6.2 Using Rsync/SSH

An **alternative approach** for configuring the `archive_command` is to utilize the `rsync` command via SSH. Here are the initial steps to set it up effectively for a Postgres server named `pg`, a Barman server named `backup` and a user named `barman`.

To locate the incoming WALs directory, use the following command and check for the `incoming_wals_directory` value:

```
barman show-servers pg | grep incoming_wals_directory

incoming_wals_directory: /var/lib/barman/pg/incoming
```

Next, edit the `postgresql.conf` file for the Postgres instance on the `pg` host to enable archive mode:

```
archive_mode = on
wal_level = 'replica'
archive_command = 'rsync -a %p barman@backup:INCOMING_WALS_DIRECTORY/%f'
```

Be sure to replace the `INCOMING_WALS_DIRECTORY` placeholder with the actual path retrieved from the previous command. After making these changes, restart the Postgres server.

For added security in the `archive_command` process, consider implementing stricter checks. For instance, the following command ensures that the hostname matches before executing the `rsync`:

```
archive_command = 'test $(/bin/hostname --fqdn) = HOSTNAME \
&& rsync -a %p barman@backup:INCOMING_WALS_DIRECTORY/%f'
```

Replace `HOSTNAME` with the output from `hostname --fqdn`. This approach acts as a safeguard against potential issues when servers are cloned, preventing WAL files from being sent by recovered Postgres instances.

Barman check

The `check` command verifies the connection to a specified server and ensures the server configuration is coherent. This command performs a series of checks to validate the proper setup of SSH and Postgres connections are functioning, the existence of backup directories, and the integrity of the WAL archive. It also verifies Postgres configurations, retention policies, backup validity, and WAL archiving processes. It identifies and reports any detected issues, making it one of the most critical features provided by Barman.

Running this command will:

- Ensure that WAL archiving is correctly configured.
- Validate the Postgres connection and necessary privileges.
- Verify that all required directories exist.
- Check that retention policies are properly set.
- Confirm the validity of backups and WAL files.
- Detect configuration errors and archiver issues.

6.1 Usage

To use the `check` command, run:

```
barman check <server_name>
```

To run a check on all configured servers, use:

```
barman check all
```



Integrate the `check` command with your alerting and monitoring infrastructure to ensure continuous oversight of your backup environment. The `--nagios` option is useful for creating plugins for Nagios/Icinga, allowing you to monitor the status of your servers seamlessly. For example, you can run:

```
barman check --nagios <server_name>
```

This will output results in a format compatible with Nagios, facilitating integration into your existing monitoring setup.

Tip

For other backup and server-specific checks, you can use the following command:

- `barman check-backup`: Checks that all necessary WAL files for verifying the consistency of a physical backup are properly archived. This command is used by the `cron` job and is automatically executed after each backup operation. See `barman check-backup` for more details.

6.2 Understanding the output

Running `barman check` will display an output similar to the following:

```
$ barman check example
Server example:
  PostgreSQL: OK
  superuser or standard user with backup privileges: OK
  PostgreSQL streaming: OK
  wal_level: OK
  replication slot: OK
  directories: OK
  retention policy settings: OK
  backup maximum age: OK (no last_backup_maximum_age provided)
  compression settings: OK
  configuration files: OK
  pg_basebackup: OK
  pg_basebackup compatible: OK
  received WAL files: OK
  archiving: OK
  archive_mode: OK
  archive_command: OK
  continuous archiving: OK
  archive_timeout: OK
```

The output label can vary depending on the status of each check. For example, if a check fails, it will be marked as **FAILED** and may include a hint to help troubleshoot the issue. If a check passes but with warnings, it will be marked as **WARNING** with additional context.

The `barman check` command performs the checks for each of the following aspects of Barman functioning:

WAL Archive

- Ensures that WAL archiving is set up correctly.
- Checks the number of WAL files in the incoming and streaming directories.

Note

If `archiver = off` in the Barman configuration and there are WALs in the incoming directory, the check will fail. This happens because WALs in the incoming directory suggest Postgres is still using `archive_command` or that the user switched from `archiver` to `streaming_archiver`, leaving WALs unsaved in Barman's archive.

To resolve this, you must determine if the WALs in incoming are necessary or can be safely deleted. This can be done by:

1. Checking if the WALs are newer than the `begin_wal` of the oldest backup.
2. Verifying if these WALs are already in Barman's archive.

The same issue applies if `streaming_archiver = off` and WALs are found in the streaming directory.

Postgres Connection

- Validates the Postgres connection.
- Ensures that the server version is supported.
- Checks for necessary privileges and streaming support.

Local Tools Validity

- Ensures that local tools for taking backups and receiving WALs will work correctly with the version of the database server, such as `pg_basebackup` for taking backups, and `pg_receivewal` for streaming WAL files.

Directory

- Ensures that all necessary backup directories exist.
- Creates directories if they do not exist.

Retention Policy

- Validates the retention policy settings.

Backup Validity

- Ensures that the backup validity requirements are satisfied.
- Checks the maximum age and minimum size of backups.

WAL Validity

- Ensures that WAL archiving requirements are met.
- Checks the maximum age and size of WAL files.

Configuration

- Inspects the server's message list for error messages.
- Outputs any errors found.

Identity

- Verifies that the system ID retrieved from the streaming connection matches the one from the standard connection and the one stored on disk.

Archiver Errors

- Inspects the errors directory for the presence of archiving errors.

7.1 Overview

The backup command is used to backup an entire Postgres server according to the configuration file parameters. To use it, run:

```
barman backup [OPTIONS] SERVER_NAME
```

Note

For detailed information on the backup command, refer to the backup command reference.

Important

Any interaction you plan to have with Barman, you will have to assure that the server is correctly configured. Refer to *Quickstart* and *Configuration Reference* sections for the steps you need to cover before trying to create any backup.

Warning

Backup initiation will fail if WAL files are not correctly archived to Barman, either through the `archiver` or the `streaming_archiver` options.

Barman offers multiple backup methods for Postgres servers, each with its own approach and requirements.

Prior to version 2.0, Barman relied solely on rsync for both standard backups and file-level incremental backups. Streaming backups were introduced in this version. Starting with version 3.11, Barman also supports block-level incremental backups through the streaming connection.

Important

For Postgres 15 and higher, exclusive backups are no longer supported. The only method for taking backups is through concurrent backup. If `backup_options` is unset, Barman will automatically set it to `concurrent_backup`.

7.2 Incremental Backups

Incremental backups involve using an existing backup as a reference to copy only the data changes that have occurred since the last backup on the Postgres server.

The primary objectives of incremental backups in Barman are:

- Shorten the duration of the full backup process.
- Reduce disk space usage by eliminating redundant data across periodic backups (data deduplication).

Barman supports two types of incremental backups:

- File-level incremental backups (using `rsync`)
- Block-level incremental backups (using `pg_basebackup` with Postgres 17)

Note

Incremental backups of different types are not compatible with each other. For example, you cannot take a block-level incremental backup on top of an `rsync` backup, nor can you take a file-level incremental backup on top of a streaming backup created with `pg_basebackup`.

7.3 Managing Bandwidth Usage

You can control I/O bandwidth usage with the `bandwidth_limit` option (global or per server) by specifying a maximum rate in kilobytes per second. By default, this option is set to `0`, meaning there is no bandwidth limit.

If you need to manage I/O workload on specific tablespaces, use the `tablespace_bandwidth_limit` option (global or per server) to set limits for individual tablespaces:

```
tablespace_bandwidth_limit = tname:bwlimit[, tname:bwlimit, ...]
```

This option takes a comma-separated list of tablespace name and bandwidth limit pairs (in kilobytes per second).

When backing up a server, Barman will check for tablespaces listed in this option. If a matching tablespace is found, the specified bandwidth limit will be applied. If no match is found, the default bandwidth limit for the server will be used.

Important

The `bandwidth_limit` option is available with `rsync` and `postgres` backup methods, but the `tablespace_bandwidth_limit` option is only applicable when using `rsync`.

7.4 Network Compression

You can reduce the size of data transferred over the network by using network compression. This can be enabled with the `network_compression` option (global or per server):

```
network_compression = true | false
```

Important

The `network_compression` option is not available with the `postgres` backup method.

Setting this option to `true` will enable data compression for network transfers during both backup and recovery. By default, this option is set to `false`.

7.5 Backup Compression

Barman supports backup compression using the `pg_basebackup` tool. This feature can be enabled with the `backup_compression` option (global or per server).

Important

The `backup_compression` option, along with other options discussed here, is only available with the `postgres` backup method.

7.5.1 Compression Algorithms

Setting the `backup_compression` option will compress the backup using the specified algorithm. Supported algorithms in Barman are: `gzip`, `lz4`, `zstd`, and `none` (which results in an uncompressed backup).

```
backup_compression = gzip | lz4 | zstd | none
```

Barman requires the corresponding CLI utilities for the selected compression algorithm to be installed on both the Barman server and Postgres server. These utilities can be installed via system packages named `gzip`, `lz4`, and `zstd` on Debian, Ubuntu, RedHat, CentOS, and SLES systems.

- On Ubuntu 18.04 (bionic), the `lz4` utility is available in the `liblz4-tool` package.
- `lz4` and `zstd` are supported with Postgres 15 or higher.

Important

If using `backup_compression`, you must also set `recovery_staging_path` to enable recovery of compressed backups. Refer to the [Recovering Compressed backups](#) section for details.

7.5.2 Compression Workers

You can use multiple threads to speed up compression by setting the `backup_compression_workers` option (default is 0):

```
backup_compression_workers = 2
```

Note

This option is available only with `zstd` compression. `zstd` version must be 1.5.0 or higher, or 1.4.4 or higher with multithreading enabled.

7.5.3 Compression Level

Specify the compression level with the `backup_compression_level` option. This should be an integer value supported by the chosen compression algorithm. If not specified, the default value for the algorithm will be used.

- For none compression, `backup_compression_level` must be set to 0.
- The available levels and default values depend on the chosen compression algorithm. Check the [backup configuration options](#) section for details.
- For Postgres versions prior to 15, `gzip` supports only `backup_compression_level = 0`, which uses the default compression level.

7.5.4 Compression Location

For Postgres 15 or higher, you can choose where compression occurs: on the server or the client. Set the `backup_compression_location` option:

```
backup_compression_location = server | client
```

- `server`: Compression occurs on the Postgres server, reducing network bandwidth but increasing server workload.
- `client`: Compression is handled by `pg_basebackup` on the client side.

You can also specify the backup format using `backup_compression_format`:

```
backup_compression_format = plain | tar
```

- `plain`: `pg_basebackup` decompresses data before writing to disk.
- `tar`: Backups are written as compressed tarballs (default).

Note

If setting `backup_compression_location = server` and `backup_compression_format = plain`, you can reduce network usage given the files are compressed on the server side and decompressed on the client side. This can be useful when the network bandwidth is limited but CPU is not, and backups need to be stored uncompressed.

Depending on the chosen `backup_compression` and `backup_compression_format`, you may need to install additional tools on both the Postgres and Barman servers.

Refer to the table below to select the appropriate tools for your configuration.

backup_compression	backup_compression_f	Postgres	Barman
gzip	plain	tar	None
gzip	tar	tar	tar
lz4	plain	tar, lz4	None
lz4	tar	tar, lz4	tar, lz4
zstd	plain	tar, zstd	None
zstd	tar	tar, zstd	tar, zstd
none	tar	tar	tar

7.6 Backup Encryption

Barman supports encryption of both backups and WAL files. This feature can be enabled with the `encryption` option (global or per server).

7.6.1 Requirements

The current encryption implementation for backups relies on the `pg_basebackup` ability to take backups in tar format. To achieve that, you need to set your configuration as follows:

- `backup_method = postgres`
- `backup_compression = <compression_method>` (none for no compression)
- `backup_compression_format = tar`

The backed up tar files are encrypted immediately after `pg_basebackup` finishes writing them on the Barman server disk.

7.6.2 Encryption Methods

Setting the `encryption` option dictates the encryption method used for base backups and WALs. Currently, only `gpg` and `none` (no encryption) are accepted values.

Note

For details about WAL encryption, refer to *WAL encryption*.

Note

For details about decryption, refer to *Recovering Encrypted Backups*.

GPG

This method is enabled by setting `encryption = gpg` in the configuration file.

To use *GPG* for encryption, you need `gpg` version 2.1 or higher installed on the server. You must also generate a GPG key pair in advance and configure the `encryption_key_id` option with the ID or recipient's email of the generated public key. The corresponding private key must be present in GPG's keyring and secured with a strong passphrase.

7.7 Immediate Checkpoint

Before starting a backup, Barman requests a checkpoint, which can generate additional workload. By default, this checkpoint is managed according to Postgres' workload control settings, which may delay the backup.

You can modify this default behavior using the `immediate_checkpoint` configuration option (default is `false`).

If `immediate_checkpoint` is set to `true`, Postgres will perform the checkpoint at maximum speed without throttling, allowing the backup to begin as quickly as possible. You can override this configuration at any time by using one of the following options with the `barman backup` command:

- `--immediate-checkpoint`: Forces an immediate checkpoint.
- `--no-immediate-checkpoint`: Waits for the checkpoint to complete before starting the backup.

7.8 Streaming Backup

Barman can perform a backup of a Postgres server using a streaming connection with `pg_basebackup`.

Important

`pg_basebackup` must be installed on the Barman server. It is recommended to use the latest version of `pg_basebackup` as it is backwards compatible. Multiple versions can be installed and specified using the `path_prefix` option in the configuration file.

To configure streaming backups, set the `backup_method` to `postgres`:

```
backup_method = postgres
```

7.8.1 Block-level Incremental Backup

This type of backup uses the native incremental backup feature introduced in Postgres 17.

Block-level incremental backups deduplicate data at the page level in Postgres. This means only pages modified since the last backup need to be stored, which is more efficient, especially for large databases with frequent writes.

To perform block-level incremental backups in Barman, use the `--incremental` option with the `backup` command. You must provide a backup ID or shortcut referencing a previous backup (full or incremental) created with `backup_method=postgres` for deduplication. Alternatively, you can use `last-full` or `latest-full` to reference the most recent eligible full backup in the catalog.

Example command:

```
barman backup --incremental BACKUP_ID SERVER_NAME
```

To use block-level incremental backups in Barman, you must:

- Use Postgres 17 or later.
- This feature relies on WAL Summarization, so `summarize_wal` must be enabled on your database server before taking the initial full backup.
- Use `backup_method=postgres`.

Note

Compressed backups are currently not supported for block-level incremental backups in Barman.

Important

If you enable `data_checksums` between block-level incremental backups, it's advisable to take a new full backup. Divergent checksum configurations can potentially cause issues during recovery.

7.9 Backup with Rsync through SSH

Barman can perform a backup of a Postgres server using Rsync, which uses SSH as a transport mechanism.

To configure a backup using rsync, include the following parameters in the Barman server configuration file:

```
backup_method = rsync
ssh_command = ssh postgres@pg
```

Here, `backup_method` activates the rsync backup method, and `ssh_command` specifies the SSH connection details from the Barman server to the Postgres server.

Note

Starting with Barman 3.11, a keep-alive mechanism is used for rsync-based backups. This mechanism sends a simple `SELECT 1` query over the libpq connection to prevent firewall or router disconnections due to idle connections. You can control or disable this mechanism using the `keepalive_interval` configuration option.

7.9.1 File-Level Incremental Backups

File-level incremental backups rely on rsync and alternatively hard links, so both the operating system and file system where the backup data is stored must support these features.

The core idea is that during a subsequent base backup, files that haven't changed since the last backup are shared, which saves disk space. This is especially beneficial in *VLDB* and those with a high percentage of read-only historical tables.

You can enable rsync incremental backups through a global/server option called `reuse_backup`, which manages the Barman backup command. It accepts three values:

- **off**: Standard full backup (default).
- **link**: File-level incremental backup that reuses the last backup and creates hard links for unchanged files, reducing both backup space and time.
- **copy**: File-level incremental backup that reuses the last backup and creates copies of unchanged files, reducing backup time but not space.

Typically, you would set `reuse_backup` to `link` as follows:

```
reuse_backup = link
```

Setting this at the global level automatically enables incremental backups for all your servers.

You can override this setting with the `--reuse-backup` runtime option when running the Barman backup command. For example, to run a one-off incremental backup, use:

```
barman backup --reuse-backup=link <server_name>
```

Note

Unlike block-level incremental backups, rsync file-level incremental backups are self-contained. If a parent backup is deleted, the integrity of other backups is not affected. Deduplication in rsync backups uses hard links, meaning that when a reused backup is deleted, you don't need to create a new full backup; shared files will remain on disk until the last backup that used those files is also deleted. Additionally, using `reuse_backup = link` or `reuse_backup = copy` for the initial backup has no effect, as it will still be treated as a full backup due to the absence of existing files to link or copy.

7.10 Concurrent Backup of a Standby

When performing a backup from a standby server, ensure the following configuration options are set to point to the standby:

- `conninfo`
- `streaming_conninfo` (if using `backup_method = postgres` or `streaming_archiver = on`)
- `ssh_command` (if using `backup_method = rsync`)
- `wal_conninfo` (connecting to the primary if `conninfo` is pointing to a standby)

The `primary_conninfo` option should point to the primary server. Barman will use `primary_conninfo` to trigger a new WAL switch on the primary, allowing the concurrent backup from the standby to complete without waiting for a natural WAL switch.

Note

It's crucial to configure `primary_conninfo` if backing up a standby during periods of minimal or no write activity on the primary.

In Barman 3.8.0 and later, if `primary_conninfo` is configured, you can also set the `primary_checkpoint_timeout` option. This specifies the maximum wait time (in seconds) for a new WAL file before Barman forces a checkpoint on the primary. This timeout should exceed the `archive_timeout` value set on the primary.

If `primary_conninfo` is not set, the backup will still proceed but will pause at the stop backup stage until the last archived WAL segment is newer than the latest WAL required by the backup.

Barman requires that WAL files and backup data originate from the same Postgres cluster. If the standby is promoted to primary, the existing backups and WALs remain valid. However, you should update the Barman configuration to use the new standby for future backups and WAL retrieval.

Note

In case of a failover on the Postgres cluster you can update the Barman configuration with *Configuration Models*.

WALs can be retrieved from the standby via WAL streaming or WAL archiving. Refer to the *concepts* section for more details. If you want to start working with WAL streaming or WAL archiving, refer to the quickstart section on *streaming*

backups with wal streaming or rsync backups with wal archiving.

Note

For Postgres 10 and earlier, Barman cannot handle simultaneous WAL streaming and archiving on a standby. You must disable one if the other is in use, as WALs from Postgres 10 and earlier may differ at the binary level, leading to false-positive detection issues in Barman.

7.11 Managing external configuration files

Barman handles *external configuration files* differently depending on the backup method used. With the `rsync` method, external files are copied into the PGDATA directory. However, with the `postgres` method, external files are not copied, and a warning is issued to notify the user about those files.

Refer to the *Managing external configuration files* section in the recovery chapter to understand how external files are handled when restoring a backup.

Hint

Since Barman does not establish SSH connections to the PostgreSQL host when `backup_method = postgres`, you may want to configure a post-backup hook and use the output of `barman show-server` command to back up the external configuration files on your own right after the backup is finished.

7.12 Using an immutable storage for backups

Barman can be configured to store backups on immutable storage to protect against malicious actors or accidental deletions. Such storage may also be referred to as *WORM* (Write Once, Read Many) storage.

The main use case for this type of storage is to protect the backups from ransomware attacks. By using immutable storage, the backups cannot be deleted or modified for a specific period of time.

In order for Barman to provide immutable backups, only the backups and WAL files should be located in the immutable storage, leaving non-restorable data in regular storage. This way Barman will be able to maintain transient information about metadata of backups and WAL files as that information needs regular updates.

Given the above, to configure Barman to store backups on an immutable storage, you need to follow these suggestions:

- Only the following two directories should be configured to be stored on the immutable storage path:
 - *basebackups_directory*: The directory where backups are stored.
 - *wal_directory*: The directory where WAL files are stored.
- All other directories should be stored on a regular storage path because they are used by Barman's internal process and don't hold data crucial for restoring the cluster. This can be accomplished by configuring the *barman_home* option to point to a regular storage in the global configuration, or the *backup_directory* option in the server section. This still requires that the options from the previous bullet points are set accordingly.
- The WAL file catalog should be stored on a regular storage path. This can be accomplished by configuring the *xlogdb_directory* option to point to a regular storage.
- Paths used for restoring incremental or compressed backups, like *recovery_staging_path* and *local_staging_path* (see *restore configuration* for details), should also live in regular storage.

- Retention policies should cover at least the full period in which the backed up files are immutable. This can be accomplished by setting the `retention_policy` option in the server section to a value that is greater than the immutable storage's period of immutability. This is to ensure that the backups are not deleted before the immutability period expires.

To configure immutability of backups there's a `worm_mode` option that needs to be enabled. This will let Barman skip processes which are problematic when backups and WAL files are stored in a *WORM* environment.

Note

The option for relocating the `xlogdb` file was included in Barman 3.12. Refer to its *configuration section* for more information.

7.12.1 Current limitations

The current implementation of immutable backup support in Barman has the following limitation:

- The WORM environment must have a grace period. A grace period provides a predefined window during which data can be modified or deleted before WORM restrictions take effect. This requirement exists because Barman makes use of renaming to safely copy WALs to external partitions, which would fail if the file has already entered a WORM state.

In general, a grace period of at least 15 minutes is recommended, as this provides enough time for Barman to complete any necessary operations.

Note

If backup encryption is also enabled, then the grace period must be long enough to cover the time required to perform the encryption (especially when the backup also includes tablespaces, which results in multiple tarballs).

This is because encryption only happens at the end of the backup process, i.e. after `pg_basebackup` is finished. As encryption can not be performed in-place, each tar file is encrypted individually, having its unencrypted version deleted once it is complete.

Given these constraints, users should evaluate whether the current implementation meets their requirements before enabling immutable backup support.

7.13 Cloud Snapshot Backups

Barman can perform backups of Postgres servers deployed in specific cloud environments by utilizing snapshots of storage volumes. In this setup, Postgres file backups are represented as volume snapshots stored in the cloud, while Barman functions as the storage server for Write-Ahead Logs (WALs) and the backup catalog. Despite the backup data being stored in the cloud, Barman manages these backups similarly to traditional ones created with `rsync` or `postgres` backup methods.

Note

Additionally, snapshot backups can be created without a Barman server by using the `barman-cloud-backup` command directly on the Postgres server. Refer to the *barman cloud client package* section for more information on how to properly work with this option.

Important

The following configuration options and equivalent command arguments (if applicable) are not available when using `backup_method=snapshot`:

- `backup_compression`
- `bandwidth_limit` (`--bwlimit`)
- `parallel_jobs` (`--jobs`)
- `network_compression`
- `reuse_backup` (`--reuse-backup`)

To configure a backup using snapshot, include the following parameters in the Barman server configuration file:

```
backup_method = snapshot
snapshot_provider = CLOUD_PROVIDER
snapshot_instance = INSTANCE_NAME
snapshot_disks = DISK_NAME1,DISK_NAME2
```

Important

Ensure `snapshot_disks` includes all disks that store Postgres data. Any data stored on a disk not listed will not be backed up and will be unavailable during recovery.

7.13.1 Requirements and Configuration

To use the snapshot backup method with Barman, your deployment must meet these requirements:

1. Postgres must be running on a compute instance provided by a supported cloud provider.
2. All critical data, including PGDATA and tablespace data, must be stored on storage volumes that support snapshots.
3. The `findmnt` command must be available on the Postgres host.

Important

Configuration files stored outside of PGDATA will not be included in the snapshots. You will need to manage these files separately, using a configuration management system or other mechanisms.

7.13.2 Google Cloud Platform

To use snapshot backups on *GCP* with Barman, please ensure the following:

1. Python Libraries

Install the `google-cloud-compute` and `grpcio` libraries for the Python distribution used by Barman. These libraries are optional and not included by default.

Install them using `pip`:

```
pip3 install grpcio google-cloud-compute
```

Note

The `google-cloud-compute` library requires Python 3.7 or newer. GCP snapshots are not compatible with earlier Python versions.

2. Disk Requirements

The disks used in the snapshot backup must be zonal persistent disks. Regional persistent disks are not supported at this time.

3. Access Control

Barman needs a service account with specific permissions. You can either attach this account to the compute instance running Barman (recommended) or use the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to specify a credentials file.

Important

Ensure the service account has the permissions listed below:

- `compute.disks.createSnapshot`
- `compute.disks.get`
- `compute.globalOperations.get`
- `compute.instances.get`
- `compute.snapshots.create`
- `compute.snapshots.delete`
- `compute.snapshots.list`

For provider specific credentials configurations, refer to the [Google authentication methods](#) and [service account impersonation](#).

4. Specific Configuration

The fields `gcp_project` and `gcp_zone` are configuration options specific to GCP.

```
gcp_project = GCP_PROJECT_ID
gcp_zone = ZONE
```

7.13.3 Microsoft Azure

To use snapshot backups on Azure with Barman, ensure the following:

1. Python Libraries

The `azure-mgmt-compute` and `azure-identity` libraries must be available for the Python distribution used by Barman. These libraries are optional and not included by default.

Install them using `pip`:

```
pip3 install azure-mgmt-compute azure-identity
```

Note

The `azure-mgmt-compute` library requires Python 3.7 or later. Azure snapshots are not compatible with earlier Python versions.

2. Disk Requirements

All disks involved in the snapshot backup must be managed disks attached to the VM instance as data disks.

3. Access Control

Barman needs to access Azure using credentials obtained via managed identity or CLI login.

The following environment variables are supported: `AZURE_STORAGE_CONNECTION_STRING`, `AZURE_STORAGE_KEY` and `AZURE_STORAGE_SAS_TOKEN`. You can also use the `--credential` option to specify either `default`, `azure-cli` or `managed-identity` credentials in order to authenticate via Azure Active Directory.

Important

Ensure the credential has the permissions listed below:

- `Microsoft.Compute/disks/read`
- `Microsoft.Compute/virtualMachines/read`
- `Microsoft.Compute/snapshots/read`
- `Microsoft.Compute/snapshots/write`
- `Microsoft.Compute/snapshots/delete`

For provider specific credential configurations, refer to the [Azure environment variables configurations](#), [Identity Package](#) and [DefaultAzureCredential documentation](#).

4. Specific Configuration

The fields `azure_subscription_id` and `azure_resource_group` are configuration options specific to Azure.

```
azure_subscription_id = AZURE_SUBSCRIPTION_ID
azure_resource_group = AZURE_RESOURCE_GROUP
```

7.13.4 Amazon Web Services

To use snapshot backups on [AWS](#) with Barman, please ensure the following:

1. Python Libraries

The `boto3` library must be available for the Python distribution used by Barman. This library is optional and not included by default.

Install it using `pip`:

```
pip3 install boto3
```

2. Disk Requirements

All disks involved in the snapshot backup must be non-root EBS volumes attached to the same VM instance.

3. Access Control

Barman needs to access AWS so you must configure the AWS credentials with the `awscli` tool as the `postgres` user, by entering the Access Key and Secret Key that must be previously created in the IAM section of the AWS console.

Important

Ensure you have the permissions listed below:

- `ec2:CreateSnapshot`
- `ec2:CreateTags`
- `ec2>DeleteSnapshot`
- `ec2:DescribeSnapshots`
- `ec2:DescribeInstances`
- `ec2:DescribeVolumes`

For provider specific credentials configurations, refer to the [AWS boto3 configurations](#).

4. Specific Configuration

The fields `aws_region`, `aws_profile` and `aws_await_snapshots_timeout` are configuration options specific to AWS.

`aws_profile` is the name of the AWS profile in the credentials file. If not used, the default profile will be applied. If no credentials file exists, credentials will come from the environment.

`aws_region` overrides any region defined in the AWS profile.

`aws_await_snapshots_timeout` is the timeout for waiting for snapshots to be created (default is 3600 seconds).

When specifying `snapshot_instance` or `snapshot_disks`, Barman accepts either the instance/volume ID or the name of the resource. If you use a name, Barman will query AWS for resources with a matching `Name` tag. If zero or multiple matches are found, Barman will return an error.

```
aws_region = AWS_REGION
aws_profile = AWS_PROFILE_NAME
aws_await_snapshots_timeout = TIMEOUT_IN_SECONDS
```

5. Ransomware Protection

Ransomware protection is essential to secure data and maintain operational stability. With Amazon EBS Snapshot Lock, snapshots are protected from deletion, providing an immutable backup that safeguards against ransomware attacks. By locking snapshots, unwanted deletions are prevented, ensuring reliable recovery options in case of compromise. Barman can prevent unwanted deletion of backups by locking the snapshots when creating the backup.

Note

To delete a locked backup, you must first manually remove the lock in the AWS console.

To lock a snapshot during backup creation, you need to configure the following options:

1. Choose the snapshot lock mode: either `compliance` or `governance`.
2. Set either the lock duration or the expiration date (not both). Lock duration is specified in days, ranging from 1 to 36,500. If you choose an expiration date, it must be at least 1 day after the snapshot creation date and time, using the format `YYYY-MM-DDTHH:MM:SS.sssZ`.

- Optionally, set a cool-off period (in hours), from 1 to 72. This option only applies when the lock mode is set to compliance.

```
aws_snapshot_lock_mode = compliance | governance
aws_snapshot_lock_duration = 1
aws_snapshot_lock_cool_off_period = 1
aws_snapshot_lock_expiration_date = "2024-10-07T21:53:00.606Z"
```

Important

Ensure you have the permission listed below:

- `ec2:LockSnapshot`

For the concepts behind AWS Snapshot Lock, refer to the [Amazon EBS snapshot lock concepts](#).

7.13.5 Backup Process

Here is an overview of the snapshot backup process:

- Barman performs checks to validate the snapshot options, instance, and disks.**

Before each backup and during the `barman check` command, the following checks are performed:

- The compute instance specified by `snapshot_instance` and any provider-specific arguments exists.
- The disks listed in `snapshot_disks` are present.
- The disks listed in `snapshot_disks` are attached to the `snapshot_instance`.
- The disks listed in `snapshot_disks` are mounted on the `snapshot_instance`.

- Barman initiates the backup using the Postgres backup API.
- The cloud provider API is used to create a snapshot for each specified disk. Barman waits until each snapshot reaches a state that guarantees application consistency before proceeding to the next disk.
- Additional provider-specific details, such as the device name for each disk, and the mount point and options for each disk are recorded in the backup metadata.

7.13.6 Metadata

Regardless of whether you provision recovery disks and instances using infrastructure-as-code, ad-hoc automation, or manually, you will need to use Barman to identify the necessary snapshots for a specific backup. You can do this with the `barman show-backup` command, which provides details for each snapshot included in the backup.

For example:

```
Backup 20240813T200506:
  Server Name      : snapshot
  System Id       : 7402620047885836080
  Status          : DONE
  PostgreSQL Version : 160004
  PGDATA directory : /opt/postgres/data
  Estimated Cluster Size : 22.7 MiB

  Server information:
    Checksums       : on
```

(continues on next page)

(continued from previous page)

```

Snapshot information:
  provider           : aws
  account_id        : 714574844897
  region            : sa-east-1

  device_name       : /dev/sdf
  snapshot_id       : snap-0d2288b4f30e3f9e3
  snapshot_name     : Barman_AWS:1:/dev/sdf-20240813t200506
  Mount point       : /opt/postgres
  Mount options     : rw,noatime,seclabel

```

```

Base backup information:
  Backup Method      : snapshot-concurrent
  Backup Size        : 1.0 KiB (16.0 MiB with WALs)
  WAL Size           : 16.0 MiB
  Timeline           : 1
  Begin WAL          : 00000000100000000000000001A
  End WAL            : 00000000100000000000000001A
  WAL number         : 1
  Begin time         : 2024-08-14 16:21:50.820618+00:00
  End time           : 2024-08-14 16:22:38.264726+00:00
  Copy time          : 47 seconds
  Estimated throughput : 22 B/s
  Begin Offset       : 40
  End Offset         : 312
  Begin LSN          : 0/1A000028
  End LSN            : 0/1A000138

```

```

WAL information:
  No of files        : 1
  Disk usage         : 16.0 MiB
  WAL rate           : 5048.32/hour
  Last available     : 00000000100000000000000001B

```

```

Catalog information:
  Retention Policy   : not enforced
  Previous Backup    : - (this is the oldest base backup)
  Next Backup        : - (this is the latest base backup)

```

The `--format=json` option can be used when integrating with external tooling.

```

{
  "snapshots_info": {
    "provider": "gcp",
    "provider_info": {
      "project": "project_id"
    },
  },
  "snapshots": [
    {
      "mount": {
        "mount_options": "rw,noatime",

```

(continues on next page)

(continued from previous page)

```

        "mount_point": "/opt/postgres"
    },
    "provider": {
        "device_name": "pgdata",
        "snapshot_name": "barman-av-ubuntu20-primary-pgdata-20230123t131430",
        "snapshot_project": "project_id"
    }
},
{
    "mount": {
        "mount_options": "rw,noatime",
        "mount_point": "/opt/postgres/tablespaces/tbs1"
    },
    "provider": {
        "device_name": "tbs1",
        "snapshot_name": "barman-av-ubuntu20-primary-tbs1-20230123t131430",
        "snapshot_project": "project_id",
    }
}
]
}
}

```

The metadata found in `snapshots_info/provider_info` and `snapshots_info/snapshots/*/provider` varies depending on the cloud provider, as detailed in the following sections.

GCP

`snapshots_info/provider_info`

- `project`: The GCP project ID of the project which owns the resources involved in backup and recovery.

`snapshots_info/snapshots/*/provider`

- `device_name`: The short device name with which the source disk for the snapshot was attached to the backup VM at the time of the backup.
- `snapshot_name`: The name of the snapshot.
- `snapshot_project`: The GCP project ID which owns the snapshot.

Azure

`snapshots_info/provider_info`

- `subscription_id`: The Azure subscription ID which owns the resources involved in backup and recovery.
- `resource_group`: The Azure resource group to which the resources involved in the backup belong.

`snapshots_info/snapshots/*/provider`

- `location`: The Azure location of the disk from which the snapshot was taken.
- `lun`: The LUN identifying the disk from which the snapshot was taken at the time of the backup.
- `snapshot_name`: The name of the snapshot.

AWS

`snapshots_info/provider_info`

- `account_id`: The ID of the AWS account which owns the resources used to make the backup.
- `region`: The AWS region in which the resources involved in backup are located.

`snapshots_info/snapshots/*/provider`

- `device_name`: The device to which the source disk was mapped on the backup VM at the time of the backup.
- `snapshot_id`: The ID of the snapshot as assigned by AWS.
- `snapshot_name`: The name of the snapshot.

Barman also offers additional features regarding WAL archiving.

8.1 WAL compression

Barman can compress WAL files as they enter the Barman's WAL archive. This process is handled automatically by `barman cron` or when the `barman archive-wal` command is executed manually.

Compression is enabled via the `compression` option in the configuration file. The option can use one of the following values:

- `lz4`: for LZ4 compression (requires the `lz4` library to be installed);
- `xz`: for XZ compression (uses Python's internal compression library);
- `zstd`: for Zstandard compression (requires the `zstandard` library to be installed);
- `gzip`: for Gzip compression (requires the `gzip` utility);
- `pygzip`: for Gzip compression (uses Python's internal compression library);
- `pigz`: for Pigz compression (requires the `pigz` utility);
- `bzip2`: for Bzip2 compression (requires the `bzip2` utility);
- `pybzip2`: for Bzip2 compression (uses Python's internal compression library);
- `custom`: for custom compression, which requires you to set the following options as well: `custom_compression_filter`, `custom_decompression_filter`, `custom_compression_magic`. Check *Write-Ahead Logs (WAL)* for details.

Note

When using options such as `bzip2`, `gzip` and `pigz` Barman will fork a new process for compression.

Important

Barman does not store metadata on how each WAL file was compressed, nor on how to decompress them. It relies mainly on the server's current configuration. Therefore, when using `compression = custom`, you must ensure that `custom_decompression_filter` remains compatible with the WALs currently available in archive. If, for instance, `custom_decompression_filter` is changed to a different compression algorithm midway through, Barman will be unable to decompress old WALs still archived with the previous compression algorithm when restoring.

A compression level can also be specified by using the `compression_level` configuration. Valid values are integers within the supported range of the chosen algorithm or one of the predefined labels: `low`, `medium`, and `high`. The range of each algorithm as well as what level each predefined label maps to can be found in [compression_level](#).

If archiving via `archive_command` with `barman-wal-archive`, compression can also be configured using the `--compression` and `--compression-level` options with an algorithm and level of your choice. In such cases, compression is done on the client side, before the WAL is sent to the Barman server. See `commands-barman-cli-barman-wal-archive`.

Note

When compression is enabled in `barman-wal-archive`, it takes precedence over the compression settings configured on the Barman server, if they differ.

Important

When compression is enabled in `barman-wal-archive`, it is performed on the client side, before the file is sent to Barman. Be mindful of the database server's load and the chosen compression algorithm and level, as higher compression can delay WAL shipping, causing WAL files to accumulate.

8.2 WAL encryption

Barman can encrypt WAL files as they enter the Barman's WAL archive. This process is handled automatically by `barman cron` or when the `barman archive-wal` command is executed manually.

Both WAL and backup encryption are enabled via the `encryption` option in the configuration file. Currently, only `pgp` and `none` (no encryption) are accepted as values. A detailed guide on how to configure encryption is available in [Backup Encryption](#).

8.3 Synchronous WAL streaming

Barman can also reduce the *RPO* to zero, by collecting the transaction WAL files like a synchronous standby server would.

To configure a scenario with *RPO* zero, the Barman server must be configured to archive WALs via a streaming connection and the `receive-wal` process has to be configured as a synchronous connection to the Postgres server.

First, you need to retrieve the application name of the Barman `receive-wal` process with the `show-servers` command:

```
barman show-servers pg | grep streaming_archiver_name
```

Output:

```
streaming_archiver_name: barman_receive_wal
```

Note

The application name Barman uses when starting the receive-wal process is configured with the `streaming_archiver_name` configuration option. The default value for this option is `barman_receive_wal`.

Then the application name should be added to the `synchronous_standby_names` parameter in the `postgresql.conf` file:

```
synchronous_standby_names = 'barman_receive_wal'
```

Important

Barman with *RPO* zero adds more security to your backups and gives you more recovery options. However, it should not be considered as a substitution of a real Postgres replica. Please read the [official Postgres documentation about "Synchronous Replication"](#) for more information on this topic.

The Postgres server configuration needs to be reloaded for the changes to take effect.

If the server has been configured correctly, the `barman replication-status` command should show the receive-wal process as a synchronous streaming client:

```
barman replication-status pg
```

Output:

```
Status of streaming clients for server 'pg':
Current xlog location on master: 0/9000098
Number of streaming clients: 1

1. #1 Sync WAL streamer
  Application name: barman_receive_wal
  Sync stage      : 3/3 Remote write
  Communication   : TCP/IP
  IP Address      : 139.59.135.32 / Port: 58262 / Host: -
  User name       : streaming_barman
  Current state   : streaming (sync)
  Replication slot: barman
  WAL sender PID  : 2501
  Started at      : 2016-09-16 10:33:01.725883+00:00
  Sent location   : 0/9000098 (diff: 0 B)
  Write location  : 0/9000098 (diff: 0 B)
  Flush location  : 0/9000098 (diff: 0 B)
```

Catalog information

The backup catalog is a comprehensive record that keeps track of all servers and backups on the Barman node. Note that servers are the Postgres database systems that are being tracked and backed up by Barman. Each server is configured within the Barman node to ensure that its data is regularly backed up and can be restored if needed. The Barman node is another server that hosts the Barman software, which manages the backup and recovery processes for these Postgres servers. It communicates with the configured servers to perform backups, store them securely, and maintain a detailed catalog of all backup activities.

The backup catalog is essential for effective backup management and recovery operations, offering a unified interface that makes it a key component of the Barman tool. It plays a central role in organizing, monitoring, and executing backup and recovery tasks by providing a comprehensive view and precise control over all backup activities. This streamlined approach enhances efficiency, simplifies management, and ensures seamless recovery processes.

The global configuration option `barman_home` specifies the directory where Barman stores and manages backups for multiple servers. By default, this directory is set to `/var/lib/barman`, but it can be customized by modifying the global configuration file found at `/etc/barman.conf`. Within this directory, backups from each server are organized into separate subdirectories, with each server's backups located under `<barman_home>/<SERVER_NAME>/base`. This structure helps keep backups organized and easily accessible.

9.1 Purpose

Serve as a centralized repository that keeps track of all Postgres server and backup-related information.

Here are some key roles it plays:

- **Backup Metadata Storage:** It stores metadata about each backup, such as the backup's start and end times, the status, the specific Postgres instance it was taken from and many other metrics. This metadata helps in tracking and managing the backup lifecycle.
- **Backup Management:** The catalog provides a way to organize and manage backups efficiently. By keeping detailed records, it simplifies operations such as listing available backups, checking the status of backups, identifying the latest or specific backups, and applying retention policies.
- **Restore Operations:** During a restore operation, the catalog helps in quickly identifying which backups are available and their details. This facilitates efficient restoration by providing necessary information about the

backups and their associated archive logs.

- **Ease of Use:** It simplifies the process of managing multiple backups and their associated metadata, making it easier for administrators to handle large numbers of backups and complex backup strategies.

9.2 Usage

Barman offers a straightforward terminal interface for managing Postgres backups and interacting with the backup catalog. This interface provides a range of sub-commands for both server management and backup operations. All Barman sub-commands can be found in the [sub commands](#) section, including two important ones which are `list-backups` and `show-backup`. These commands can be found below with an example.

9.2.1 list-backups

Show available backups for a server. This command is useful to retrieve a list of backups with minimal yet important information, such as backup ID and the backup type.

For example:

```
svr_pg17 20240901T103000 - F - Mon Nov  2 15:12:03 2024 - Size: 820.0 MiB - WAL Size: 22.0 MiB
```

- `svr_pg17` is the server name.
- `20240901T103000` is the backup ID.
- `F` is the backup type label.
- `Mon 2 15:12:03 2024` is the date and time the backup operation ended.
- `Size: 820.0 MiB` is the size of the backup.
- `WAL Size: 22 MiB` is the size of all WAL files related to this backup.

Note

The backup type label can be `F` for full backups and `I` for block-level incremental backups. `R` for rsync backups and `S` for cloud snapshot backups.

9.2.2 show-backup

Show detailed information about a specific backup. For example, a block-level incremental backup:

```
Backup 20240902T130000:
  Server Name       : prod_pg17
  Status            : DONE
  PostgreSQL Version : 170000
  PGDATA directory  : /var/lib/pgsql/17/data
  Estimated Cluster Size : 244.7 GiB

Server information:
  Checksums         : on
  WAL summarizer    : on

Base backup information:
```

(continues on next page)

(continued from previous page)

```

Backup Method      : postgres
Backup Type       : incremental
Backup Size       : 3.4 GiB (36.7 GiB with WALs)
WAL Size          : 32.3 GiB
Resources saved   : 241.3 GiB (98.61%)
Timeline          : 1
Begin WAL         : 0000000100000CFD000000AD
End WAL           : 0000000100000D0D00000008
WAL number        : 3932
WAL compression ratio: 79.51%
Begin time        : 2024-09-02 13:00:01.633925+00:00
End time          : 2024-09-03 10:27:06.522846+00:00
Copy time         : 1 second
Estimated throughput : 2.0 MiB/s
Begin Offset      : 1575048
End Offset        : 13853016
Begin XLOG        : CFD/AD180888
End XLOG          : D0D/8D36158

```

WAL information:

```

No of files       : 35039
Disk usage        : 121.5 GiB
WAL rate          : 275.50/hour
Compression ratio : 77.81%
Last available    : 0000000100000D95000000E7

```

Catalog information:

```

Retention Policy  : not enforced
Previous Backup   : 20240902T120001
Next Backup       : - (this is the latest base backup)
Root Backup       : 20240801T015504
Parent Backup     : 20240831T016504
Backup chain size : 3
Children Backup(s) : 20240903T018515,20240903T019515

```

Note

The output of the `show-backup` command can vary depending on the version of your Postgres server and the type of backup.

- The fields `Root Backup`, `Parent Backup`, `Backup chain size` and `Children Backup(s)` are relevant only for block-level incremental backups taken with `backup_method=postgres` on Postgres 17 or newer. These fields will not be shown for other types of backups or older Postgres versions.
- The `show-backup` command relies on backup metadata. If a backup was created with Barman version 3.10 or earlier, it will not include fields introduced in version 3.11, such as those related to block-level incremental backups in Postgres 17.
- The field `Resource Saved` is available for `rsync` and incremental backups, and `Snapshot Information` is only available for snapshot backups.
- If the backup is encrypted, the `Encryption` field is shown in the output, containing the encryption method that was used.

- The possible values for the field Backup Type are:
 - `rsync`: for a backup taken with `rsync`.
 - `full`: for a full backup taken with `pg_basebackup`.
 - `incremental`: for an incremental backup taken with `pg_basebackup`.
 - `snapshot`: for a snapshot-based backup taken in the cloud.

The restore command is used to restore an entire Postgres server from a backup created with the backup command. When creating a backup, a `backup_id` is assigned, which uniquely identifies the backup. To use the restore command, run:

```
barman restore [OPTIONS] SERVER_NAME BACKUP_ID DESTINATION_PATH
```

It is possible to restore a backup by specifying the `backup_id` as `auto`. In this case, Barman will automatically choose the most suitable backup from the catalog. If no recovery target is provided, it will select the most recent available backup by default. If a recovery target is specified, Barman will use it to identify the appropriate backup based on the recovery criteria:

- `target_time`: Barman will retrieve the most recent backup available up to `target_time`, e.g., `2025-01-21 10:00:00`.
- `target_lsn`: Barman will retrieve the most recent backup available up to `target_lsn`, e.g., `3/64000000`.
- `target_tli`: Barman will retrieve the most recent backup available from timeline `target_tli`, e.g., `2`.
- `target_tli` combined with one of the other two targets: Barman will retrieve the most recent backup available up to the `target_lsn` or `target_time` which belongs to the `target_tli`, e.g., `2025-01-21 10:00:00` from timeline `2`.

Note

- Refer to *Restore and recover* for a clearer understanding of the recovery concept in Barman.
- The backup is restored to the specified directory, which will be ready to start a Postgres instance.
- Use the `list-backups` command to find the specific backup ID you need.
- Barman does not track symbolic links inside PGDATA (except for tablespaces). Ensure you manage symbolic links and include them in your disaster recovery plans.
- When specifying the `backup_id` as `auto`, Barman does not consider the *backup_method*. If the selected backup requires specific arguments or doesn't support certain options, the restoration may fail. In such

cases, you will receive an error message that can help adjust the restore command accordingly. For example, `--snapshot-recovery-instance` is required when restoring a snapshot backup.

Danger

Do not run the restore command with the destination directory set to a location where a Postgres instance is actively running. If you intend to reuse that directory, ensure that Postgres is fully stopped before initiating recovery, including recovery of tablespace directories.

10.1 Local recovery

In the absence of `--remote-ssh-command`, Barman restores the backup locally to the path specified in `DESTINATION_PATH`.

All files and directories will be owned by the `barman` user. Therefore, there are two options when starting Postgres to perform the recovery:

1. Start Postgres as the `barman` user. In this case, if using `--get-wal`, the `restore_command` can invoke `get-wal` without requiring `sudo`.
2. Start Postgres as the `postgres` user. In this case, you must change the ownership of all files and directories restored (this includes PGDATA and any tablespaces) to the `postgres` user before starting the server. If using `--get-wal`, the `restore_command` must use `sudo` to run `get-wal` as the `barman` user.

Check the *Fetching WALs* for more information on settings for WAL restore.

10.2 Remote Recovery

Use `--remote-ssh-command COMMAND` to perform recovery on a remote server via SSH. It's recommended to use the `postgres` user on the target node for remote recovery.

Known Limitations

- Requires at least 4GB of free space in the system's temporary directory unless `get-wal` is specified.
- SSH must use public key authentication.
- The remote user must be able to create the necessary directory structure.
- Ensure there is enough free space on the remote server for the base backup and WAL files.

10.3 Tablespace Remapping

Use `--tablespace NAME:DIRECTORY` to remap tablespaces to a new location. Barman will attempt to create the destination directory if it doesn't exist.

Danger

By default, tablespaces are restored to the same path they had on the source server. Be cautious when restoring a backup without any remapping to a destination where a Postgres instance already exists, as it can end up overriding existing tablespace directories.

10.4 Point-in-Time Recovery

Specify a recovery target with one of the options:

- `--target-time`: Recover to a specific timestamp.
- `--target-xid`: Recover to a specific transaction ID.
- `--target-lsn`: Recover to a Log Sequence Number.
- `--target-name`: Recover to a named restore point.
- `--target-tli`: Recover to a specific timeline.
- `--target-immediate`: End recovery when a consistent state is reached.

Note

- Recovery targets must be a value after the end of the backup. To recover to a point in time within a backup, use the previous backup.
- Timezone defaults to the Barman host if not specified in `--target-time`'s timestamp.
- Use `--exclusive` to control whether to stop right before the target or including the target.
- `--target-tli` sets the target timeline. Use numeric IDs or shortcut values (`latest` or `current`).
- When at least one `--target-*` option is specified, a `recovery.signal` file is created by Barman when restoring the backup, which signals the server to start a targeted recovery.
- When specifying the `backup_id` as `auto`, the only recovery targets allowed are: `--target-time`, `--target-lsn` and `--target-tli`. Note that `--target-time` and `--target-lsn` are mutually exclusive, while `--target-tli` can be used independently or together with `--target-time` or `--target-lsn`.

The previous targets can be used with a `--target-action` which can take these values:

- `shutdown`: Shut down Postgres when the target is reached.
- `pause`: Pause Postgres for inspection when the target is reached.
- `promote`: Promote Postgres to primary when the target is reached.

You can also configure the instance as a standby by calling `--standby-mode`. After the backup is restored, ensure you modify the configuration to connect to the intended upstream node before starting the restored node in recovery mode.

Note

- When `--standby-mode` is specified, a `standby.signal` file is created instead of a `recovery.signal` file.
- When using `--standby-mode`, although possible, you are not expected to set any of the `--target-*` options.

See also

For more information regarding Postgres recovery behavior, refer to [Archive Recovery](#) and [Recovery Target](#)

10.5 Fetching WALs from Barman

Use `--get-wal` to configure Postgres to fetch WALs from Barman during recovery. If not set, Barman will copy all the WALs required for Postgres recovery as part of the restore command.

Note

When using `--no-get-wal` with targets like `--target-xid`, `--target-name`, or `--target-time`, Barman will copy the entire WAL archive to ensure availability.

Another option is to include `get-wal` inside the `recovery_options` configuration at the global/server level prior to a recovery operation to retrieve WAL files during the recovery process without the need to specifying the `--get-wal`, effectively turning the Barman server into a WAL hub for your servers.

```
recovery_options = 'get-wal'
```

If `get-wal` is included during restore, Barman will set up the `restore_command` to use either `barman get-wal` or `barman-wal-restore` to retrieve the required WAL files, depending on whether the recovery is local or remote.

If `get-wal` is specified in `recovery_options` but not needed during a specific recovery, you can disable it using the `--no-get-wal` option with the `barman restore` command.

10.5.1 Using get-wal for local recovery

Here's an example of a `restore_command` for local recovery:

```
restore_command = 'sudo -u barman barman get-wal SERVER %f > %p'
```

Remember that the `barman get-wal` command should always be executed as the `barman` user, with the necessary permissions to access WAL files from the catalog, which is why `sudo -u barman` is used in this example.

To allow the `postgres` user to run the `get-wal` command as the `barman` user, you can add the following line to the `/etc/sudoers` file (replace `SERVER` with the actual server name):

```
postgres ALL=(barman) NOPASSWD: /usr/bin/barman get-wal SERVER *
```

10.5.2 Using get-wal for remote recovery

For remote recovery, setting `recovery_options` to `get-wal` will create a `restore_command` using the `commands-barman-cli-barman-wal-restore` script, which is designed to handle SSH connection errors more robustly.

This script offers useful features like automatic compression and decompression of WAL files and the `peek` feature, allowing you to retrieve upcoming WAL files while Postgres is processing earlier ones, optimizing bandwidth between Postgres and Barman.

`barman-wal-restore` is included in the `barman-cli` package. Here's an example of a `restore_command` for **remote recovery**:

```
restore_command = 'barman-wal-restore -U barman backup SERVER_NAME %f %p'
```

Here, `backup` refers to the host where Barman is installed. Since it communicates via SSH, SSH key authentication is required for the `postgres` user to log in as `barman` on the backup server. If you need to use a non-default SSH port, you can specify it with the `--port` option.

To verify that `barman-wal-restore` can connect to the Barman server and that the required Postgres server is set up to send WAL files, use the following command:

```
barman-wal-restore --test backup pg DUMMY DUMMY
```

Here, `backup` refers to the host where Barman is installed, `pg` is the name of the Postgres server configured in Barman, and `DUMMY` acts as a placeholder (the script needs two arguments for the WAL file name and destination directory, which will be ignored).

If everything is set up correctly, you should see:

```
Ready to retrieve WAL files from the server pg
```

For further details on the `barman-wal-restore` command, type `man barman-wal-restore` on the host where `barman-cli` was installed or refer to the `commands-barman-cli-barman-wal-restore` command reference.

Tip

When both the `pg_wal` directory and the `spool` directory are located on the same filesystem, serving WAL files will be faster because the files are renamed rather than copied. However, if these directories are on different filesystems, there will be no performance improvement, as the operation will involve both copying the file and then removing the original. Be mindful of the filesystem locations to optimize WAL file management efficiency.

10.6 Recovering Encrypted Backups

Encrypted backups and WALs are decrypted during the restore phase, before they are copied to the final destination. During the restore, a command to fetch the private key's passphrase must be present in `encryption_passphrase_command`. This command must output the passphrase to standard output and can be used to retrieve it from a secure location such as a password vault, an external key management service, or a file.

These are some examples of how to set the passphrase command:

- Example reading from an environment variable:

```
encryption_passphrase_command="echo $BARMAN_PASSPHRASE"
```

- Example reading from a file:

```
encryption_passphrase_command="cat /path/to/barman_passphrase"
```

- Example reading from HashiCorp Vault:

```
encryption_passphrase_command="vault kv get -field=<FIELD> <KEY>"
```

- Example reading from AWS Secret Manager:

```
encryption_passphrase_command="aws secretsmanager get-secret-value --secret-id  
<SECRET_NAME> --profile <AWS_PROFILE> --output text --query SecretString | jq -r  
'.<SECRET_KEY>'"
```

The decryption of backups happens as follows:

1. The backup is decrypted into a staging directory on the Barman server. This location is defined by the `local_staging_path` option.
2. If any additional operations are required — such as decompression or combination (in the case of incremental backups) — they are performed using the staging directory's content as source. Otherwise, the decrypted files are copied directly to the final destination.

3. The staging directory is removed after the restore is complete.

Decryption of WAL files depends on how they are retrieved during recovery:

1. If using the `--no-get-wal` option (default), all required WAL files are decrypted into the staging directory and then copied to the final destination. That means the `encryption_passphrase_command` is invoked once and the output is reused for all WAL files. Also, the command is only required during the execution of the `barman restore` command, and not during the Postgres recovery process.
2. Using the `--get-wal` option, WAL files are served to the Postgres server when needed during recovery process. In this scenario, Barman decrypts each WAL file locally before sending it to the Postgres server. This also means that `encryption_passphrase_command` is invoked once for each WAL file being fetched through the `restore_command`.

10.7 Recovering Compressed Backups

If a backup is compressed using the `backup_compression` option, Barman can decompress it during restore.

The process involves a few steps:

1. The compressed backup files are copied to a staging directory on either the local or remote server using Rsync.
2. These files are then decompressed to the restore destination directory.
3. For remote recovery, configuration files requiring special handling are copied from the restore destination directory to a local temporary directory in the barman node, edited and mangled as needed, and then returned to the restore directory using Rsync. For local recovery, the local temporary directory is the restore destination itself, so editing and mangling operations are done in place. This intermediate step is necessary because Barman can only access individual files in the restore directory, as the backup directory contains only a compressed tarball file.
4. The staging directory is removed after the restore is complete.

Since Barman does not have knowledge of the deployment environment, it depends on the `recovery_staging_path` option to determine an appropriate location for the staging directory. Set the option in the global/server configuration or use the `--recovery-staging-path` option with the `barman restore` command. Failing to do so will result in an error, as Barman cannot guess a suitable location on its own.

10.8 Recovering block-level incremental Backups

If you are recovering from a block-level incremental backup, Barman combines the backup chain using `pg_combinebackup`. This chain consists of the root backup and all subsequent incremental backups up to the one being recovered.

To successfully recover from a block-level incremental backup, you must specify the `local_staging_path` in the global/server configuration or use the `--local-staging-path` option with the `barman restore` command. Failing to do so will result in an error, as Barman cannot automatically determine a suitable staging location.

The process involves the following steps:

1. Barman creates a synthetic backup by combining the chain of backups. This is done in a staging directory on the Barman server using `pg_combinebackup`. Barman will create a subfolder inside the staging directory with the ID of the backup.
2. If the recovery is local, the synthetic backup is moved directly to the target location. If it is a remote recovery, the synthetic backup is transferred to the target location using Rsync.
3. After the restore is complete, the temporary subfolder in the local staging directory used for combining backups is removed. The local staging directory itself is kept.

Important

If any backups in the chain were taken with checksums disabled, but the final backup has checksums enabled, the resulting synthetic backup may contain pages with invalid checksums. Please refer to the limitations in the [pg_combinebackup documentation](#) for more details.

10.9 Limitations of .partial WAL files

When using `streaming_archiver`, Barman relies on `pg_receivewal` to continuously receive transaction logs from a Postgres server (either master or standby) through the native streaming replication protocol. By default, `pg_receivewal` writes these logs to files with a `.partial` suffix, indicating they are not yet complete. Barman looks for these `.partial` files in the `streaming_wals_directory`. Once `pg_receivewal` completes the file, it removes the `.partial` suffix and hands it over to Barman's `archive-wal` command for permanent storage and compression.

If the master Postgres server suddenly fails and cannot be recovered, the `.partial` file that was streamed to Barman may contain crucial data that might not have been delivered to the archiving process.

Starting with Barman version 2.10, the `get-wal` command can retrieve the content of the current `.partial` WAL file using the `--partial` or `-P` option. This is useful for recovery, whether performing a full restore or a point-in-time recovery. When you initiate a restore command with `get-wal` and without `--standby-mode`, Barman will automatically include the `-P` option in the `barman-wal-restore` command to handle the `.partial` file.

Moreover, `get-wal` will check the incoming directory for any WAL files that have been sent to Barman but not yet archived.

10.10 Managing external configuration files

Barman restores external configuration files differently depending on how the backup was originally taken. When restoring a `rsync` backup, external files are restored into the `PGDATA` directory via `rsync`, and not in the original location. A warning is issued regarding potentially risky settings, including the ones related to configuration files. In contrast, when restoring a `postgres` backup, external files are not restored as they were not backed up. A warning is provided to inform the user about the files that were not restored.

Refer to the [Managing external configuration files](#) section in the backup chapter to understand how external files are handled when creating a backup.

10.11 Recovering from Snapshot Backups

Barman currently does not support fully automated recovery from snapshot backups. This limitation arises because snapshot recovery requires provisioning and managing new infrastructure, a task best handled by dedicated [IAC](#) solutions like Terraform or OpenTofu.

However, you can still use the `barman restore` command to validate the snapshot recovery instance and perform post-recovery tasks, such as checking the Postgres configuration for unsafe settings and configuring any necessary PITR options. The command will also copy the `backup_label` file into place, as this file is not included in the volume snapshots, and will transfer any required WAL files--unless the `--get-wal` recovery option is specified, in which case it configures the Postgres `restore_command` to fetch the WALs.

If restoring from a backup created with `barman-cloud-backup`, you should use the `barman-cloud-restore` command instead of `barman restore`.

Note

The same requirements and configurations apply for restore when working with a cloud provider. See the [Requirements and Configuration](#) section and the specific cloud provider you are working with in the [Cloud Snapshot Backups](#) section.

10.11.1 Recovery Steps

1. Provision a new disk for each snapshot taken during the backup.
2. Provision a compute instance to which each disk from step 1 is attached and mounted according to the backup metadata.
3. Use the `barman restore` or `barman-cloud-restore` command to validate and finalize the recovery.

Steps 1 and 2 are ideally managed by an existing IAC system, but they can also be performed manually or via a custom script.

10.11.2 Helpful Resources

[Example recovery script for GCP.](#)

[Example runbook for Azure.](#)

These resources make assumptions about your backup and recovery environment and should be customized before use in production.

10.11.3 Running the restore command

Once the recovery instance is provisioned and the disks cloned from the backup snapshots are attached and mounted, execute the `barman restore` command with the following additional arguments:

- `--remote-ssh-command`: The SSH command required to log into the recovery instance.
- `--snapshot-recovery-instance`: The name of the recovery instance as specified by your cloud provider.
- Any additional arguments specific to the snapshot provider.

Example Command

```
barman restore SERVER_NAME BACKUP_ID REMOTE_RECOVERY_DIRECTORY \  
--remote-ssh-command 'ssh USER@HOST' \  
--snapshot-recovery-instance INSTANCE_NAME
```

Barman will automatically recognize the backup as a snapshot and verify that the attached disks were cloned from the corresponding snapshots. It will then prepare Postgres for recovery by copying the backup label and WALs into place and adjusting the Postgres configuration with the necessary recovery options.

Provider-Specific Arguments

For GCP:

- `--gcp-zone`: The availability zone where the recovery instance is located. If omitted, Barman will use the `gcp_zone` value set in the server config.

For Azure:

- `--azure-resource-group`: The resource group for the recovery instance. If not provided, Barman will refer to the `azure_resource_group` value in the server config.

For AWS:

- `--aws-region`: The AWS region of the recovery instance. If not specified, Barman will default to the `aws_region` value set in the server config.

11.1 Overview

A retention policy for backups is a set of strategic guidelines designed to manage how backup copies of your data are handled over time. This policy outlines the rules and guidelines for how long backups are kept, when they should be archived or deleted, and how they are organized. Implementing a well-defined retention policy is essential for ensuring data protection, optimizing storage use, and meeting compliance requirements.

11.2 Key Components of Retention Policies

Understanding the key components of these policies is crucial for designing a system that balances data protection, storage efficiency, and compliance.

11.2.1 Retention Duration

- **Time-Based Retention:** This component specifies how long backups are retained, such as keeping backups for a fixed period (e.g., 30 days, 1 year). Time-based retention is straightforward and ensures that backups older than a certain age are automatically deleted.
- **Quantity-Based Retention:** Alternatively, retention policies can be based on the number of backups retained (e.g., the last 10 backups). This method is useful for maintaining a specific number of recent backups, regardless of their age.

11.2.2 Backup Types

- **Full Backups:** These backups capture the entire dataset and are often retained longer due to their comprehensive nature. Different retention policies may apply to full backups compared to other types.
- **Incremental Backups:** Incremental backups capture changes since the last backup. Retention policies for these backups may differ, reflecting their role in the backup chain and their dependency on other backups.

11.2.3 Cleanup Rules

- **Automated Cleanup:** Effective retention policies include automated cleanup mechanisms that identify and remove outdated backups according to predefined rules. This reduces manual intervention and minimizes the risk of retaining unnecessary data.
- **Archiving and Deletion:** Cleanup rules can specify whether old backups are archived before deletion or if they are removed directly. Archiving can be useful for maintaining historical data for compliance or other purposes.

11.3 Key Objectives of Retention Policies

Implementing a robust retention policy is fundamental to effective backup management, encompassing some key objectives:

11.3.1 Ensuring Sufficient Data Protection

- **Historical Recovery:** Retention policies define the duration for which backups are kept to facilitate recovery from various points in time. This is crucial for recovering data not only from recent backups but also from older ones in case of data loss, corruption, or inadvertent changes.
- **Recovery Flexibility:** By retaining backups over different periods, you can respond to different types of data recovery scenarios, whether it's restoring the latest data, addressing corruption issues, or undoing erroneous operations.

11.3.2 Optimizing Storage Usage

- **Efficient Storage Management:** Retention policies help prevent the accumulation of obsolete backups that consume valuable storage space. This is achieved by setting limits on the number of backups or the duration for which they are kept, thereby optimizing storage utilization and managing costs effectively.
- **Cost Control:** By automating the cleanup of outdated backups, organizations can avoid unnecessary expenses related to storage infrastructure and associated maintenance.

11.3.3 Compliance and Regulation

- **Meeting Legal Requirements:** Many industries have specific regulations governing data retention, which may dictate minimum retention periods for backups. A well-defined retention policy ensures that these regulatory requirements are met, helping organizations stay compliant with legal and industry standards.
- **Audit Readiness:** Proper retention policies facilitate easier audits by maintaining a clear and organized backup history that demonstrates compliance with retention regulations.

11.4 Minimum redundancy safety

You can set a minimum number of backups for your Postgres server using the `minimum_redundancy` option in the global or per-server configuration. By default, this option is set to 0.

If you set `minimum_redundancy` to a number greater than 0, Barman will ensure that you always have at least that many backups available on the server.

This setting helps protect against accidental deletion of backups.

Note

Make sure your retention policy does not conflict with the minimum redundancy setting. Check Barman's logs regularly for any related messages.

11.5 Scope of retention policies

Barman allows you to define retention policies by two methods.

11.5.1 Backup Redundancy

Specifies the number of backups to retain. Barman keeps the most recent backups up to the specified number. This type of policy does not consider the time period for retention but focuses on the number of backups.

For example, if you set a redundancy of 3, Barman will retain the three most recent backups and discard older ones.

11.5.2 Recovery Window

Specifies the duration for which backups must be retained to allow recovery to any point within that window. The interval window always ends at the current time and spans backward for the specified period. Barman retains backups and archive logs necessary for point-in-time recovery to any moment within this window.

For example, if you set a 7-day recovery window, Barman will keep backups and WAL files to allow recovery to any point within the past 7 days. This means that the first backup that falls outside the window will still be retained with its corresponding WALs, but backups before this one and all the older WALs will be marked as obsolete and eventually be evicted.

11.5.3 Keep command

The `keep` command can be used to mark a specific backup so its kept indefinitely. This overrides the retention policy explained earlier for that backup. You can find more information on the `keep` command in the Barman `keep` command documentation.

11.6 Use cases

11.6.1 Point-In-Time Recovery

Base backups and archived WAL files have the same retention policy. This setup allows you to recover the data from your Postgres server to any point in time from the end time of the earliest available backup.

11.6.2 Operational Efficiency and Space Management

You may want to maintain a certain number of recent backups while periodically removing older ones to save on storage cost and manage storage space effectively, especially in environments with limited resources.

11.6.3 Long-Term Archival

For compliance or historical purposes, you may need to retain backups for extended periods beyond the usual operational requirements. This is often required in regulated industries where data must be kept for a certain period.

11.7 How retention policies are enforced

Retention policies in Barman are enforced automatically by Barman's maintenance tasks which are executed by `barman cron`.

11.8 Configurations and Syntax

Retention policies are configured globally or per server using the `retention_policy` option offering flexibility in a multi-server environment. By default, the value of the `retention_policy` option is not set, so no retention is enforced.

Retention policies have the following syntax:

```
retention_policy = {REDUNDANCY value | RECOVERY WINDOW OF value {DAYS | WEEKS | MONTHS}}
```

- value must be an integer greater than 0.
- For backup redundancy, value must meet or exceed the server's minimum redundancy level.
- For recovery window, value must be at least as high as the server's minimum redundancy level in reverse order.
- If value is not assigned, a warning is generated.

Important

Block-level incremental backups are not considered in retention policies, as they depend on their parent backups and the root backup. Only the root backup is used to determine retention.

11.9 Retention policy for block-level incremental backups

When retention policy is applied:

- Barman will focus on the root backup.
- If the root backup is marked as `KEEP:FULL`, all associated incremental backups are marked as `VALID`, regardless of whether the root backup is within the retention policy.
- If the root backup is marked as `KEEP:STANDALONE` and is still within the retention policy, all associated incremental backups are marked as `VALID`. However, if the root backup is outside the retention policy, all associated incremental backups are marked as `OBSOLETE`.
- If the root backup is not marked with a `KEEP` flag, all associated incremental backups inherit the same label. For instance, if the root backup is marked as `OBSOLETE`, all associated incremental backups are also marked as `OBSOLETE`.

11.10 Retention policy for Cloud Backups

We can have two scenarios for Cloud Backups:

1. Using *snapshots backups* with a Barman Server as the centralized Backup and Recovery manager.
2. Using *cloud backups* with cloud object storages to manage backups without a Barman Server.

In the first scenario, Barman uses `cron` for maintenance operations and enforcing the retention policy, as outlined in *How retention policies are enforced*. In this case, snapshot backups are treated the same as any other `rsync` or `postgres` backup.

In the second scenario, since there is no Barman server, you won't have cron for maintenance operations for enforcing the retention policy. Instead, you'll need to use `barman-cloud-backup-delete` with the `-r RETENTION_POLICY` option (see the command reference). This will delete any backups that do not meet the specified retention policy. Additionally, you can also schedule these commands using hook scripts or custom scripts to simulate cron maintenance for cloud backups.

Diagnose and troubleshooting

One of the features that tools require is the ability to troubleshoot problems in an efficient way. Barman provides multiple ways to diagnose and troubleshoot problems.

Usually problems arise from errors or warning messages returned by *barman check*, but they may come up from other sources as well.

12.1 Barman status

You can check the status of a specific Postgres server using the `barman status` command. This command provides a detailed view of the status of the server, such as the PostgreSQL version, the current data size, the PostgreSQL data directory, the current WAL, the archive command, the last archived WAL, the number of available backups, and more:

```
$ barman status pg16
Server pg16:
  Description: PostgreSQL 16 Database (via SSH)
  Active: True
  Disabled: False
  PostgreSQL version: 16.4
  Cluster state: in production
  Current data size: 31.9 MiB
  PostgreSQL Data directory: /var/lib/pgsql/16/data
  Current WAL segment: 000000010000003B00000010
  PostgreSQL 'archive_command' setting: barman-wal-archive barman_server pg16 %p
  Last archived WAL: 000000010000003B0000000F, at Mon Nov 18 07:54:54 2024
  Failures of WAL archiver: 0
  Passive node: False
  Retention policies: not enforced
  No. of available backups: 2
  First available backup: 20240911T085206
  Last available backup: 20240911T085736
  Minimum redundancy requirements: satisfied (2/0)
```

This is a good tool to collaborate that the server's configuration aligns with the configuration of the barman server.

12.2 Barman show servers

You can use the barman show-servers command to list all the configuration options and values for a specific Postgres server configured for backups.

```
$ barman show-servers pg16
Server local16:
  active: True
  archive_command: barman-wal-archive barman-server pg16 %p
  archive_mode: on
  archive_timeout: 0
  archived_count: 1
  ...
```

12.3 Barman replication status

You can do a quick check of the replication status for a specific Postgres server by using the barman replication-status command. This command provides a detailed view of the status of the replication.

```
$ barman replication-status pg16
Status of streaming clients for server 'pg16':
Current LSN on master: 3B/10000148
Number of streaming clients: 1

1. Async WAL streamer
  Application name: barman_receive_wal
  Sync stage      : 3/3 Remote write
  Communication   : Unix domain socket
  User name       : barman
  Current state   : streaming (async)
  Replication slot: standby
  WAL sender PID  : 165959
  Started at      : 2024-11-18 07:49:01.837787+00:00
  Sent LSN        : 3B/10000148 (diff: 0 B)
  Write LSN       : 3B/10000148 (diff: 0 B)
  Flush LSN       : 3B/10000000 (diff: -328 B)
```

12.4 Barman diagnose

The barman diagnose command gathers important information about the status of all the configured servers. It's an overall view of the configured Postgres servers that are being backed up by Barman.

```
barman diagnose
```

The diagnose command output is a full snapshot of the barman server, providing useful information, such as global configuration, SSH version, Python version, rsync version, PostgreSQL clients version, as well as current configuration and status of all servers.

The diagnose command is extremely useful for troubleshooting problems, as it gives a global view on the status of your Barman installation.

Configuration Reference

Barman follows a convention over configuration approach, which simplifies configuration by allowing some options to be defined globally and overridden at the server level. This means you can set a default behavior for your servers and then customize specific servers as needed. This design reduces the need for excessive configuration while maintaining flexibility.

13.1 Usage

Proper configuration is critical for its effective operation. Barman uses different types of configuration files to manage global settings, server-specific settings, and model-specific settings that is made up of three scopes:

1. **Global Configuration:** It comprises one file with a set of configurations for the barman system, such as the main directory, system user, log file, and other general options. Default location is `/etc/barman.conf` and it can be overridden on a per-user level by `~/.barman.conf` or by specifying a `.conf` file using the `-c / --config` with the barman command directly in the CLI.
2. **Server Configuration:** It comprises one or more files with a set of configurations for a Postgres server that you want to keep track and interact for backup, recovery and/or replication. Default location is `/etc/barman.d` and must use the `.conf` suffix. You may have one or multiple files for servers. You can override the default location by setting the `configuration_files_directory` option in the global configuration file and placing the files in that particular location.
3. **Model Configuration:** It comprises one or more files with a set of configurations overrides that can be applied to Barman servers within the same cluster as the model. These overrides can be implemented using the `barman config-switch` command. Default location is `/etc/barman.d` and must use the `.conf` suffix. The same `configuration_files_directory` override option from the server configuration applies for models. You may have one or multiple files for models.

Note

Historically, you could have a single configuration file containing global, server, and model options, but, for maintenance reasons, this approach is deprecated.

Configuration files follow the INI format and consist of sections denoted by headers in square brackets. Each section can include various options.

Models and servers must have unique identifiers, and reserved words cannot be used as names.

Reserved Words

The following reserved words cannot be used as server or model names:

- **barman**: Identifies the global section.
- **all**: A special shortcut for executing commands on all managed servers.

Parameter Types

Configuration options can be of the following types:

- **String**: Textual data (e.g., file paths, names).
- **Enum**: Enumerated values, often limited to predefined choices.
- **Integer**: Numeric values.
- **Boolean**: Can be **on**, **true**, **1** (**true**) or **off**, **false**, **0** (**false**).

Note

Some enums allow **off**, but not **false**.

13.2 Options

Options in the configuration files can have specific or shared scopes. The following configuration options are used not only for configuring how Barman will execute backups and recoveries, but also for configuring various aspects of how Barman interacts with the configured Postgres servers to be able to apply your Backup and Recovery, and High-Availability strategies.

13.2.1 General

These are general configurations options.

active

When this option is set to **true** (default), the server operates fully. If set to **false**, the server is restricted to diagnostic use only, meaning that operational commands such as backup execution or WAL archiving are temporarily disabled. When incorporating a new server into Barman, we recommend initially setting **active = false**. Verify that barman check shows no issues before activating the server. This approach helps prevent excessive error logging in Barman during the initial setup.

Scope: Server / Model.

archiver

This option enables log file shipping through Postgres' `archive_command` for a server. When set to **true**, Barman expects continuous archiving to be configured and will manage WAL files that Postgres stores in the incoming directory (`incoming_wals_directory`), including their checks, handling, and compression. When set to **false** (default), continuous archiving is disabled.

Note

If neither `archiver` nor `streaming_archiver` is configured, Barman will automatically set this option to `true` to maintain compatibility with the previous default behavior where archiving was enabled by default.

Scope: Global / Server / Model.

archiver_batch_size

This option enables batch processing of WAL files for the archiver process by setting it to a value greater than 0. If not set, the archiver will use unlimited (default) processing mode for the WAL queue. With batch processing enabled, the archiver process will handle a maximum of `archiver_batch_size` WAL segments per run. This value must be an integer.

Scope: Global / Server / Model.

bandwidth_limit

Specifies the maximum transfer rate in kilobytes per second for backup and recovery operations. A value of 0 indicates no limit (default).

Note

Applies only when `backup_method = postgres | rsync`.

Scope: Global / Server / Model.

barman_home

Designates the main data directory for Barman. Defaults to `/var/lib/barman`.

Scope: Global.

barman_lock_directory

Specifies the directory for lock files. The default is `barman_home`.

Note

The `barman_lock_directory` should be on a non-network local filesystem.

Scope: Global.

check_timeout

Sets the maximum execution time in seconds for a Barman check command per server. Set to 0 to disable the timeout. Default is 30 seconds. Must be a non-negative integer.

Scope: Global / Server / Model.

cluster

Tag the server or model to an associated cluster name. Barman uses this association to override configuration for all servers/models in this cluster. If omitted for servers, it defaults to the server's name.

Note

Must be specified for configuration models to group applicable servers.

Scope: Server / Model.

config_changes_queue

Designates the filesystem location for Barman's queue that handles configuration changes requested via the `barman config-update` command. This queue manages the serialization and retry of configuration change requests. By default, Barman writes to a file named `cfg_changes.queue` under `barman_home`.

Scope: Global.

configuration_files_directory

Designates the directory where server/model configuration files will be read by Barman. Defaults to `/etc/barman.d/`.

Scope: Global.

conninfo

Specifies the connection string used by Barman to connect to the Postgres server. This is a libpq connection string. Commonly used keys include: `host`, `hostaddr`, `port`, `dbname`, `user` and `password`. See the [libpq-connstring](#) PostgreSQL documentation for details.

Scope: Server / Model.

create_slot

Determines whether Barman should automatically create a replication slot if it's not already present for streaming WAL files. When set to `auto` and `slot_name` is defined, Barman will attempt to create the slot automatically. When set to `manual` (default), the replication slot must be created manually.

Scope: Global / Server / Model.

description

Provides a human-readable description of a server.

Scope: Server / Model.

errors_directory

The directory where WAL files that were errored while being archived by Barman are stored. This includes duplicate WAL files (e.g., an archived WAL file that has already been streamed but have different hash) and unexpected files found in the WAL archive directory.

The purpose of placing the files in this directory is so someone can later review why they failed to be archived and take appropriate actions (dispose of, store somewhere else, replace the duplicate file archived before, etc.)

Scope: Server.

encryption

Specifies the encryption method used for encrypting backups and WAL files. Supported values are:

- `none` (default): No encryption is applied.
- `gpg`: Uses [GPG](#) for encryption. Requires [GPG](#) to be installed and properly configured on the system.

Scope: Global / Server / Model.

encryption_key_id

Specifies the encryption key ID used for encrypting backups and WAL files. This option is required when `encryption = gpg` and must correspond to a valid *GPG* key ID available on the system.

Scope: Global / Server / Model.

encryption_passphrase_command

Specifies a command used to retrieve the encryption passphrase for decrypting backups and WAL files. The command must write the passphrase to standard output.

Scope: Global / Server / Model.

forward_config_path

Determines whether a passive node should forward its configuration file path to its primary node during `cron` or `sync-info` commands. Set to `true` if Barman is invoked with the `-c / --config` option and the configuration paths are identical on both passive and primary Barman servers. Defaults to `false`.

Scope: Global / Server / Model.

immediate_checkpoint

Controls how Postgres handles checkpoints at the start of a backup. Set to `false` (default) to allow the checkpoint to complete according to `checkpoint_completion_target`. Set to `true` for an immediate checkpoint, where Postgres completes the checkpoint as quickly as possible.

Scope: Global / Server / Model.

keepalive_interval

Sets the interval in seconds for sending a heartbeat query to keep the libpq connection active during an `rsync` backup. Default is 60 seconds. Setting this to 0 disables the heartbeat.

Scope: Global / Server / Model.

lock_directory_cleanup

Enables automatic cleanup of unused lock files in the `barman_lock_directory`.

Scope: Global.

log_file

Specifies the location of Barman's log file. Defaults to `/var/log/barman/barman.log`.

Scope: Global.

log_level

Sets the level of logging. Options include: `DEBUG`, `INFO`, `WARNING`, `ERROR` and `CRITICAL`.

Scope: Global.

minimum_redundancy

Specifies the minimum number of backups to retain. Default is 0.

Scope: Global / Server / Model.

model

When set to `true`, turns a server section from a configuration file into a model for a cluster. There is no `false` option in this case. If you want to simulate a `false` option, comment out (`#model=true`) or remove the option in the configuration. Defaults to the server name.

Scope: Model.

network_compression

Enables or disables data compression for network transfers. Set to `false` (default) to disable compression, or `true` to enable it and reduce network usage.

Scope: Global / Server / Model.

parallel_jobs

Controls the number of parallel workers used to copy files during backup or recovery. It must be a positive integer. Default is 1.

Note

Applies only when `backup_method = rsync`.

Scope: Global / Server / Model.

parallel_jobs_start_batch_period

Specifies the time interval in seconds over which a single batch of parallel jobs will start. Default is 1 second. This means that if `parallel_jobs_start_batch_size` is 10 and `parallel_jobs_start_batch_period` is 1, this will yield an effective rate limit of 10 jobs per second, because there is a maximum of 10 jobs that can be started within 1 second.

Note

Applies only when `backup_method = rsync`.

Scope: Global / Server / Model.

parallel_jobs_start_batch_size

Defines the maximum number of parallel jobs to start in a single batch. Default is 10 jobs. This means that if `parallel_jobs_start_batch_size` is 10 and `parallel_jobs_start_batch_period` is 2, this will yield a maximum of 10 jobs that can be started within 2 seconds.

Note

Applies only when `backup_method = rsync`.

Scope: Global / Server / Model.

path_prefix

Lists one or more absolute paths, separated by colons, where Barman looks for executable files such as PostgreSQL binaries (from the appropriate `bin` directory for the `PG_MAJOR_VERSION`), `rsync`, encryption, and compression tools. These paths are prepended to the `PATH` environment variable and are checked before any others.

Scope: Global / Server / Model.

primary_checkpoint_timeout

Time to wait for new WAL files before forcing a checkpoint on the primary server. Defaults to 0.

Scope: Server / Model.

primary_conninfo

Connection string for Barman to connect to the primary Postgres server during a standby backup.

Scope: Server / Model.

primary_ssh_command

SSH command for connecting to the primary Barman server if Barman is passive.

Scope: Global / Server / Model.

slot_name

Replication slot name for the `receive-wal` command when `streaming_archiver` is enabled.

Scope: Global / Server / Model.

ssh_command

SSH command used by Barman to connect to the Postgres server for rsync backups.

Scope: Server / Model.

streaming_archiver

Enables Postgres' streaming protocol for WAL files. Defaults to `false`.

Note

If neither `archiver` nor `streaming_archiver` is configured, Barman will automatically set `archiver` option to `true` to maintain compatibility with the previous default behavior where archiving was enabled by default.

Scope: Global / Server / Model.

streaming_archiver_batch_size

Batch size for processing WAL files in streaming archiver. Defaults to `0`.

Scope: Global / Server / Model.

streaming_archiver_name

Application name for the `receive-wal` command. Defaults to `barman_receive_wal`.

Scope: Global / Server / Model.

streaming_backup_name

Application name for the `pg_basebackup` command. Defaults to `barman_streaming_backup`.

Scope: Global / Server / Model.

streaming_conninfo

Connection string for streaming replication protocol. Defaults to `conninfo`.

Scope: Server / Model.

tablespace_bandwidth_limit

Maximum transfer rate for specific tablespaces for backup and recovery operations. A value of `0` indicates no limit (default).

Note

Applies only when `backup_method = rsync`.

Scope: Global / Server / Model.

13.2.2 Backups

These configurations options are related to how Barman will execute backups.

autogenerate_manifest

This is a boolean option that allows for the automatic creation of backup manifest files. The manifest file, which is a JSON document, lists all files included in the backup. It is generated upon completion of the backup and saved in the backup directory. The format of the manifest file adheres to the specifications outlined in the [backup manifest format](#) PostgreSQL documentation and is compatible with the `pg_verifybackup` tool. Default is `false`.

Note

This option is ignored if the `backup_method` is not `rsync`.

Scope: Global / Server / Model.

backup_compression

Specifies the compression method for the backup process. It can be set to `gzip`, `lz4`, `zstd`, or `none`. Ensure that the CLI tool for the chosen compression method is available on both the Barman and Postgres servers.

Note

Note that `lz4` and `zstd` require Postgres version 15 or later. Unsetting this option or using `none` results in an uncompressed archive (default). Only supported when `backup_method = postgres`.

Scope: Global / Server / Model.

backup_compression_format

Determines the format `pg_basebackup` should use when saving compressed backups. Options are `plain` or `tar`, with `tar` as the default if unset. The `plain` format is available only if Postgres version 15 or later is in use and `backup_compression_location` is set to `server`.

Note

Only supported when `backup_method = postgres`.

Scope: Global / Server / Model.

backup_compression_level

Defines the level of compression for backups as an integer. The permissible values depend on the compression method specified in `backup_compression`.

Note

Only supported when `backup_method = postgres`.

Scope: Global / Server / Model.

backup_compression_location

Specifies where compression should occur during the backup: either `client` or `server`. The `server` option is available only if Postgres version 15 or later is being used.

Note

Only supported when `backup_method = postgres`.

Scope: Global / Server / Model.

backup_compression_workers

Sets the number of threads used for compression during the backup process. This is applicable only when `backup_compression=zstd`. The default value is 0, which uses the standard compression behavior.

Note

Only supported when `backup_method = postgres`.

Scope: Global / Server / Model.

backup_directory

Specifies the directory where backup data for a server will be stored. Defaults to `<barman_home>/<server_name>`.

Scope: Server.

backup_method

Defines the method Barman uses to perform backups. Options include:

- `rsync` (default): Executes backups using the `rsync` command over SSH (requires `ssh_command`).
- `postgres`: Uses the `pg_basebackup` command for backups.
- `local-rsync`: Assumes Barman runs on the same server and as the same user as the Postgres database, performing an `rsync` file system copy.
- `snapshot`: Utilizes the API of the cloud provider specified in the `snapshot_provider` option to create disk snapshots as defined in `snapshot_disks` and saves only the backup label and metadata to its own storage.

Scope: Global / Server / Model.

backup_options

Controls how Barman interacts with Postgres during backups. This is a comma-separated list that can include:

- `concurrent_backup` (default): Uses concurrent backup, recommended for Postgres versions 9.6 and later, and supports backups from standby servers.
- `exclusive_backup`: Uses the deprecated exclusive backup method. Only for Postgres versions older than 15. This option will be removed in the future.
- `external_configuration`: Suppresses warnings about external configuration files during backup execution.

Note

`exclusive_backup` and `concurrent_backup` cannot be used together.

Scope: Global / Server / Model.

basebackups_directory

Specifies the directory where base backups are stored. Defaults to <backup_directory>/base.

Scope: Server.

basebackup_retry_sleep

Sets the number of seconds to wait after a failed base backup copy before retrying. Default is 30 seconds. Must be a non-negative integer.

Note

This applies to both backup and recovery operations.

Scope: Global / Server / Model.

basebackup_retry_times

Defines the number of retry attempts for a base backup copy after an error occurs. Default is 0 (no retries). Must be a non-negative integer.

Note

This applies to both backup and recovery operations.

Scope: Global / Server / Model.

reuse_backup

Controls incremental backup support when using backup_method=rsync by reusing the last available backup. The options are:

- **off** (default): Standard full backup.
- **copy**: File-level incremental backup, by reusing the last backup for a server and creating a copy of the unchanged files (just for backup time reduction)
- **link**: File-level incremental backup, by reusing the last backup for a server and creating a hard link of the unchanged files (for backup space and time reduction)

Note

This option will be ignored when backup_method=postgres.

Scope: Global / Server / Model.

worm_mode

If set to **on**, enables support for WORM (Write Once Read Many) storage, allowing Barman to handle backups on immutable storage correctly. Default is **off**.

Scope: Global / Server / Model.

13.2.3 Cloud Backups

These configuration options are related to how Barman will execute backups in the cloud.

aws_await_snapshots_timeout

Specifies the duration in seconds to wait for AWS snapshots to be created before a timeout occurs. The default value is 3600 seconds. This must be a positive integer.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_profile

The name of the AWS profile to use when authenticating with AWS (e.g. INI section in AWS credentials file).

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_region

Indicates the AWS region where the EC2 VM and storage volumes, as defined by `snapshot_instance` and `snapshot_disks`, are located.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_snapshot_lock_mode

The lock mode for the snapshot. This is only valid if `snapshot_instance` and `snapshot_disk` are set.

Allowed options:

- `compliance`.
- `governance`.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_snapshot_lock_duration

The lock duration is the period of time (in days) for which the snapshot is to remain locked, ranging from 1 to 36,500. Set either the lock duration or the expiration date (not both).

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_snapshot_lock_cool_off_period

The cooling-off period is an optional period of time (in hours) that you can specify when you lock a snapshot in compliance mode, ranging from 1 to 72.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

aws_snapshot_lock_expiration_date

The lock duration is determined by an expiration date in the future. It must be at least 1 day after the snapshot creation date and time, using the format `YYYY-MM-DDTHH:MM:SS.sssZ`. Set either the lock duration or the expiration date (not both).

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = aws`.

Scope: Global / Server / Model.

azure_credential

Specifies the type of Azure credential to use for authentication, either `azure-cli`, `managed-identity` or `default`. If not provided, the default Azure authentication method will be used.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = azure`.

Scope: Global / Server / Model.

azure_resource_group

Specifies the name of the Azure resource group containing the compute instance and disks defined by `snapshot_instance` and `snapshot_disks`.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = azure`.

Scope: Global / Server / Model.

azure_subscription_id

Identifies the Azure subscription that owns the instance and storage volumes defined by `snapshot_instance` and `snapshot_disks`.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = azure`.

Scope: Global / Server / Model.

gcp_project

Specifies the ID of the GCP project that owns the instance and storage volumes defined by `snapshot_instance` and `snapshot_disks`.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = gcp`.

Scope: Global / Server / Model.

gcp_zone

Indicates the availability zone where the compute instance and disks are located for snapshot backups.

Note

Only supported when `backup_method = snapshot` and `snapshot_provider = gcp`.

Scope: Server / Model.

snapshot_disks

This option is a comma-separated list of disks to include in cloud snapshot backups.

Note

Required when `backup_method = snapshot`.

Ensure that the `snapshot_disks` list includes all disks that store Postgres data, as any data not on these listed disks will not be included in the backup and will be unavailable during recovery.

Scope: Server / Model.

snapshot_instance

The name of the VM or compute instance where the storage volumes are attached.

Note

Required when `backup_method = snapshot`.

Scope: Server / Model.

snapshot_provider

13.2. Options

The name of the cloud provider to use for creating snapshots. Supported value: `aws`, `azure` and `gcp`.

Note

Required when `backup_method = snapshot`.

Scope: Global / Server / Model.

13.2.4 Hook Scripts

These configuration options are related to the pre or post execution of hook scripts.

post_archive_retry_script

Specifies a hook script to run after a WAL file is archived. Barman will retry this script until it returns `SUCCESS` (0), `ABORT_CONTINUE` (62), or `ABORT_STOP` (63). In a post-archive scenario, `ABORT_STOP` has the same effect as `ABORT_CONTINUE`.

Scope: Global / Server.

post_archive_script

Specifies a hook script to run after a WAL file is archived, following the `post_archive_retry_script`.

Scope: Global / Server.

post_backup_retry_script

Specifies a hook script to run after a base backup. Barman will retry this script until it returns `SUCCESS` (0), `ABORT_CONTINUE` (62), or `ABORT_STOP` (63). In a post-backup scenario, `ABORT_STOP` has the same effect as `ABORT_CONTINUE`.

Scope: Global / Server.

post_backup_script

Specifies a hook script to run after a base backup, following the `post_backup_retry_script`.

Scope: Global / Server.

post_delete_retry_script

Specifies a hook script to run after deleting a backup. Barman will retry this script until it returns `SUCCESS` (0), `ABORT_CONTINUE` (62), or `ABORT_STOP` (63). In a post-delete scenario, `ABORT_STOP` has the same effect as `ABORT_CONTINUE`.

Scope: Global / Server.

post_delete_script

Specifies a hook script to run after deleting a backup, following the `post_delete_retry_script`.

Scope: Global / Server.

post_recovery_retry_script

Specifies a hook script to run after a recovery. Barman will retry this script until it returns `SUCCESS` (0), `ABORT_CONTINUE` (62), or `ABORT_STOP` (63). In a post-recovery scenario, `ABORT_STOP` has the same effect as `ABORT_CONTINUE`.

Scope: Global / Server.

post_recovery_script

Specifies a hook script to run after a recovery, following the `post_recovery_retry_script`.

Scope: Global / Server.

post_wal_delete_retry_script

Specifies a hook script to run after deleting a WAL file. Barman will retry this script until it returns SUCCESS (0), ABORT_CONTINUE (62), or ABORT_STOP (63). In a post-WAL-delete scenario, ABORT_STOP has the same effect as ABORT_CONTINUE.

Scope: Global / Server.

post_wal_delete_script

Specifies a hook script to run after deleting a WAL file, following the `post_wal_delete_retry_script`.

Scope: Global / Server.

pre_archive_retry_script

Specifies a hook script that runs before a WAL file is archived during maintenance, following the `pre_archive_script`. As a retry hook script, Barman will repeatedly execute the script until it returns either SUCCESS (0), ABORT_CONTINUE (62), or ABORT_STOP (63). Returning ABORT_STOP will escalate the failure and halt the WAL archiving process.

Scope: Global / Server.

pre_archive_script

Specifies a hook script launched before a WAL file is archived by maintenance.

Scope: Global / Server.

pre_backup_retry_script

Specifies a hook script that runs before a base backup, following the `pre_backup_script`. As a retry hook script, Barman will attempt to execute the script repeatedly until it returns SUCCESS (0), ABORT_CONTINUE (62), or ABORT_STOP (63). Returning ABORT_STOP will escalate the failure and interrupt the backup process.

Scope: Global / Server.

pre_backup_script

Specifies a hook script to run before starting a base backup.

Scope: Global / Server.

pre_delete_retry_script

Specifies a retry hook script to run before backup deletion, following the `pre_delete_script`. As a retry hook script, Barman will attempt to execute the script repeatedly until it returns SUCCESS (0), ABORT_CONTINUE (62), or ABORT_STOP (63). Returning ABORT_STOP will escalate the failure and interrupt the backup deletion.

Scope: Global / Server.

pre_delete_script

Specifies a hook script run before deleting a backup.

Scope: Global / Server.

pre_recovery_retry_script

Specifies a retry hook script to run before recovery, following the `pre_recovery_script`. As a retry hook script, Barman will attempt to execute the script repeatedly until it returns SUCCESS (0), ABORT_CONTINUE (62), or ABORT_STOP (63). Returning ABORT_STOP will escalate the failure and interrupt the recover process.

Scope: Global / Server.

pre_recovery_script

Specifies a hook script run before starting a recovery.

Scope: Global / Server.

pre_wal_delete_retry_script

Specifies a retry hook script for WAL file deletion, executed before `pre_wal_delete_script`. As a retry hook script, Barman will attempt to execute the script repeatedly until it returns `SUCCESS` (0), `ABORT_CONTINUE` (62), or `ABORT_STOP` (63). Returning `ABORT_STOP` will escalate the failure and interrupt the WAL file deletion.

Scope: Global / Server.

pre_wal_delete_script

Specifies a hook script run before deleting a WAL file.

Scope: Global / Server.

13.2.5 Write-Ahead Logs (WAL)

These configuration options are related to how Barman will manage the Write-Ahead Logs (WALs) of the PostgreSQL servers.

compression

Specifies the standard compression algorithm for WAL files. Options include: `lz4`, `xz`, `zstd`, `gzip`, `pygzip`, `pigz`, `bzip2`, `pybzip2`, `snappy` and `custom`.

Note

All of these options require the module to be installed in the location where the compression will occur.

The `custom` option is for custom compression, which requires you to set the following options as well:

- `custom_compression_filter`: a compression filter.
- `custom_decompression_filter`: a decompression filter
- `custom_compression_magic`: a hex string to identify a custom compressed wal file.

Scope: Global / Server / Model.

custom_compression_filter

Specifies a custom compression algorithm for WAL files. It must be a `string` that will be used internally to create a bash command and it will prefix to the following string `> "$2" < "$1";`. Write to standard output and do not delete input files.

Tip

```
custom_compression_filter = "xz -c"
```

This is the same as running `xz -c > "$2" < "$1";`.

Scope: Global / Server / Model.

custom_compression_magic

Defines a custom magic value to identify the custom compression algorithm used in WAL files. If this is set, Barman will avoid applying custom compression to WALs that have already been compressed with the specified algorithm. If not configured, Barman will apply custom compression to all WAL files, even those pre-compressed.

Tip

For example, in the `xz` compression algorithm, the magic number is used to detect the format of `.xz` files.

For `xz` files, the magic number is the following sequence of bytes:

Magic Number: FD 37 7A 58 5A 00

In hexadecimal representation, this can be expressed as:

Hex String: fd377a585a00

As Barman expects the value of `custom_compression_magic` to be prefixed with `0x`, you would need to set that config option like this:

```
custom_compression_magic = 0xfd377a585a00
```

Reference: [xz-file-format](#)

Scope: Global / Server / Model.

custom_decompression_filter

Specifies a custom decompression algorithm for compressed WAL files. It must be a `string` that will be used internally to create a bash command and it will prefix to the following string `> "$2" < "$1";`. It must correspond with the compression algorithm used.

Tip

```
custom_compression_filter = "xz -c -d"
```

This is the same as running `xz -c -d > "$2" < "$1";`.

Scope: Global / Server / Model.

compression_level

Specifies the compression level to be used by the selected compression algorithm. Valid values are integers within the supported range of the chosen algorithm or one of the predefined labels: `low`, `medium`, and `high`, which serve as shortcuts.

- `low`: uses low level of compression, favoring compression speed over compression ratio.
- `medium`: uses a medium level of compression, balancing between compression speed and compression ratio.
- `high`: uses a high level of compression, favoring compression ratio over compression speed.

Predefined labels map to algorithm-specific levels, as detailed below:

Table 1: Compression levels

Algorithm	Level range	low	medium	high
lz4	0 to 16	0	6	10
xz	1 to 9	1	3	5
zstd	-22 to 22	1	4	9
gzip, pigzip and pigz	1 to 9	1	6	9
bzip2 and pybzip2	1 to 9	1	5	9

If the specified compression level is greater than the algorithm's maximum level, that maximum level is used. Similarly, if it is lower than the minimum level, that minimum level is used. The default value is `medium`.

Scope: Global / Server / Model.

incoming_wals_directory

Specifies the directory where incoming WAL files are archived. Requires `archiver` to be enabled. Defaults to `<backup_directory>/incoming`.

Scope: Server.

last_wal_maximum_age

Defines the time frame within which the latest archived WAL file must fall. If the latest WAL file is older than this period, the barman check command will report an error. If left empty (default), the age of the WAL files is not checked. Format is the same as `last_backup_maximum_age`.

Scope: Global / Server / Model.

max_incoming_wals_queue

Defines the maximum number of WAL files allowed in the incoming queue (including both streaming and archiving pools) before the barman check command returns an error. Default is `None` (disabled).

Scope: Global / Server / Model.

streaming_wals_directory

Directory for streaming WAL files. Defaults to `<backup_directory>/streaming`.

Note

This option is applicable when `streaming_archiver` is activated.

Scope: Server.

wal_conninfo

The `wal_conninfo` connection string is used by Barman for monitoring the status of the replication slot receiving WALs. If specified, it takes precedence over `wal_streaming_conninfo` for these checks. If `wal_conninfo` is not set but `wal_streaming_conninfo` is, `wal_conninfo` will fall back to `wal_streaming_conninfo`. If neither `wal_conninfo` nor `wal_streaming_conninfo` is set, `wal_conninfo` will fall back to `conninfo`. Both connection strings must access a Postgres instance within the same cluster as defined by `streaming_conninfo` and `conninfo`. If both `wal_conninfo` and `wal_streaming_conninfo` are set, only `wal_conninfo` needs the appropriate permissions to read settings and check the replication slot status. However, if only `wal_streaming_conninfo` is set, it must have the necessary permissions to perform these tasks. The required permissions include roles such as `pg_monitor`, both `pg_read_all_settings` and `pg_read_all_stats`, or superuser privileges.

Scope: Server / Model.

wal_streaming_conninfo

This connection string is used by Barman to connect to the Postgres server for receiving WAL segments via streaming replication and checking the replication slot status, if `wal_conninfo` is not set. If not specified, Barman defaults to using `streaming_conninfo` for these tasks. `wal_streaming_conninfo` must connect to a Postgres instance within the same cluster as defined by `streaming_conninfo`, and it must support streaming replication. If both `wal_streaming_conninfo` and `wal_conninfo` are set, only `wal_conninfo` needs the required permissions to read settings and check the replication slot status. If only `wal_streaming_conninfo` is specified, it must have these permissions. The necessary permissions include roles such as `pg_monitor`, both `pg_read_all_settings` and `pg_read_all_stats`, or superuser privileges.

Scope: Server / Model.

wals_directory

Directory containing WAL files. Defaults to `<backup_directory>/wals`.

Scope: Server.

xlogdb_directory

A custom directory for the `SERVER-xlog.db` file, `SERVER` being the server name. This file stores metadata of archived WAL files and is used internally by Barman. If unset, it defaults to the value of `wals_directory`.

Scope: Global / Server.

13.2.6 Restore

These configuration options are related to how Barman manages restoration backups.

local_staging_path

Specifies the local path for combining block-level incremental backups during recovery. This location must have sufficient space to temporarily store the new synthetic backup. Required for recovery from a block-level incremental backup.

Note

Applies only when `backup_method = postgres`.

Scope: Global / Server / Model.

recovery_options

Options for recovery operations. Currently, only `get-wal` is supported. This option enables the creation of a basic `restore_command` in the recovery configuration, which uses the `barman get-wal` command to retrieve WAL files directly from Barman's WAL archive. This setting accepts a comma-separated list of values and defaults to empty.

Scope: Global / Server / Model.

recovery_staging_path

Specifies the path on the recovery host for staging files from compressed backups. This location must have sufficient space to temporarily store the compressed backup.

Note

Applies only for compressed backups.

Scope: Global / Server / Model.

staging_path

A path where intermediate files are staged during restore. When restoring a compressed backup, it serves as a temporary location for decompression before copying to the final destination. When restoring an incremental backup, it is where backups are combined before copying to the final destination. This location must have enough space to store the decompressed/combined backup.

Scope: Global / Server / Model.

staging_location

Specifies whether `staging_path` is a local or remote path. Valid values are `local` and `remote`.

Scope: Global / Server / Model.

13.2.7 Retention Policies

These configuration options are related to how Barman manages retention policies of the backups.

last_backup_maximum_age

Defines the time frame within which the latest backup must fall. If the latest backup is older than this period, the barman check command will report an error. If left empty (default), the latest backup is always considered valid. The accepted format is "`n {DAYS|WEEKS|MONTHS|HOURS}`", where `n` is an integer greater than zero.

Scope: Global / Server / Model.

last_backup_minimum_size

Specifies the minimum acceptable size for the latest successful backup. If the latest backup is smaller than this size, the barman check command will report an error. If left empty (default), the latest backup is always considered valid. The accepted format is "`n {k|Ki|M|Mi|G|Gi|T|Ti}`" and case-sensitive, where `n` is an integer greater than zero, with an optional SI or IEC suffix. `k` stands for kilo with `k = 1000`, while `Ki` stands for kilobytes `Ki = 1024`. The rest of the options have the same reasoning for greater units of measure.

Scope: Global / Server / Model.

retention_policy

Defines how long backups and WAL files should be retained. If this option is left blank, no retention policies will be applied. Options include redundancy and recovery window policies.

`retention_policy = {REDUNDANCY value | RECOVERY WINDOW OF value {DAYS | WEEKS | MONTHS}}`

- `retention_policy = REDUNDANCY 2` will keep only 2 backups in the backup catalog automatically deleting the older one as new backups are created. The number must be a positive integer.
- `retention_policy = RECOVERY WINDOW OF 2 DAYS` will only keep backups needed to recover to any point in time in the last two days, automatically deleting backups that are older. The period number must be a positive integer, and the following options can be applied to it: `DAYS`, `WEEKS`, `MONTHS`.

Scope: Global / Server / Model.

retention_policy_mode

Mode for enforcing retention policies. Currently only supports `auto`.

Scope: Global / Server / Model.

wal_retention_policy

Policy for retaining WAL files. Currently only `main` is available.

Scope: Global / Server / Model.

13.3 Configuration Models

Configuration models provide a systematic approach to manage and apply configuration overrides for Postgres servers by organizing them under a specific `cluster` name.

13.3.1 Purpose

The primary goal of a configuration model is to simplify the management of configuration settings for Postgres servers grouped by the same `cluster`. By using a model, you can apply a set of common configuration overrides, enhancing operational efficiency. They are especially beneficial in clustered environments, allowing you to create various configuration models that can be utilized during failover events.

13.3.2 Application

The configurations defined in a model file can be applied to Postgres servers that share the same `cluster` name specified in the model. Consequently, any server utilizing that model can inherit these settings, promoting a consistent and adaptable configuration across all servers.

13.3.3 Usage

Model options can only be defined within a model section, which is identified in the same way as a server section. It is important to ensure that there are no conflicts between the identifiers of server sections and model sections.

To apply a configuration model, execute the `barman config-switch SERVER_NAME MODEL_NAME`. This command facilitates the application of the model's overrides to the relevant Barman server associated with the specified cluster name.

If you wish to remove the overrides, the deletion of the model configuration file alone will not have any effect, so you can do so by using the `--reset` argument with the command, as follows: `barman config-switch SERVER_NAME --reset`.

Note

The `config-switch` command will only succeed if model name exists and is associated with the same `cluster` as the server. Additionally, there can be only one active model at a time; if you execute the command multiple times with different models, only the overrides defined in the last model will be applied.

Not all options can be configured through models. Please review the scope of the available configurations to determine which settings apply to models.

13.3.4 Benefits

- **Consistency:** Ensures uniform configuration across multiple Barman servers within a cluster.
- **Efficiency:** Simplifies configuration management by allowing centralized updates and overrides.
- **Flexibility:** Allows the use of multiple model files, providing the ability to define various sets of overrides as necessary.

Commands Reference

Barman has a command-line interface named `barman`, which is used basically to interact with Barman's backend.

Before jumping into each of the sub-commands of `barman`, be aware that `barman` has global options available for all of the sub-commands. These options can modify the behavior of the sub-commands and can be used as follows:

14.1 `barman`

14.1.1 Synopsis

```
barman
[ { -c | --config } CONFIG ]
[ { --color | --colour } { never | always | auto } ]
[ { -d | --debug } ]
[ { -f | --format } { json | console } ]
[ { -h | --help } ]
[ --log-level { NOTSET | DEBUG | INFO | WARNING | ERROR | CRITICAL } ]
[ { -q | --quiet } ]
[ { -v | --version } ]
[ SUBCOMMAND ]
```

Note

This is the syntax for the synopsis:

- Options between square brackets are optional.
- Options between curly brackets represent a choose one of set operation.
- Options with [...] can be specified multiple times.
- Things written in uppercase represent a literal that should be given a value to.

We will use this same syntax when describing **barman** sub-commands in the following sections.

Also, when describing sub-commands in the following sections, the commands' synopsis should be seen as a replacement for the SUBCOMMAND.

14.1.2 Parameters

-c / --config CONFIG

Specify the configuration file to be used. Defaults to `/etc/barman.conf` if not provided.

--color / --colour { never | always | auto }

Control whether to use colors in the output. The default is `auto`. Options are:

- **never**: Do not use color.
- **always**: Always use color.
- **auto**: Use color if the output is to a terminal.

-d / --debug

Enable debug output. Default is `false`. Provides detailed logging information for troubleshooting.

-f / --format { json | console }

Specify the output format. Options are:

- **json**: Output in JSON format.
- **console**: Output in human-readable format (default).

-h / --help

Show a help message and exit. Provides information about command usage.

--log-level { NOTSET | DEBUG | INFO | WARNING | ERROR | CRITICAL }

Override the default logging level. Options are:

- **NOTSET**: This is the default level when no specific logging level is set. It essentially means "no filtering" of log messages, allowing all messages to be processed according to the levels that are set in the configuration.
- **DEBUG**: This level is used for detailed, diagnostic information, often useful for developers when diagnosing problems. It includes messages that are more granular and detailed, intended to help trace the execution of the program.
- **INFO**: This level provides general information about the application's normal operation. It's used for messages that indicate the progress of the application or highlight key points in the execution flow that are useful but not indicative of any issues.
- **WARNING**: This level indicates that something unexpected happened or that there might be a potential problem. It's used for messages that are not critical but could be of concern, signaling that attention might be needed.
- **ERROR**: This level is used when an error occurs that prevents a particular operation from completing successfully. It's used to indicate significant issues that need to be addressed but do not necessarily stop the application from running.
- **CRITICAL**: This is the highest level of severity, indicating a serious error that has likely caused the application to terminate or will have severe consequences if not addressed immediately. It's used for critical issues that demand urgent attention.

-q / --quiet

Suppress all output. Useful for cron jobs or automated scripts.

-v / --version

Show the program version number and exit.

14.2 Shortcuts

For some commands, you can use the following shortcuts or aliases to identify a backup for a given server. Specifically, the `all` shortcut can be used to identify all servers:

Shortcut	Description
all	All available servers
first/oldest	Oldest available backup for the server, in chronological order.
last/latest	Most recent available backup for the server, in chronological order.
last-full/latest-full	Most recent full backup taken with methods <code>rsync</code> or <code>postgres</code> .
last-failed	Most recent backup that failed, in chronological order.

14.3 Exit Statuses

Status code **0** means **success**, while status code **Non-Zero** means **failure**.

14.4 Sub-Commands

`barman` exposes several handy operations. This section is intended to describe each of them.

In the following sections you can find a description of each command implemented by `barman`. Some of these commands may have more detailed information in another main section in this documentation. If that is the case, a reference is provided to help you quickly navigate to it.

14.4.1 `barman archive-wal`

Synopsis

```
archive-wal
  [ { -h | --help } ]
  SERVER_NAME
```

Description

Fetch WAL files received from either the standard `archive_command` or streaming replication with `pg_receivewal` and store them in the server's WAL archive. If you have enabled `compression` in the configuration file, the WAL files will be compressed before they are archived.

Parameters

SERVER_NAME

Name of the server in barman node.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.2 barman backup

Synopsis

```
backup
[ --bqlimit KBPS ]
[ { -h | --help } ]
[ --incremental BACKUP_ID ]
[ --immediate-checkpoint ]
[ { -j | --jobs } PARALLEL_WORKERS ]
[ --jobs-start-batch-period PERIOD ]
[ --jobs-start-batch-size SIZE ]
[ --keepalive-interval SECONDS ]
[ --manifest ]
[ --name NAME ]
[ --no-immediate-checkpoint ]
[ --no-manifest ]
[ --no-retry ]
[ --retry-sleep SECONDS ]
[ --retry-times NUMBER ]
[ --reuse-backup { off | copy | link } ]
[ { --wait | -w } ]
[ --wait-timeout SECONDS ]
SERVER_NAME [ SERVER_NAME ... ]
```

Description

Execute a PostgreSQL server backup. Barman will use the parameters specified in the Global and Server configuration files. Specify all shortcut instead of the server name to execute backups from all servers configured in the Barman node. You can also specify multiple server names in sequence to execute backups for specific servers.

Parameters

SERVER_NAME

Name of the server in barman node.

--bqlimit

Specify the maximum transfer rate in kilobytes per second. A value of 0 indicates no limit. This setting overrides the `bandwidth_limit` configuration option.

-h / --help

Show a help message and exit. Provides information about command usage.

--incremental

Execute a block-level incremental backup. You must provide a `BACKUP_ID` or a shortcut to a previous backup, which will serve as the parent backup for the incremental backup.

Note

The backup to be and the parent backup must have `backup_method=postgres`.

--immediate-checkpoint

Forces the initial checkpoint to be executed as soon as possible, overriding any value set for the `immediate_checkpoint` parameter in the configuration file.

-j / --jobs

Specify the number of parallel workers to use for copying files during the backup. This setting overrides the `parallel_jobs` parameter if it's specified in the configuration file.

--jobs-start-batch-period

Specify the time period, in seconds, for starting a single batch of jobs. This value overrides the `parallel_jobs_start_batch_period` parameter if it is set in the configuration file. The default is 1 second.

--jobs-start-batch-size

Specify the maximum number of parallel workers to initiate in a single batch. This value overrides the `parallel_jobs_start_batch_size` parameter if it is defined in the configuration file. The default is 10 workers.

--keepalive-interval

Specify an interval, in seconds, for sending a heartbeat query to the server to keep the libpq connection active during a Rsync backup. The default is 60 seconds. A value of 0 disables the heartbeat.

--manifest

Forces the creation of a backup manifest file upon completing a backup. Overrides the `autogenerate_manifest` parameter from the configuration file. Applicable only to rsync backup strategy.

--name

Specify a friendly name for this backup which can be used in place of the backup ID in barman commands.

--no-immediate-checkpoint

Forces the backup to wait for the checkpoint to be executed overriding any value set for the `immediate_checkpoint` parameter in the configuration file.

--no-manifest

Disables the automatic creation of a backup manifest file upon completing a backup. This setting overrides the `autogenerate_manifest` parameter from the configuration file and applies only to rsync backup strategy.

--no-retry

There will be no retry in case of an error. It is the same as setting `--retry-times 0`.

--retry-sleep

Specify the number of seconds to wait after a failed copy before retrying. This setting applies to both backup and recovery operations and overrides the `basebackup_retry_sleep` parameter if it is defined in the configuration file.

--retry-times

Specify the number of times to retry the base backup copy in case of an error. This applies to both backup and recovery operations and overrides the `basebackup_retry_times` parameter if it is set in the configuration file.

--reuse-backup

Overrides the behavior of the `reuse_backup` option configured in the configuration file. The possible values are:

- `off`: Do not reuse the last available backup.
- `copy`: Reuse the last available backup for a server and create copies of unchanged files (reduces backup time).
- `link` (default): Reuse the last available backup for a server and create hard links to unchanged files (saves both backup time and space).

Note

This will only have any effect if the last available backup was executed with `backup_method=rsync`.

--wait / -w

Wait for all necessary WAL files required by the base backup to be archived.

--wait-timeout

Specify the duration, in seconds, to wait for the required WAL files to be archived before timing out.

14.4.3 barman check-backup

Synopsis

```
check-backup
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

Check that all necessary WAL files for verifying the consistency of a physical backup are properly archived. This command is automatically executed by the cron job and at the end of each backup operation. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node.

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.4 barman check

Synopsis

```
check
[ { -h | --help } ]
[ --nagios ]
SERVER_NAME [ SERVER_NAME ... ]
```

Description

Display status information about a server, such as SSH connection, Postgres version, configuration and backup directories, archiving and streaming processes, replication slots, and more. Use `all` as shortcut to show diagnostic information for all configured servers.

Parameters

SERVER_NAME

Name of the server in barman node.

-h / --help

Show a help message and exit. Provides information about command usage.

--nagios

Nagios plugin compatible output.

14.4.5 barman config-switch

Synopsis

```
config-switch
  [ { -h | --help } ]
  SERVER_NAME { --reset | MODEL_NAME }
```

Description

Apply a set of configuration overrides from the model to a server in Barman. The final configuration will combine or override the server's existing settings with the ones specified in the model. You can reset the server configurations with the `--reset` argument.

Note

Only one model can be active at a time for a given server.

Parameters

SERVER_NAME

Name of the server in barman node.

MODEL_NAME

Name of the model.

-h / --help

Show a help message and exit. Provides information about command usage.

--reset

Reset the server's configurations.

14.4.6 barman config-update

Synopsis

```
config-update
  [ { -h | --help } ]
  STRING
```

Description

Create or update the configurations for servers and/or models in Barman. The parameter should be a JSON string containing an array of documents. Each document must include a `scope` key, which can be either `server` or `model`, and either a `server_name` or `model_name` key, depending on the `scope` value. Additionally, the document should include other keys representing Barman configuration options and their desired values.

Note

The `barman config-update` command writes configuration options to a file named `.barman.auto.conf`, located in the `barman_home` directory. This configuration file has higher precedence and will override values from the global Barman configuration file (usually `/etc/barman.conf`) and from any included files specified in `configuration_files_directory` (typically files in `/etc/barman.d`). Be aware of this if you decide to manually modify configuration options in those files later.

Parameters

STRING

List of JSON formatted string.

`-h / --help`

Show a help message and exit. Provides information about command usage.

Example

```
JSON_STRING='[{ "scope" : "server", "server_name" : "my_server", "archiver" : "on",  
  "streaming_archiver" : "off" }]'
```

14.4.7 `barman cron`

Synopsis

```
cron  
  [ { -h | --help } ]  
  [ --keep-descriptors ]
```

Description

Carry out maintenance tasks, such as enforcing retention policies or managing WAL files.

Parameters

`-h / --help`

Show a help message and exit. Provides information about command usage.

`--keep-descriptors`

Keep the `^stdout^` and `^stderr^` streams of the Barman subprocesses connected to the main process. This is especially useful for Docker-based installations.

14.4.8 `barman delete`

Synopsis

```
delete
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

Delete the specified backup. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.9 barman diagnose

Synopsis

```
diagnose
[ { -h | --help } ]
[ --show-config-source ]
```

Description

Display diagnostic information about the Barman node, which is the server where Barman is installed, as well as all configured Postgres servers. This includes details such as global configuration, SSH version, Python version, rsync version, the current configuration and status of all servers, and many more.

Parameters

-h / --help

Show a help message and exit. Provides information about command usage.

--show-config-source

Include the source file which provides the effective value for each configuration option.

14.4.10 barman generate-manifest

Synopsis

```
generate-manifest
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

Generates a `backup_manifest` file for a backup. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.11 barman get-wal

Synopsis

```
get-wal
[ { --bzip | -j } ]
[ { --gzip | -z | -x } ]
[ { -h | --help } ]
[ --keep-compression ]
[ { --output-directory | -o } OUTPUT_DIRECTORY ]
[ { --peek | -p } VALUE ]
[ { -P | --partial } ]
[ { -t | --test } ]
SERVER_NAME WAL_NAME
```

Description

Retrieve a WAL file from the xlog archive of a specified server. By default, if the requested WAL file is found, it is returned as uncompressed content to STDOUT.

Parameters

SERVER_NAME

Name of the server in barman node

WAL_NAME

Id of the backup in barman catalog.

--bzip2 / -j

Output will be compressed using bzip2.

--gzip / -z / -x

Output will be compressed using gzip.

-h / --help

Show a help message and exit. Provides information about command usage.

--keep-compression

Do not uncompress the file content. The output will be the original compressed file.

--output-directory / -o

Destination directory where barman will store the WAL file.

--peek / -p

Specify an integer value greater than or equal to 1 to retrieve WAL files from the specified WAL file up to the value specified by this parameter. When using this option, `get-wal` returns a list of zero to the specified WAL segment names, with one name per row.

-P / --partial

Additionally, collect partial WAL files (.partial).

-t / --test

Test both the connection and configuration of the specified Postgres server in Barman for WAL retrieval. When this option is used, the required WAL_NAME argument is disregarded.

Warning

-z / --gzip and **-j / --bzip2** options are deprecated and will be removed in the future. For WAL compression, please make sure to enable it directly on the Barman server via the `compression` configuration option.

14.4.12 barman keep

Synopsis

```
keep
[ { -h | --help } ]
{ { -r | --release } | { -s | --status } | --target { full | standalone } }
SERVER_NAME BACKUP_ID
```

Description

Mark the specified backup with a `target` as an archival backup to be retained indefinitely, overriding any active retention policies. You can also check the keep status of a backup and release the keep mark from a backup. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

-r / --release

Release the keep mark from this backup. This will remove its archival status and make it available for deletion, either directly or by retention policy.

-s / --status

Report the archival status of the backup. The status will be either `full` or `standalone` for archival backups, or `nokeep` for backups that have not been designated as archival.

--target

Define the recovery target for the archival backup. The possible values are:

- `full`: The backup can be used to recover to the most recent point in time. To support this, Barman will keep all necessary WALs to maintain the backup's consistency as well as any subsequent WALs.
- `standalone`: The backup can only be used to restore the server to its state at the time of the backup. Barman will retain only the WALs required to ensure the backup's consistency.

14.4.13 barman list-backups

Synopsis

```
list-backups
[ { -h | --help } ]
[ --minimal ]
SERVER_NAME
```

Description

Display the available backups for a server. This command is useful for retrieving both the backup ID and the backup type. You can find details about this command in *Catalog usage*.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

--minimal

Machine readable output.

14.4.14 barman list-files

Synopsis

```
list-files
[ { -h | --help } ]
[ --target { data | full | standalone | wal } ]
[ --list-empty-directories ]
SERVER_NAME BACKUP_ID
```

Description

List all files in a specific backup. You can use a shortcut instead of BACKUP_ID.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

--target

Define specific files to be listed. The possible values are:

- standalone (default): List the base backup files, including required WAL files.
- data: List just the data files.

- **wal**: List all the WAL files between the start of the base backup and the end of the log or the start of the following base backup (depending on whether the specified base backup is the most recent one available).
- **full**: same as **data + wal**.

--list-empty-directories

Add empty directories to the listing.

14.4.15 barman list-processes**Synopsis**

```
list-processes
  [ { -h | --help } ]
  SERVER_NAME
```

Description

The **list-processes** sub-command outputs all active subprocesses for a Barman server. It displays the process identifier (PID) and the corresponding barman task for each active subprocess.

Parameters**SERVER_NAME**

Name of the server for which to list active subprocesses.

-h / --help

Displays a help message and exits.

14.4.16 barman list-servers**Synopsis**

```
list-servers
  [ { -h | --help } ]
  [ --minimal ]
```

Description

Display all configured servers along with their descriptions.

Parameters**-h / --help**

Show a help message and exit. Provides information about command usage.

--minimal

Machine readable output.

14.4.17 barman lock-directory-cleanup**Synopsis**

```
lock-directory-cleanup
  [ { -h | --help } ]
```

Description

Automatically removes unused lock files from the `barman_lock_directory`.

Parameters

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.18 `barman put-wal`

Synopsis

```
put-wal
[ { -h | --help } ]
[ { -t | --test } ]
SERVER_NAME
```

Description

Receive a WAL file from a remote server and securely save it into the server incoming directory. The WAL file should be provided via STDIN, encapsulated in a tar stream along with a SHA256SUMS or MD5SUMS file for validation (`sha256` is the default hash algorithm, but the user can choose `md5` when setting the `archive-command` via `barman-wal-archive`). This command is intended to be executed via SSH from a remote `barman-wal-archive` utility (included in the `barman-cli` package). Avoid using this command directly unless you fully manage the content of the files.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

-t / --test

Test both the connection and configuration of the specified Postgres server in Barman for WAL retrieval.

14.4.19 `barman rebuild-xlogdb`

Synopsis

```
rebuild-xlogdb
[ { -h | --help } ]
SERVER_NAME [ SERVER_NAME ... ]
```

Description

Rebuild the WAL file metadata for a server (or for all servers using the `all` shortcut) based on the disk content. The WAL archive metadata is stored in the `xlog.db` file, with each Barman server maintaining its own copy.

Parameters

SERVER_NAME

Name of the server in barman node.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.20 **barman recover**

Note

This command is deprecated. Use the `commands-barman-restore` command instead.

14.4.21 **barman receive-wal**

Synopsis

```
receive-wal
[ --create-slot ]
[ --drop-slot ]
[ { -h | --help } ]
[ --reset ]
[ --stop ]
SERVER_NAME
```

Description

Initiate the streaming of transaction logs for a server. This process uses `pg_receivewal` or `pg_receivexlog` to receive WAL files from Postgres servers via the streaming protocol.

Parameters

SERVER_NAME

Name of the server in barman node.

--create-slot

Create the physical replication slot configured with the `slot_name` configuration parameter.

--drop-slot

Drop the physical replication slot configured with the `slot_name` configuration parameter.

-h / --help

Show a help message and exit. Provides information about command usage.

--reset

Reset the status of `receive-wal`, restarting the streaming from the current WAL file of the server.

--stop

Stop the process for the server.

Warning

The `--stop` option for the `barman receive-wal` command will be obsoleted in a future release. Users should favor using the `terminate-process` command instead, which is the new way of handling this feature.

14.4.22 barman restore

Synopsis

```
restore
[ --aws-region AWS_REGION } ]
[ --azure-resource-group AZURE_RESOURCE_GRP ]
[ --bwlimit KBPS ]
[ --exclusive ]
[ --gcp-zone GCP_ZONE ]
[ { --get-wal | --no-get-wal } ]
[ { -h | --help } ]
[ { -j | --jobs } PARALLEL_WORKERS ]
[ --jobs-start-batch-period SECONDS ]
[ --jobs-start-batch-size NUMBER ]
[ --local-staging-path PATH ]
[ { --network-compression | --no-network-compression } ]
[ --no-retry ]
[ --recovery-conf-filename FILENAME ]
[ --recovery-staging-path PATH ]
[ --staging-path STAGING_PATH ]
[ --staging-location STAGING_LOCATION ]
[ --remote-ssh-command STRING ]
[ --retry-sleep SECONDS ]
[ --retry-times NUMBER ]
[ --snapshot-recovery-instance INSTANCE_NAME ]
[ --snapshot-recovery-zone GCP_ZONE ]
[ --standby-mode ]
[ --tablespace NAME:LOCATION [ --tablespace NAME:LOCATION ... ] ]
[ --target-action { pause | shutdown | promote } ]
[ --target-immediate ]
[ --target-lsn LSN ]
[ --target-name RESTORE_POINT_NAME ]
[ --target-time TIMESTAMP ]
[ --target-tli TLI ]
[ --target-xid XID ]
[ --staging-wal-directory ]
SERVER_NAME BACKUP_ID DESTINATION_DIR
```

Description

Execute a PostgreSQL server restore operation. Barman will restore the backup from a server in the destination directory. The restoration can be performed locally (on the barman node itself) or remotely (on another machine accessible via SSH). The location is determined by whether or not the `--remote-ssh-command` option is used. More information on this command can be found in the [Recovery](#) section. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in the barman catalog. Use `auto` to have Barman automatically find the most suitable backup for the restore operation.

DESTINATION_DIR

Destination directory to restore the backup.

--aws-region

Specify the AWS region where the instance and disks for snapshot recovery are located. This option allows you to override the `aws_region` value in the Barman configuration.

--azure-resource-group

Specify the Azure resource group containing the instance and disks for snapshot recovery. This option allows you to override the `azure_resource_group` value in the Barman configuration.

--bwlimit

Specify the maximum transfer rate in kilobytes per second. A value of 0 indicates no limit. This setting overrides the `bandwidth_limit` configuration option.

--exclusive

Set target (time, XID or LSN) to be non inclusive.

--gcp-zone

Specify the GCP zone where the instance and disks for snapshot recovery are located. This option allows you to override the `gcp_zone` value in the Barman configuration.

--get-wal / --no-get-wal

Enable/disable usage of `get-wal` for WAL fetching during recovery. Default is based on `recovery_options` setting.

-h / --help

Show a help message and exit. Provides information about command usage.

-j / --jobs

Specify the number of parallel workers to use for copying files during the backup. This setting overrides the `parallel_jobs` parameter if it is specified in the configuration file.

--jobs-start-batch-period

Specify the time period, in seconds, for starting a single batch of jobs. This value overrides the `parallel_jobs_start_batch_period` parameter if it is set in the configuration file. The default is 1 second.

--jobs-start-batch-size

Specify the maximum number of parallel workers to initiate in a single batch. This value overrides the `parallel_jobs_start_batch_size` parameter if it is defined in the configuration file. The default is 10 workers.

--local-staging-path

Specify path on the Barman host where the chain of backups will be combined before being copied to the destination directory. The contents created within the staging path will be removed upon completion of the restore process. This option is necessary for restoring from block-level incremental backups and has no effect otherwise.

--network-compression / --no-network-compression

Enable/disable network compression during remote restore. Default is based on `network_compression` configuration setting.

--no-retry

There will be no retry in case of an error. It is the same as setting `--retry-times 0`.

--recovery-conf-filename

Specify the name of the file where Barman should write recovery options when recovering backups for Postgres versions 12 and later. By default, this is set to `postgresql.auto.conf`. However, if `--recovery-conf-filename` is specified, recovery options will be written to the specified value instead. While the default value is suitable for most Postgres installations, this option allows you to specify an alternative location

if Postgres is managed by tools that alter the configuration mechanism (for example, if `postgresql.auto.conf` is symlinked to `/dev/null`).

--recovery-staging-path

Specify a path on the recovery host where files for a compressed backup will be staged before being uncompressed to the destination directory. Backups will be staged in their own directory within the staging path, following the naming convention: `barman-staging-SERVER_NAME-BACKUP_ID`. This staging directory will be removed after the restore process is complete. This option is mandatory for restoring from compressed backups and has no effect otherwise.

--staging-path

A path where intermediate files are staged during restore. When restoring a compressed backup, it serves as a temporary location for decompression before copying to the final destination. When restoring an incremental backup, it is where backups are combined before copying to the final destination. This location must have enough space to store the decompressed/combined backup.

--staging-location

Specifies whether `--staging-path` is a local or remote path. Valid values are `local` and `remote`.

--remote-ssh-command

This option enables remote restore by specifying the secure shell command to execute on a remote host. It functions similarly to the `ssh_command` server option in the configuration file for remote restore, that is, `'ssh USER@SERVER'`.

--retry-sleep

Specify the number of seconds to wait after a failed copy before retrying. This setting applies to both backup and restore operations and overrides the `basebackup_retry_sleep` parameter if it is defined in the configuration file.

--retry-times

Specify the number of times to retry the base backup copy in case of an error. This applies to both backup and restore operations and overrides the `basebackup_retry_times` parameter if it is set in the configuration file.

--snapshot-recovery-instance

Specify the name of the instance where the disks recovered from the snapshots are attached. This option is necessary when recovering backups created with `backup_method=snapshot`.

--snapshot-recovery-zone (deprecated)

Zone containing the instance and disks for the snapshot recovery (deprecated: replaced by `--gcp-zone`)

--standby-mode

Whether to start the Postgres server as a standby.

--tablespace

Specify tablespace relocation rule. `NAME` is the tablespace name and `LOCATION` is the recovery host destination path to restore the tablespace.

--target-action

Trigger the specified action when the recovery target is reached. This option requires defining a target along with one of these actions. The possible values are:

- **pause**: Once recovery target is reached, the server is started in pause state, allowing users to inspect the instance
- **promote**: Once recovery target is reached, the server will exit the recovery operation and is promoted as a master.
- **shutdown**: Once recovery target is reached, the server is shut down.

--target-immediate

Recovery is completed when a consistent state is reached (end of the base backup).

--target-lsn

Recover to the specified LSN (Log Sequence Number). Requires Postgres 10 or above.

--target-name

Recover to the specified name of a restore point previously created with the `pg_create_restore_point(name)`.

--target-time

Recover to the specified time. Use the format `YYYY-MM-DD HH:MM:SS.mmm`.

--target-tli

Recover the specified timeline. You can use the special values `current` and `latest` in addition to a numeric timeline ID. For Postgres versions 12 and above, the default is to recover to the latest timeline in the WAL archive. For Postgres versions below 12, the default is to recover to the timeline that was current at the time the backup was taken.

--target-xid

Recover to the specified transaction ID.

--staging-wal-directory

A staging directory on the destination host for WAL files when performing PITR. If unspecified, it uses a `barman_wal` directory inside the destination directory.

14.4.23 barman replication-status

Synopsis

```
replication-status
[ { -h | --help } ]
[ --minimal ]
[ --source { backup-host | wal-host } ]
[ --target { hot-standby | wal-streamer | all } ]
SERVER_NAME [ SERVER_NAME ... ]
```

Description

Display real-time information and status of any streaming clients connected to the specified server. Specify `all` shortcut to display information for all configured servers.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

--minimal

Machine readable output.

--source

The possible values are:

- `backup-host` (default): List clients using the backup connection information for a server.
- `wal-host`: List clients using the WAL streaming connection information for a server.

--target

The possible values are:

- `hot-standby`: List only hot standby servers.
- `wal-streamer`: List only WAL streaming clients, such as `pg_receivewal`.
- `all` (default): List all streaming clients.

14.4.24 `barman show-backup`

Synopsis

```
show-backup
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

Display detailed information about a specific backup. You can find details about this command in *Catalog usage*. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.25 `barman show-servers`

Synopsis

```
show-servers
[ { -h | --help } ]
SERVER_NAME [ SERVER_NAME ... ]
```

Description

Display detailed information about a server, including `conninfo`, `backup_directory`, `wals_directory`, `archive_command`, and many more. To view information about all configured servers, specify the `all` shortcut instead of the server name.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.26 barman status

Synopsis

```
status
  [ { -h | --help } ]
  SERVER_NAME [ SERVER_NAME ... ]
```

Description

Display information about a server's status, including details such as the state, Postgres version, WAL information, available backups and more.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.27 barman switch-wal

Synopsis

```
switch-wal
  [ --archive ]
  [ --archive-timeout ]
  [ --force ]
  [ { -h | --help } ]
  SERVER_NAME [ SERVER_NAME ... ]
```

Description

Execute `pg_switch_wal()` on the target server (Postgres versions 10 and later) or `pg_switch_xlog()` (for Postgres versions 8.3 to 9.6).

Parameters

SERVER_NAME

Name of the server in barman node

--archive

Waits for one WAL file to be archived. If no WAL file is archived within a specified time (default: 30 seconds), Barman will terminate with a failure exit code. This option is also available on standby servers.

--archive-timeout

Specify the amount of time in seconds (default: 30 seconds) that the archiver will wait for a new WAL file to be archived before timing out. This option is also available on standby servers.

--force

Forces the switch by executing a CHECKPOINT before `pg_switch_wal()`.

Note

Running a CHECKPOINT may increase I/O load on the Postgres server, so use this option cautiously.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.28 **barman switch-xlog**

Description

Alias for the `switch-wal` command.

14.4.29 **barman sync-backup**

Synopsis

```
sync-backup
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

This command synchronizes a passive node with its primary by copying all files from a backup present on the server node. It is available only for passive nodes and uses the `primary_ssh_command` option to establish a secure connection with the primary node. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.30 **barman sync-info**

Synopsis

```
sync-info
[ { -h | --help } ]
[ --primary ]
SERVER_NAME [ LAST_WAL [ LAST_POS ] ]
```

Description

Gather information about the current status of a Barman server for synchronization purposes.

This command returns a JSON output for a server that includes: all successfully completed backups, all archived WAL files, the configuration, the last WAL file read from `xlog.db`, and its position within the file.

Parameters

SERVER_NAME

Name of the server in barman node

LAST_WAL

Instructs sync-info to skip any WAL files that precede the specified file (for incremental synchronization).

LAST_POS

Hint for quickly positioning in the `xlog.db` file (for incremental synchronization).

-h / --help

Show a help message and exit. Provides information about command usage.

--primary

Execute the sync-info on the primary node (if set).

14.4.31 barman sync-wals

Synopsis

```
sync-wals
  [ { -h | --help } ]
  SERVER_NAME
```

Description

This command synchronizes a passive node with its primary by copying all archived WAL files from the server node. It is available only for passive nodes and utilizes the `primary_ssh_command` option to establish a secure connection with the primary node.

Parameters

SERVER_NAME

Name of the server in barman node

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.32 barman terminate-process

Synopsis

```
terminate-process SERVER_NAME TASK_NAME
```

Description

The `barman terminate-process` command terminates an active Barman subprocess on a specified server. The target process is identified by its task name (for example, `backup` or `receive-wal`). Note that only processes that are running on the server level can be terminated, so global processes like `cron` or `config-update` can not be terminated by this command.

You can also use the output of `barman list-processes` to display all active processes for a given server and determine which tasks can be terminated. More details about this command can be found in `barman list-processes`.

Parameters

SERVER_NAME

The name of the server where the subprocess is running.

TASK_NAME

The task name that identifies the subprocess to be terminated.

14.4.33 **barman verify-backup**

Synopsis

```
verify-backup
[ { -h | --help } ]
SERVER_NAME BACKUP_ID
```

Description

Runs `pg_verifybackup` on a backup manifest file (available since Postgres version 13). For rsync backups, it can be used after creating a manifest file using the `generate-manifest` command. Requires `pg_verifybackup` to be installed on the backup server. You can use a shortcut instead of `BACKUP_ID`.

Parameters

SERVER_NAME

Name of the server in barman node

BACKUP_ID

Id of the backup in barman catalog.

-h / --help

Show a help message and exit. Provides information about command usage.

14.4.34 **barman verify**

Description

Alias for `verify-backup` command.

14.5 **barman-cli commands**

The `barman-cli` package includes a collection of recommended client utilities that should be installed alongside the Postgres server. Here are the command references for both utilities.

14.5.1 **barman-wal-archive**

Synopsis

```
barman-wal-archive
[ { -h | --help } ]
[ { -V | --version } ]
[ { -U | --user } USER ]
[ --port PORT ]
[ { { -z | --gzip } | { -j | --bzip2 } | --xz | --snappy | --zstd | --lz4 } ]
```

(continues on next page)

(continued from previous page)

```
[ --compression-level COMPRESSION_LEVEL ]
[ { -c | --config } CONFIG ]
[ { -t | --test } ]
[ --md5 ]
BARMAN_HOST SERVER_NAME WAL_PATH
```

Description

This script can be utilized in the `archive_command` of a Postgres server to transfer WAL files to a Barman host using the `put-wal` command (introduced in Barman 2.6). It establishes an SSH connection to the Barman host, enabling seamless integration of Barman within Postgres clusters for improved business continuity.

Exit Statuses are:

- 0 for SUCCESS.
- non-zero for FAILURE.

Parameters

SERVER_NAME

The server name configured in Barman for the Postgres server from which the WAL file is retrieved.

BARMAN_HOST

The host of the Barman server.

WAL_PATH

The value of the '%p' keyword (according to `archive_command`).

-h / --help

Display a help message and exit.

-V / --version

Display the program's version number and exit.

-U / --user

Specify the user for the SSH connection to the Barman server (defaults to `barman`).

--port

Define the port used for the SSH connection to the Barman server.

-z / --gzip

gzip-compress the WAL file before sending it to the Barman server.

-j / --bzip2

bzip2-compress the WAL file before sending it to the Barman server.

--xz

xz-compress the WAL file before sending it to the Barman server.

--snappy

snappy-compress the WAL file before sending it to the Barman server (requires the `python-snappy` Python library to be installed).

--zstd

zstd-compress the WAL file before sending it to the Barman server (requires the `zstandard` Python library to be installed).

--lz4

lz4-compress the WAL file before sending it to the Barman server (requires the `lz4` Python library to be installed).

--compression-level

A compression level to be used by the selected compression algorithm. Valid values are integers within the supported range of the chosen algorithm or one of the predefined labels: `low`, `medium`, and `high`. The range of each algorithm as well as what level each predefined label maps to can be found in *compression_level*.

-c / --config

Specify the configuration file on the Barman server.

-t / --test

Test the connection and configuration of the specified Postgres server in Barman to ensure it is ready to receive WAL files. This option ignores the mandatory argument `WAL_PATH`.

--md5

Use MD5 instead of SHA256 as the hash algorithm to calculate the checksum of the WAL file when transmitting it to the Barman server. This is used to maintain compatibility with older server versions, as older versions of Barman server used to support only MD5.

Note

When compression is enabled in `barman-wal-archive`, it takes precedence over the compression settings configured on the Barman server, if they differ.

Important

When compression is enabled in `barman-wal-archive`, it is performed on the client side, before the file is sent to Barman. Be mindful of the database server's load and the chosen compression algorithm and level, as higher compression can delay WAL shipping, causing WAL files to accumulate.

14.5.2 barman-wal-restore

Synopsis

```
barman-wal-restore
[ { -h | --help } ]
[ { -V | --version } ]
[ { -U | --user } USER ]
[ --port PORT ]
[ { -s | --sleep } SECONDS ]
[ { -p | --parallel } JOBS ]
[ --spool-dir SPOOL_DIR ]
[ { -P | --partial } ]
[ { { -z | --gzip } | { -j | --bzip2 } | --keep-compression } ]
[ { -c | --config } CONFIG ]
[ { -t | --test } ]
BARMAN_HOST SERVER_NAME WAL_NAME WAL_DEST
```

Description

This script serves as a `restore_command` for Postgres servers, enabling the retrieval of WAL files through Barman's `get-wal` feature. It establishes an SSH connection to the Barman host and facilitates the integration of Barman within Postgres clusters, enhancing business continuity.

Exit Statuses are:

- 0 for SUCCESS.
- 1 for remote get-wal command FAILURE, likely because the requested WAL could not be found.
- 2 for ssh connection FAILURE.
- Any other non-zero for FAILURE.

Parameters

SERVER_NAME

The server name configured in Barman for the Postgres server from which the WAL file is retrieved.

BARMAN_HOST

The host of the Barman server.

WAL_NAME

The value of the '%f' keyword (according to `restore_command`).

WAL_DEST

The value of the '%p' keyword (according to `restore_command`).

-h / --help

Display a help message and exit.

-V / --version

Display the program's version number and exit.

-U / --user

Specify the user for the SSH connection to the Barman server (defaults to `barman`).

--port

Define the port used for the SSH connection to the Barman server.

-s / --sleep

Pause for SECONDS after a failed `get-wal` request (defaults to 0 - no wait).

-p / --parallel

Indicate the number of files to peek and transfer simultaneously (defaults to 0 - disabled).

--spool-dir

Specify the spool directory for WAL files (defaults to `/var/tmp/walrestore`).

-P / --partial

Include partial WAL files (.partial) in the retrieval.

-z / --gzip

Transfer WAL files compressed with `gzip`.

-j / --bzip2

Transfer WAL files compressed with `bzip2`.

--keep-compression

If specified, compressed files will be transferred as-is and decompressed on arrival on the client-side.

-c / --config

Specify the configuration file on the Barman server.

-t / --test

Test the connection and configuration of the specified Postgres server in Barman to ensure it is ready to receive WAL files. This option ignores the mandatory arguments `WAL_NAME` and `WAL_DEST`.

 Warning

-z / --gzip and -j / --bzip2 options are deprecated and will be removed in the future. For WAL compression, please make sure to enable it directly on the Barman server via the `compression` configuration option.

Geographical Redundancy

It's possible to set up cascading backup architectures with Barman, where the source of a Barman backup server is another Barman installation rather than a PostgreSQL server.

This feature allows users to transparently keep geographically distributed copies of PostgreSQL backups.

In Barman jargon, a Barman backup server that is connected to another Barman installation, rather than a PostgreSQL server, is defined as a passive node. A passive node is configured through the `primary_ssh_command` option, available both at global (for a full replica of a primary Barman installation) and server level (for mixed scenarios, having both direct and passive servers).

15.1 Sync information

The `barman sync-info` command is used to collect information regarding the current status of a Barman server which is useful for synchronization purposes. The available syntax is the following:

```
barman sync-info [--primary] <server_name> [<last_wal> [<last_position>]]
```

The command returns a JSON object containing:

- A map with all the backups having status `DONE` for that server.
- A list with all the archived WAL files.
- The configuration for the server.
- The last read position (in the xlog database file).
- The name of the last read WAL file.

The JSON response contains all the required information for the synchronisation between the `primary` and a `passive` node.

If `--primary` is specified, the command is executed on the defined primary node, rather than locally.

15.2 Configuration

Configuring a server as a passive node is a quick operation. Simply add to the server configuration the following option:

```
primary_ssh_command = ssh barman@primary_barman
```

This option specifies the SSH connection parameters to the primary server, identifying the source of the backup data for the passive server.

If you are invoking barman with the `-c/--config` option and you want to use the same option when the passive node invokes barman on the primary node then add the following option:

```
forward_config_path = true
```

15.3 Node synchronization

When a node is marked as passive it is treated in a special way by Barman:

- It is excluded from standard maintenance operations.
- Direct operations to Postgres are forbidden, including `barman backup`.

Synchronization between a passive server and its primary is automatically managed by `barman cron` which will transparently invoke:

1. `barman sync-info --primary`, in order to collect synchronization information.
2. `barman sync-backup`, in order to create a local copy of every backup that is available on the primary node.
3. `barman sync-wals`, in order to copy locally all the WAL files available on the primary node.

15.4 Manual synchronization

Although `barman cron` automatically manages passive/primary node synchronization, it is possible to manually trigger synchronization of a backup through:

```
barman sync-backup <server_name> <backup_id>
```

Launching `sync-backup` barman will use the `primary_ssh_command` to connect to the primary server, then, if the backup is present on the remote machine, it will begin to copy all the files using `rsync`. Only one synchronization process per backup is allowed at a time.

WAL files can also be synchronized, through:

```
barman sync-wals <server_name>
```

Hook Scripts

Barman allows *DBAs* to run hook scripts during specific events:

- *Before and after creating a backup.*
- *Before and after deleting a backup.*
- *Before and after a WAL file is archived.*
- *Before and after a WAL file is deleted.*
- *Before and after restoring a backup.*

Important

These scripts can be configured using the global options, which can be overridden for individual servers. Deletion and recovery hook scripts were introduced in **version 2.4**.

There are two types of hook scripts that Barman can manage:

- **Standard Hook Scripts:** These scripts are executed once and do not have their return codes checked.
- **Retry Hook Scripts:** These scripts might be executed multiple times, depending on their return codes.

When executing a retry hook script, Barman checks the return code and will continue to retry the script until it returns one of the following:

- SUCCESS (standard return code 0)
- ABORT_CONTINUE (return code 62)
- ABORT_STOP (return code 63)

Any other return code is treated as a transient failure, prompting Barman to retry the script. This gives users greater control, allowing hook scripts to determine if a failure should be considered transient. Additionally, for a pre-hook script, returning ABORT_STOP allows users to request that Barman interrupts the main operation with a failure.

Hook scripts are executed in the following sequence (skipped if not present):

1. **Standard pre hook script.**
2. **Retry pre hook script.**
3. **Main event** (backup operation, restore operation or WAL archiving), unless the retry pre hook script was aborted with `ABORT_STOP`.
4. **Retry post hook script.**
5. **Standard post hook script.**

All output generated by hook scripts is logged in Barman's log file..

Note

Currently, `ABORT_STOP` is ignored by retry post-hook scripts. In such cases, this will result in logging an additional warning, and `ABORT_STOP` will behave the same as `ABORT_CONTINUE`.

16.1 Before and after creating a backup

- `pre_backup_script`: A hook script executed once before a base backup, without checking the exit code.
- `pre_backup_retry_script`: A retry hook script executed before a base backup, running repeatedly until it succeeds or is aborted.
- `post_backup_retry_script`: A retry hook script executed after a base backup, running repeatedly until it succeeds or is aborted.
- `post_backup_script`: A hook script executed once after a base backup, without checking the exit code.

The shell environment will include the following variables for **backup scripts**:

- `BARMAN_BACKUP_DIR`: The destination directory for the backup.
- `BARMAN_BACKUP_ID`: The ID of the backup.
- `BARMAN_BACKUP_INFO_PATH`: The path of the *backup.info* file.
- `BARMAN_CONFIGURATION`: The configuration file used by Barman.
- `BARMAN_ERROR`: Any error message (only applicable in the **post** phase).
- `BARMAN_PHASE`: Indicates the phase of the script, either **pre** or **post**.
- `BARMAN_PREVIOUS_ID`: The ID of the previous backup, if available.
- `BARMAN_RETRY`: Set to 1 if it is a retry script; otherwise, it is 0.
- `BARMAN_SERVER`: The name of the server.
- `BARMAN_STATUS`: The status of the backup.
- `BARMAN_VERSION`: The version of Barman.

16.2 Before and after deleting a backup

- `pre_delete_script`: A hook script triggered once before the backup deletion, with no exit code check.
- `pre_delete_retry_script`: A retry hook script that runs before the backup deletion, executing repeatedly until it succeeds or is aborted.

- **post_delete_retry_script**: A retry hook script that runs after the backup deletion, executing repeatedly until it succeeds or is aborted.
- **post_delete_script**: A hook script triggered once after the backup deletion, with no exit code check.

Delete scripts utilize the same environmental variables as **backup scripts**, plus:

BARMAN_NEXT_ID: The ID of the next backup, if available.

16.3 Before and after a WAL is archived

- **pre_archive_script**: A hook script that runs once before a WAL file is archived by maintenance (typically via Barman cron), without checking the exit code.
- **pre_archive_retry_script**: A retry hook script that executes before a WAL file is archived by maintenance, running repeatedly until it succeeds or is aborted.
- **post_archive_retry_script**: A retry hook script that executes after a WAL file is archived by maintenance, running repeatedly until it succeeds or is aborted.
- **post_archive_script**: A hook script that runs once after a WAL file is archived by maintenance, without checking the exit code.

WAL archive scripts share several environmental variables with **backup scripts**:

- **BARMAN_CONFIGURATION**: The configuration file used by Barman.
- **BARMAN_ERROR**: Any error message generated (only for the post phase).
- **BARMAN_PHASE**: The phase of the script, either pre or post.
- **BARMAN_SERVER**: The name of the server.

Additionally, the following variables are specific to **WAL archive scripts**:

- **BARMAN_SEGMENT**: The name of the WAL file.
- **BARMAN_FILE**: The full path of the WAL file.
- **BARMAN_SIZE**: The size of the WAL file.
- **BARMAN_TIMESTAMP**: The timestamp of the WAL file.
- **BARMAN_COMPRESSION**: The type of compression applied to the WAL file.

16.4 Before and after a WAL file is deleted

- **pre_wal_delete_script**: A hook script that runs before a WAL file is deleted.
- **pre_wal_delete_retry_script**: A retry hook script that executes before the deletion of a WAL file, running repeatedly until it succeeds or is aborted.
- **post_wal_delete_retry_script**: A retry hook script that runs after a WAL file is deleted, executing repeatedly until it succeeds or is aborted.
- **post_wal_delete_script**: A hook script that runs after a WAL file is deleted.

WAL delete scripts utilize the same environmental variables as **WAL archive scripts**.

16.5 Before and after restoring a backup

- `pre_recovery_script`: A hook script that runs once before a backup restore, without checking the exit code.
- `pre_recovery_retry_script`: A retry hook script that executes before a backup restore, running repeatedly until it succeeds or is aborted.
- `post_recovery_retry_script`: A retry hook script that runs after a backup restore, executing repeatedly until it succeeds or is aborted.
- `post_recovery_script`: A hook script that runs once after a backup restore, without checking the exit code.

Recovery scripts utilize the same environmental variables as **backup scripts**, plus:

- `BARMAN_DESTINATION_DIRECTORY`: The directory where the new instance is restored.
- `BARMAN_TABLESPACES`: The tablespace relocation map (in JSON format, if applicable).
- `BARMAN_REMOTE_COMMAND`: The secure shell command used during restore (if applicable).
- `BARMAN_RECOVER_OPTIONS`: Additional recovery options (in JSON format, if applicable).

16.6 Using `barman-cloud-*` scripts as hooks in barman

Follow the process in the *installation section* to start using the *barman-cloud commands*.

Note

For detailed information on configuration options, refer to the *configuration section*.

You can use `barman-cloud-backup` as a `post-backup script` for the following Barman backup types:

- Backups created with `backup_method = rsync`.
- Backups created with `backup_method = postgres` when `backup_compression` is not applied.

To configure this, add the following line to your server configuration in Barman:

```
post_backup_retry_script = barman-cloud-backup [ OPTIONS ] DESTINATION_URL SERVER_NAME
```

Warning

When used as a hook script, `barman-cloud-backup` requires the backup status to be `DONE`. It will fail if the backup has any other status. To avoid issues, it is recommended to run backups with the `-w / --wait` option to ensure the hook script is not executed while the backup status is `WAITING_FOR_WALS`.

Additionally, set up `barman-cloud-wal-archive` as a pre-WAL archive script by adding the following line to the Barman configuration for your Postgres server:

```
pre_archive_retry_script = barman-cloud-wal-archive [ OPTIONS ] DESTINATION_URL SERVER_
↪NAME
```

Barman for the cloud

Barman offers two primary methods for backing up Postgres servers to the cloud:

- **Creating disk volume snapshots as base backups.**

This can be achieved through 2 different approaches:

1. Setting up a Barman server to store the backup metadata and the WAL files, while your backups are created as disk volume snapshots in the cloud. This is an integrated feature of Barman. If you choose this approach, please consult the *cloud snapshots backups* section for details.
2. Interacting and managing backups directly with the command line utility provided by the barman cloud client package without the need of a Barman server. The backup metadata and the WAL files are stored in the cloud object storage, while your base backup is created as disk volume snapshots in the cloud.

- **Creating and transferring base backups to a cloud object storage.**

This can also be achieved through 2 different approaches:

1. Using the utility provided by the barman cloud client package in the Postgres host, without a Barman server. Both the base backup and the WALs are read from the local host (Postgres host), and they are stored along with the backup metadata in the cloud object storage.
2. Setting up a Barman server to take base backups and store the backup metadata and the WAL files, then use the utility provided by the barman cloud client package as hook scripts to copy them to the cloud object storage. If you choose this approach, please consult the *Using barman-cloud-* scripts as hooks in barman* section for details.

This section of the documentation is focused in the `barman-cloud-*` commands that can be used to manage and interact with backups without the need of a dedicated barman server. To start working with it, you will need to install the barman cloud client package on the same machine as your Postgres server.

Understanding these options will help you select the right approach for your cloud backup and recovery needs, ensuring you leverage Barman's full potential.

17.1 Barman cloud client package

The barman cloud client package provides commands for managing cloud backups, both in object storage and as disk volume snapshots, without requiring a Barman server.

With this utility, you can:

- Create and manage snapshot backups directly.
- Create and transfer backups to cloud object storage.

While it offers additional functionality for handling backups in cloud storage and disk volumes independently, it does not fully extend Barman's native capabilities. It has limitations compared to the integrated features of Barman and some operations may differ.

Note

Barman supports AWS S3 (and S3 compatible object stores), Azure Blob Storage and Google Cloud Storage.

17.2 Installation

To back up Postgres servers directly to a cloud provider, you need to install the *barman cloud client package* on those servers. Keep in mind that the installation process varies based on the distribution you are using.

Refer to the [installation](#) section for the installation process, and make sure to note the important information for each distribution.

17.3 Commands Reference

You have several commands available to manage backup and recovery in the cloud using this utility. The exit statuses for them are SUCCESS (0), FAILURE (1), FAILED CONNECTION (2) and INPUT_ERROR (3). Any other non-zero is FAILURE.

Note

When running Barman cloud commands, it is possible to specify some backup *shortcuts* instead of a backup ID. The cloud commands support the following shortcuts: *first/oldest*, *last/latest* and *last-failed*.

17.3.1 barman-cloud-backup

Synopsis

```
barman-cloud-backup
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage } ]
]

    [ { { -z | --gzip } | { -j | --bzip2 } | --snappy } ]
    [ { -h | --host } HOST ]
    [ { -p | --port } PORT ]
    [ { -U | --user } USER ]
```

(continues on next page)

(continued from previous page)

```

[ { -d | --dbname } DBNAME ]
[ { -n | --name } BACKUP_NAME ]
[ { -J | --jobs } JOBS ]
[ { -S | --max-archive-size } MAX_ARCHIVE_SIZE ]
[ --immediate-checkpoint ]
[ --min-chunk-size MIN_CHUNK_SIZE ]
[ --max-bandwidth MAX_BANDWIDTH ]
[ --snapshot-instance SNAPSHOT_INSTANCE ]
[ --snapshot-disk NAME ]
[ --snapshot-zone GCP_ZONE ]
[ --snapshot-gcp-project GCP_PROJECT ]
[ --tags TAG [ TAG ... ] ]
[ --endpoint-url ENDPOINT_URL ]
[ { -P | --aws-profile } AWS_PROFILE ]
[ --profile AWS_PROFILE ]
[ --read-timeout READ_TIMEOUT ]
[ { -e | --encryption } { AES256 | aws:kms } ]
[ --sse-kms-key-id SSE_KMS_KEY_ID ]
[ --aws-region AWS_REGION ]
[ --aws-await-snapshots-timeout AWS_AWAIT_SNAPSHOTS_TIMEOUT ]
[ --aws-snapshot-lock-mode { compliance | governance } ]
[ --aws-snapshot-lock-duration DAYS ]
[ --aws-snapshot-lock-cool-off-period HOURS ]
[ --aws-snapshot-lock-expiration-date DATETIME ]
[ { --azure-credential | --credential } { azure-cli | managed-identity |
↵default } ]
[ --encryption-scope ENCRYPTION_SCOPE ]
[ --azure-subscription-id AZURE_SUBSCRIPTION_ID ]
[ --azure-resource-group AZURE_RESOURCE_GROUP ]
[ --gcp-project GCP_PROJECT ]
[ --kms-key-name KMS_KEY_NAME ]
[ --gcp-zone GCP_ZONE ]
DESTINATION_URL SERVER_NAME

```

Description

The `barman-cloud-backup` script is used to create a local backup of a Postgres server and transfer it to a supported cloud provider, bypassing the Barman server. It can also be utilized as a hook script for copying Barman backups from the Barman server to one of the supported clouds (`post_backup_retry_script`).

This script requires read access to PGDATA and tablespaces, typically run as the postgres user. When used on a Barman server, it requires read access to the directory where Barman backups are stored. If `--snapshot-` arguments are used and snapshots are supported by the selected cloud provider, the backup will be performed using snapshots of the specified disks (`--snapshot-disk`). The backup label and metadata will also be uploaded to the cloud.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Important

The cloud upload may fail if any file larger than the configured `--max-archive-size` is present in the data directory or tablespaces. However, Postgres files up to 1GB are always allowed, regardless of the `--max-archive-size` setting.

Parameters

SERVER_NAME

Name of the server to be backed up.

DESTINATION_URL

URL of the cloud destination, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., `-vv` is more than `-v`).

-q / --quiet

Decrease output verbosity (e.g., `-qq` is less than `-q`).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options:

- `aws-s3`.
- `azure-blob-storage`.
- `google-cloud-storage`.

-z / --gzip

gzip-compress the backup while uploading to the cloud (should not be used with python < 3.2).

-j / --bzip2

bzip2-compress the backup while uploading to the cloud (should not be used with python < 3.3).

--snappy

snappy-compress the backup while uploading to the cloud (requires optional `python-snappy` library).

-h / --host

Host or Unix socket for Postgres connection (default: libpq settings).

-p / --port

Port for Postgres connection (default: libpq settings).

-U / --user

User name for Postgres connection (default: libpq settings).

-d / --dbname

Database name or conninfo string for Postgres connection (default: "postgres").

-n / --name

A name which can be used to reference this backup in commands such as `barman-cloud-restore` and `barman-cloud-backup-delete`.

- J / --jobs**
Number of subprocesses to upload data to cloud storage (default: 2).
- S / --max-archive-size**
Maximum size of an archive when uploading to cloud storage (default: 100GB).
- immediate-checkpoint**
Forces the initial checkpoint to be done as quickly as possible.
- min-chunk-size**
Minimum size of an individual chunk when uploading to cloud storage (default: 5MB for aws-s3, 64KB for azure-blob-storage, not applicable for google-cloud-storage).
- max-bandwidth**
The maximum amount of data to be uploaded per second when backing up to object storages (default: 0 - no limit).
- snapshot-instance**
Instance where the disks to be backed up as snapshots are attached.
- snapshot-disk**
Name of a disk from which snapshots should be taken.
- tags**
Tags to be added to all uploaded files in cloud storage, and/or to snapshots created, if snapshots are used.

Extra options for the AWS cloud provider

- endpoint-url**
Override default S3 endpoint URL with the given one.
- P / --aws-profile**
Profile name (e.g. INI section in AWS credentials file).
- profile (deprecated)**
Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.
- read-timeout**
The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).
- e / --encryption**
The encryption algorithm used when storing the uploaded data in S3.
Allowed options:
- AES256.
 - aws:kms.
- sse-kms-key-id**
The AWS KMS key ID that should be used for encrypting the uploaded data in S3. Can be specified using the key ID on its own or using the full ARN for the key. Only allowed if -e / --encryption is set to aws:kms.
- aws-region**
The name of the AWS region containing the EC2 VM and storage volumes defined by the --snapshot-instance and --snapshot-disk arguments.
- aws-await-snapshots-timeout**
The length of time in seconds to wait for snapshots to be created in AWS before timing out (default: 3600 seconds).
- aws-snapshot-lock-mode**
The lock mode for the snapshot. This is only valid if --snapshot-instance and --snapshot-disk are set.
Allowed options:

- compliance.
- governance.

--aws-snapshot-lock-duration

The lock duration is the period of time (in days) for which the snapshot is to remain locked, ranging from 1 to 36,500. Set either the lock duration or the expiration date (not both).

--aws-snapshot-lock-cool-off-period

The cooling-off period is an optional period of time (in hours) that you can specify when you lock a snapshot in compliance mode, ranging from 1 to 72.

--aws-snapshot-lock-expiration-date

The lock duration is determined by an expiration date in the future. It must be at least 1 day after the snapshot creation date and time, using the format YYYY-MM-DDTHH:MM:SS.sssZ. Set either the lock duration or the expiration date (not both).

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options:

- azure-cli.
- managed-identity.
- default.

--encryption-scope

The name of an encryption scope defined in the Azure Blob Storage service which is to be used to encrypt the data in Azure.

--azure-subscription-id

The ID of the Azure subscription which owns the instance and storage volumes defined by the --snapshot-instance and --snapshot-disk arguments.

--azure-resource-group

The name of the Azure resource group to which the compute instance and disks defined by the --snapshot-instance and --snapshot-disk arguments belong.

Extra options for GCP cloud provider**--gcp-project**

GCP project under which disk snapshots should be stored.

--snapshot-gcp-project (deprecated)

GCP project under which disk snapshots should be stored - replaced by --gcp-project.

--kms-key-name

The name of the GCP KMS key which should be used for encrypting the uploaded data in GCS.

--gcp-zone

Zone of the disks from which snapshots should be taken.

--snapshot-zone (deprecated)

Zone of the disks from which snapshots should be taken - replaced by --gcp-zone.

17.3.2 barman-cloud-backup-delete

Synopsis

```
barman-cloud-backup-delete
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]

    [ --endpoint-url ENDPOINT_URL ]
    [ { -r | --retention-policy } RETENTION_POLICY ]
    [ { -m | --minimum-redundancy } MINIMUM_REDUNDANCY ]
    [ { -b | --backup-id } BACKUP_ID ]
    [ --dry-run ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
↵default } ]
    [--batch-size DELETE_BATCH_SIZE]
    SOURCE_URL SERVER_NAME
```

Description

The `barman-cloud-backup-delete` script is used to delete one or more backups created with the `barman-cloud-backup` command from cloud storage and to remove the associated WAL files.

Backups can be specified for deletion either by their backup ID (as obtained from `barman-cloud-backup-list`) or by a retention policy. Retention policies mirror those used by the Barman server, deleting all backups that are not required to meet the specified policy. When a backup is deleted, any unused WAL files associated with that backup are also removed.

WALs are considered unused if:

- The WALs predate the `begin_wal` value of the oldest remaining backup.
- The WALs are not required by any archival backups stored in the cloud.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Important

Each backup deletion involves three separate requests to the cloud provider: one for the backup files, one for the `backup.info` file, and one for the associated WALs. Deleting by retention policy may result in a high volume of delete requests if a large number of backups are accumulated in cloud storage.

Parameters

SERVER_NAME

Name of the server that holds the backup to be deleted.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: s3://bucket/path/to/folder.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., -vv is more than -v).

-q / --quiet

Decrease output verbosity (e.g., -qq is less than -q).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- aws-s3.
- azure-blob-storage.
- google-cloud-storage.

-b / --backup-id

ID of the backup to be deleted. You can use a shortcut instead of the backup ID.

-m / --minimum-redundancy

The minimum number of backups that should always be available.

-r / --retention-policy

If specified, delete all backups eligible for deletion according to the supplied retention policy.

Syntax: REDUNDANCY value | RECOVERY WINDOW OF value { DAYS | WEEKS | MONTHS }

--batch-size

The maximum number of objects to be deleted in a single request to the cloud provider. If unset then the maximum allowed batch size for the specified cloud provider will be used (1000 for aws-s3, 256 for azure-blob-storage and 100 for google-cloud-storage).

--dry-run

Find the objects which need to be deleted but do not delete them.

Extra options for the AWS cloud provider

--endpoint-url

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider

--azure-credential / --credential

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- `azure-cli`.
- `managed-identity`.
- `default`.

17.3.3 barman-cloud-backup-show**Synopsis**

```
barman-cloud-backup-show
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]
    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
default } ]
    [ --format FORMAT ]
SOURCE_URL SERVER_NAME BACKUP_ID
```

Description

This script displays detailed information about a specific backup created with the `barman-cloud-backup` command. The output is similar to the `barman show-backup` from the `barman show-backup` command reference, but it has fewer information.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters**BACKUP_ID**

The ID of the backup. You can use a shortcut instead of the backup ID.

SERVER_NAME

Name of the server that holds the backup to be displayed.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., -vv is more than -v).

-q / --quiet

Decrease output verbosity (e.g., -qq is less than -q).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- aws-s3.
- azure-blob-storage.
- google-cloud-storage.

--format

Output format (console or json). Default console.

Extra options for the AWS cloud provider**--endpoint-url**

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- azure-cli.
- managed-identity.
- default.

17.3.4 barman-cloud-backup-list

Synopsis

```
barman-cloud-backup-list
    [ { -V | --version } ]
    [ --help ]
```

(continues on next page)

(continued from previous page)

```

]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }

    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
default } ]
    [ --format FORMAT ]
SOURCE_URL SERVER_NAME

```

Description

This script lists backups stored in the cloud that were created using the `barman-cloud-backup` command.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters**SERVER_NAME**

Name of the server that holds the backup to be listed.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., `-vv` is more than `-v`).

-q / --quiet

Decrease output verbosity (e.g., `-qq` is less than `-q`).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- `aws-s3`.
- `azure-blob-storage`.
- `google-cloud-storage`.

--format

Output format (console or json). Default console.

Extra options for the AWS cloud provider

--endpoint-url

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- azure-cli.
- managed-identity.
- default.

17.3.5 barman-cloud-backup-keep

Synopsis

```
barman-cloud-backup-keep
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]
    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
↪default } ]
    [ { { -r | --release } | { -s | --status } | --target { full |
↪standalone } } ]
    SOURCE_URL SERVER_NAME BACKUP_ID
```

Description

Use this script to designate backups in cloud storage as archival backups, ensuring their indefinite retention regardless of retention policies.

This script allows you to mark backups previously created with `barman-cloud-backup` as archival backups. Once flagged as archival, these backups are preserved indefinitely and are not subject to standard retention policies.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters**SERVER_NAME**

Name of the server that holds the backup to be kept.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

BACKUP_ID

The ID of the backup to be kept. You can use a shortcut instead of the backup ID.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., `-vv` is more than `-v`).

-q / --quiet

Decrease output verbosity (e.g., `-qq` is less than `-q`).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- `aws-s3`.
- `azure-blob-storage`.
- `google-cloud-storage`.

-r / --release

If specified, the command will remove the keep annotation and the backup will be eligible for deletion.

-s / --status

Print the keep status of the backup.

--target

Specify the recovery target for this backup. Allowed options are:

- `full`
- `standalone`

Extra options for the AWS cloud provider**--endpoint-url**

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by `--aws-profile`.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- `azure-cli`.
- `managed-identity`.
- `default`.

17.3.6 barman-cloud-check-wal-archive

Synopsis

```
barman-cloud-check-wal-archive
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]

    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential }
      { azure-cli | managed-identity | default } ]
    [ --timeline TIMELINE ]
    DESTINATION_URL SERVER_NAME
```

Description

Verify that the WAL archive destination for a server is suitable for use with a new Postgres cluster. By default, the check will succeed if the WAL archive is empty or if the target bucket is not found. Any other conditions will result in a failure.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters**SERVER_NAME**

Name of the server that needs to be checked.

DESTINATION_URL

URL of the cloud destination, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., -vv is more than -v).

-q / --quiet

Decrease output verbosity (e.g., -qq is less than -q).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- aws-s3.
- azure-blob-storage.
- google-cloud-storage.

--timeline

The earliest timeline whose WALs should cause the check to fail.

Extra options for the AWS cloud provider**--endpoint-url**

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- azure-cli.
- managed-identity.
- default.

17.3.7 barman-cloud-restore

Synopsis

```
barman-cloud-restore
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]
    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
↵default } ]
    [ --snapshot-recovery-instance SNAPSHOT_RECOVERY_INSTANCE ]
    [ --snapshot-recovery-zone GCP_ZONE ]
    [ --aws-region AWS_REGION ]
    [ --gcp-zone GCP_ZONE ]
    [ --azure-resource-group AZURE_RESOURCE_GROUP ]
    [ --tablespace NAME:LOCATION [ --tablespace NAME:LOCATION ... ] ]
    [ --target-lsn LSN ]
    [ --target-time TIMESTAMP ]
    [ --target-tli TLI ]
    SOURCE_URL SERVER_NAME BACKUP_ID RECOVERY_DESTINATION
```

Description

Use this script to restore a backup directly from cloud storage that was created with the `barman-cloud-backup` command. Additionally, this script can prepare for recovery from a snapshot backup by verifying that attached disks were cloned from the correct snapshots and by downloading the backup label from object storage.

This command does not automatically prepare Postgres for recovery. You must manually manage any *PITR* options, custom `restore_command` values, signal files, or required WAL files to ensure Postgres starts, either manually or using external tools.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters

SERVER_NAME

Name of the server that holds the backup to be restored.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

BACKUP_ID

The ID of the backup to be restored. You can use a shortcut instead of the backup ID. Besides that, you can use `auto` to have Barman automatically find the most suitable backup for the restore operation.

RECOVERY_DESTINATION

The path to a directory for recovery.

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., -vv is more than -v).

-q / --quiet

Decrease output verbosity (e.g., -qq is less than -q).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- aws-s3.
- azure-blob-storage.
- google-cloud-storage.

--snapshot-recovery-instance

Instance where the disks recovered from the snapshots are attached.

--tablespace

Tablespace relocation rule.

--target-lsn

The recovery target lsn, e.g., 3/64000000.

--target-time

The recovery target timestamp with or without timezone, in the format %Y-%m-%d %H:%M:%S.

--target-tli

The recovery target timeline.

Extra options for the AWS cloud provider**--endpoint-url**

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by --aws-profile.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

--aws-region

The name of the AWS region containing the EC2 VM and storage volumes defined by the --snapshot-instance and --snapshot-disk arguments.

Extra options for the Azure cloud provider**--azure-credential / --credential**

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used

for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- `azure-cli`.
- `managed-identity`.
- `default`.

--azure-resource-group

The name of the Azure resource group to which the compute instance and disks defined by the `--snapshot-instance` and `--snapshot-disk` arguments belong.

Extra options for GCP cloud provider

--gcp-zone

Zone of the disks from which snapshots should be taken.

--snapshot-recovery-zone (deprecated)

Zone containing the instance and disks for the snapshot recovery - replaced by `--gcp-zone`.

17.3.8 barman-cloud-wal-archive

Synopsis

```
barman-cloud-wal-archive
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage }
]
    [ { { -z | --gzip } | { -j | --bzip2 } | --xz | --snappy | --zstd | --
lz4 } ]
    [ --compression-level COMPRESSION_LEVEL ]
    [ --tags TAG [ TAG ... ] ]
    [ --history-tags HISTORY_TAG [ HISTORY_TAG ... ] ]
    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { -e | --encryption } ENCRYPTION ]
    [ --sse-kms-key-id SSE_KMS_KEY_ID ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity |
default } ]
    [ --encryption-scope ENCRYPTION_SCOPE ]
    [ --max-block-size MAX_BLOCK_SIZE ]
    [ --max-concurrency MAX_CONCURRENCY ]
    [ --max-single-put-size MAX_SINGLE_PUT_SIZE ]
    [ --kms-key-name KMS_KEY_NAME ]
    DESTINATION_URL SERVER_NAME [ WAL_PATH ]
```

Description

The `barman-cloud-wal-archive` command is designed to be used in the `archive_command` of a Postgres server to directly ship WAL files to cloud storage.

Note

If you are using Python 2 or unsupported versions of Python 3, avoid using the compression options `--gzip` or `--bzip2`. The script cannot restore gzip-compressed WALs on Python < 3.2 or bzip2-compressed WALs on Python < 3.3.

This script enables the direct transfer of WAL files to cloud storage, bypassing the Barman server. Additionally, it can be utilized as a hook script for WAL archiving (`pre_archive_retry_script`).

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters**SERVER_NAME**

Name of the server that will have the WALs archived.

DESTINATION_URL

URL of the cloud destination, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

WAL_PATH

The value of the '%p' keyword (according to `archive_command`).

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., `-vv` is more than `-v`).

-q / --quiet

Decrease output verbosity (e.g., `-qq` is less than `-q`).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- `aws-s3`.
- `azure-blob-storage`.
- `google-cloud-storage`.

-z / --gzip

gzip-compress the WAL while uploading to the cloud.

-j / --bzip2

bzip2-compress the WAL while uploading to the cloud.

--xz

xz-compress the WAL while uploading to the cloud.

--snappy

snappy-compress the WAL while uploading to the cloud (requires the python-snappy Python library to be installed).

--zstd

zstd-compress the WAL while uploading to the cloud (requires the zstandard Python library to be installed).

--lz4

lz4-compress the WAL while uploading to the cloud (requires the lz4 Python library to be installed).

--compression-level

A compression level to be used by the selected compression algorithm. Valid values are integers within the supported range of the chosen algorithm or one of the predefined labels: `low`, `medium`, and `high`. The range of each algorithm as well as what level each predefined label maps to can be found in [compression_level](#).

--tags

Tags to be added to archived WAL files in cloud storage.

--history-tags

Tags to be added to archived history files in cloud storage.

Extra options for the AWS cloud provider

--endpoint-url

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by `--aws-profile`.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

-e / --encryption

The encryption algorithm used when storing the uploaded data in S3.

Allowed options:

- `AES256`.
- `aws:kms`.

--sse-kms-key-id

The AWS KMS key ID that should be used for encrypting the uploaded data in S3. Can be specified using the key ID on its own or using the full ARN for the key. Only allowed if `-e / --encryption` is set to `aws:kms`.

Extra options for the Azure cloud provider

--azure-credential / --credential

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- `azure-cli`.
- `managed-identity`.
- `default`.

--encryption-scope

The name of an encryption scope defined in the Azure Blob Storage service which is to be used to encrypt the data in Azure.

--max-block-size

The chunk size to be used when uploading an object via the concurrent chunk method (default: 4MB).

--max-concurrency

The maximum number of chunks to be uploaded concurrently (default: 1).

--max-single-put-size

Maximum size for which the Azure client will upload an object in a single request (default: 64MB). If this is set lower than the Postgres WAL segment size after any applied compression then the concurrent chunk upload method for WAL archiving will be used.

Extra options for GCP cloud provider**--kms-key-name**

The name of the GCP KMS key which should be used for encrypting the uploaded data in GCS.

17.3.9 barman-cloud-wal-restore

Synopsis

```
barman-cloud-wal-restore
    [ { -V | --version } ]
    [ --help ]
    [ { { -v | --verbose } | { -q | --quiet } } ]
    [ { -t | --test } ]
    [ --cloud-provider { aws-s3 | azure-blob-storage | google-cloud-storage } ]
]
    [ --endpoint-url ENDPOINT_URL ]
    [ { -P | --aws-profile } AWS_PROFILE ]
    [ --profile AWS_PROFILE ]
    [ --read-timeout READ_TIMEOUT ]
    [ { --azure-credential | --credential } { azure-cli | managed-identity
    | default } ]
    [ --no-partial ]
    SOURCE_URL SERVER_NAME WAL_NAME WAL_DEST
```

Description

The `barman-cloud-wal-restore` script functions as the `restore_command` for retrieving WAL files from cloud storage and placing them directly into a Postgres standby server, bypassing the Barman server.

This script is used to download WAL files that were previously archived with the `barman-cloud-wal-archive` command. Disable automatic download of `.partial` files by calling `--no-partial` option.

Important

On the target Postgres node, when `pg_wal` and the spool directory are on the same filesystem, files are moved via renaming, which is faster than copying and deleting. This speeds up serving WAL files significantly. If the directories are on different filesystems, the process still involves copying and deleting, so there's no performance gain in that case.

Note

For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported.

Parameters

SERVER_NAME

Name of the server that will have WALs restored.

SOURCE_URL

URL of the cloud source, such as a bucket in AWS S3. For example: `s3://bucket/path/to/folder`.

WAL_NAME

The value of the '%f' keyword (according to `restore_command`).

WAL_DEST

The value of the '%p' keyword (according to `restore_command`).

-V / --version

Show version and exit.

--help

show this help message and exit.

-v / --verbose

Increase output verbosity (e.g., `-vv` is more than `-v`).

-q / --quiet

Decrease output verbosity (e.g., `-qq` is less than `-q`).

-t / --test

Test cloud connectivity and exit.

--cloud-provider

The cloud provider to use as a storage backend.

Allowed options are:

- `aws-s3`.
- `azure-blob-storage`.
- `google-cloud-storage`.

--no-partial

Do not download partial WAL files

Extra options for the AWS cloud provider

--endpoint-url

Override default S3 endpoint URL with the given one.

-P / --aws-profile

Profile name (e.g. INI section in AWS credentials file).

--profile (deprecated)

Profile name (e.g. INI section in AWS credentials file) - replaced by `--aws-profile`.

--read-timeout

The time in seconds until a timeout is raised when waiting to read from a connection (defaults to 60 seconds).

Extra options for the Azure cloud provider

--azure-credential / --credential

Optionally specify the type of credential to use when authenticating with Azure. If omitted then Azure Blob Storage credentials will be obtained from the environment and the default Azure authentication flow will be used for authenticating with all other Azure services. If no credentials can be found in the environment then the default Azure authentication flow will also be used for Azure Blob Storage.

Allowed options are:

- `azure-cli`.
- `managed-identity`.
- `default`.

AWS

Amazon Web Services

Barman

Backup and Recovery Manager.

DBA

Database Administrator.

DEB

Debian Package.

External Configuration Files

Files that are stored outside of PGDATA. For example, when you create a new PostgreSQL cluster using `pg_createcluster` (e.g., `sudo pg_createcluster 14 main`) in Ubuntu, it sets up a new data directory, typically under `/var/lib/postgresql/`, but the configuration files, like `postgresql.conf` and `pg_hba.conf`, are usually stored under `/etc/postgresql/<version>/main/` to separate them from the actual data.

GCP

Google Cloud Platform

GPG

GNU Privacy Guard

IAC

Infrastructure As Code

ICT

Information and Communication Technology.

libpq

The C application programmer's interface to Postgres. `libpq` is a set of library functions that allow client programs to pass queries to the Postgres backend server and to receive the results of these queries.

PGDATA

PostgreSQL data directory.

PGDG

Postgres Global Development Group.

PITR

Point-in-time Recovery.

RHEL

Red Hat Enterprise Linux.

RPM

Red Hat Package Manager.

RPO

Recovery Point Objective. The maximum targeted period in which data might be lost from an IT service due to a major incident. In summary, it represents the maximum amount of data you can afford to lose.

SLES

SUSE Linux Enterprise Server

SPOF

Single Point of Failure

VLDB

Very Large DataBase

WORM

Write Once, Read Many

A

AWS, [165](#)

B

Barman, [165](#)

D

DBA, [165](#)

DEB, [165](#)

E

External Configuration Files, [165](#)

G

GCP, [165](#)

GPG, [165](#)

I

IAC, [165](#)

ICT, [165](#)

L

libpq, [165](#)

P

PGDATA, [165](#)

PGDG, [166](#)

PITR, [166](#)

R

RHEL, [166](#)

RPM, [166](#)

RPO, [166](#)

S

SLES, [166](#)

SPOF, [166](#)

V

VLDB, [166](#)

W

WORM, [166](#)