Barman User Guide

Release 3.19.0

EnterpriseDB UK Limited

May 26, 2026

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 . . . 12
- 2.2 RHEL-based distributions . . . 12
- 2.3 Debian-based distributions . . . 13
- 2.4 SLES-based distributions . . . 14
- 2.5 Additional Requirements . . . 14

3 Quick start **17**
- 3.1 Configuring your first server . . . 17
- 3.2 Taking your first backup . . . 21
- 3.3 Restoring a backup . . . 22

4 Architectural backup designs with barman **23**
- 4.1 Where to install Barman . . . 23
- 4.2 One Barman, many Postgres servers . . . 25
- 4.3 Backup strategies . . . 25
- 4.4 WAL archiving strategies . . . 25
- 4.5 Two typical scenarios for backups . . . 26
- 4.6 Geographical redundancy . . . 30
- 4.7 Cloud snapshot backups . . . 31

5 Pre-requisites **33**
- 5.1 Database users . . . 33
- 5.2 Postgres connection . . . 34
- 5.3 Postgres client tools . . . 34
- 5.4 Postgres streaming replication connection . . . 35
- 5.5 SSH connections . . . 36
- 5.6 WAL archiving via `archive_command` . . . 37

6 Barman check **39**
- 6.1 Usage . . . 39

6.2 Understanding the output . . . 40

7 Backups **43**
7.1 Overview . . . 43
7.2 Streaming Backup . . . 44
7.3 Rsync Backups . . . 46
7.4 Local Rsync Backups . . . 47
7.5 Local-to-Cloud Backups . . . 47
7.6 General Backup Settings . . . 50

8 Cloud Snapshot Backups **57**
8.1 Requirements and Configuration . . . 58
8.2 Google Cloud Platform . . . 58
8.3 Microsoft Azure . . . 59
8.4 Amazon Web Services . . . 60
8.5 Backup Process . . . 62

9 WAL archiving **67**
9.1 WAL compression . . . 67
9.2 WAL encryption . . . 68
9.3 Synchronous WAL streaming . . . 69

10 Catalog information **71**
10.1 Purpose . . . 71
10.2 Usage . . . 72

11 Recovery **75**
11.1 Local recovery . . . 76
11.2 Remote Recovery . . . 76
11.3 Delta Recovery . . . 76
11.4 Tablespace Remapping . . . 77
11.5 Point-in-Time Recovery . . . 77
11.6 Fetching WALs from Barman . . . 78
11.7 Recovering Encrypted Backups . . . 81
11.8 Recovering Compressed Backups . . . 82
11.9 Recovering block-level incremental Backups . . . 82
11.10 Recovery Pipeline for multi-format backups . . . 83
11.11 Recovering Backups from the Cloud . . . 84
11.12 Limitations of .partial WAL files . . . 85
11.13 Managing external configuration files . . . 85
11.14 Recovering from Snapshot Backups . . . 86

12 Retention policies **89**
12.1 Overview . . . 89
12.2 Key Components of Retention Policies . . . 89
12.3 Key Objectives of Retention Policies . . . 90
12.4 Minimum redundancy safety . . . 90
12.5 Scope of retention policies . . . 91
12.6 Use cases . . . 92
12.7 How retention policies are enforced . . . 92
12.8 Configurations and Syntax . . . 92
12.9 Retention policy for block-level incremental backups . . . 92
12.10 Retention policy for Cloud Backups . . . 93

13 Backup export and import **95**
13.1 Overview . . . 95
13.2 Limitations . . . 95
13.3 Tarball layout . . . 96
13.4 Tarball filename and integrity . . . 96
13.5 Usage . . . 96

14 Diagnose and troubleshooting **99**
14.1 Barman status . . . 99
14.2 Barman show servers . . . 100
14.3 Barman replication status . . . 100
14.4 Barman diagnose . . . 100

15 Configuration Reference **101**
15.1 Usage . . . 101
15.2 Options . . . 102
15.3 Configuration Models . . . 126

16 Commands Reference **127**
16.1 `barman` . . . 127
16.2 Shortcuts . . . 129
16.3 Exit Statuses . . . 129
16.4 Sub-Commands . . . 129
16.5 `barman-cli` commands . . . 155

17 Geographical Redundancy **159**
17.1 Sync information . . . 159
17.2 Configuration . . . 160
17.3 Node synchronization . . . 160
17.4 Manual synchronization . . . 160

18 Hook Scripts **161**
18.1 Before and after creating a backup . . . 162
18.2 Before and after deleting a backup . . . 162
18.3 Before and after a WAL is archived . . . 163
18.4 Before and after a WAL file is deleted . . . 163
18.5 Before and after restoring a backup . . . 164
18.6 Using `barman-cloud-*` scripts as hooks in barman . . . 164

19 Barman for the cloud **167**
19.1 Barman cloud client package . . . 168
19.2 Installation . . . 169
19.3 Object Lock Support . . . 169
19.4 Commands Reference . . . 170

20 FAQ **197**
20.1 General . . . 197
20.2 Backup . . . 198
20.3 Installation & Configuration . . . 199
20.4 Recovery . . . 199
20.5 Requirements . . . 200

21 License **201**

22 Barman release notes **203**

22.1 3.19.0 (2026-05-20) . . . 203
22.2 3.18.0 (2026-03-12) . . . 206
22.3 3.17.0 (2026-01-07) . . . 209
22.4 3.16.2 (2025-11-04) . . . 212
22.5 3.16.1 (2025-10-14) . . . 212
22.6 3.16.0 (2025-10-02) . . . 212
22.7 3.15.0 (2025-08-05) . . . 214
22.8 3.14.1 (2025-06-18) . . . 218
22.9 3.14.0 (2025-05-15) . . . 219
22.10 3.13.3 (2025-04-24) . . . 221
22.11 3.13.2 (2025-03-27) . . . 222
22.12 3.13.1 (2025-03-20) . . . 222
22.13 3.13.0 (2025-02-20) . . . 223
22.14 3.12.1 (2024-12-09) . . . 226
22.15 3.12.0 (2024-11-21) . . . 226
22.16 3.11.1 (2024-08-22) . . . 229
22.17 3.11.0 (2024-08-22) . . . 229
22.18 3.10.1 (2024-06-12) . . . 232
22.19 3.10.0 (2024-01-24) . . . 232
22.20 3.9.0 (2023-10-03) . . . 232
22.21 3.8.0 (2023-08-31) . . . 233
22.22 3.7.0 (2023-07-25) . . . 233
22.23 3.6.0 (2023-06-15) . . . 234
22.24 3.5.0 (2023-03-29) . . . 234
22.25 3.4.0 (2023-01-26) . . . 235
22.26 3.3.0 (2022-12-14) . . . 235
22.27 3.2.0 (2022-10-20) . . . 236
22.28 3.1.0 (2022-09-14) . . . 236
22.29 3.0.1 (2022-06-27) . . . 237
22.30 3.0.0 (2022-06-23) . . . 237
22.31 2.19 (2022-03-09) . . . 238
22.32 2.18 (2022-01-21) . . . 238
22.33 2.17 (2021-12-01) . . . 239
22.34 2.16 (2021-11-17) . . . 239
22.35 2.15 (2021-10-12) . . . 239
22.36 2.14 (2021-09-22) . . . 240
22.37 2.13 (2021-07-26) . . . 240
22.38 2.12.1 (2021-06-30) . . . 241
22.39 2.12 (2020-11-05) . . . 241
22.40 2.11 (2020-07-09) . . . 241
22.41 2.10 (2019-12-05) . . . 242
22.42 2.9 (2019-08-01) . . . 243
22.43 2.8 (2019-05-17) . . . 243
22.44 2.7 (2019-03-12) . . . 244
22.45 2.6 (2019-02-04) . . . 244
22.46 2.5 (2018-10-23) . . . 245
22.47 2.4 (2018-05-25) . . . 245
22.48 2.3 (2017-09-05) . . . 246
22.49 2.2 (2017-07-17) . . . 246
22.50 2.1 (2017-01-05) . . . 247
22.51 2.0 (2016-09-27) . . . 248
22.52 1.6.1 (2016-05-23) . . . 249
22.53 1.6.0 (2016-02-29) . . . 249
22.54 1.5.1 (2015-11-16) . . . 250

22.55 1.5.0 (2015-09-28) 251
22.56 1.4.1 (2015-05-05) 252
22.57 1.4.0 (2015-01-26) 252
22.58 1.3.3 (2014-08-21) 253
22.59 1.3.2 (2014-04-15) 253
22.60 1.3.1 (2014-04-14) 253
22.61 1.3.0 (2014-02-03) 254
22.62 1.2.3 (2013-09-05) 254
22.63 1.2.2 (2013-06-24) 255
22.64 1.2.1 (2013-06-17) 255
22.65 1.2.0 (2013-01-31) 255
22.66 1.1.2 (2012-11-29) 256
22.67 1.1.1 (2012-10-16) 256
22.68 1.1.0 (2012-10-12) 256
22.69 1.0.0 (2012-07-06) 256

23 Glossary **259**

Index **261**

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.

CHAPTER 1

Concepts

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 significantlly 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 for managing the backups of a specific Postgres server. If you need to switch backup methods, it's advisable to set up a new server in the Barman configuration.

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 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 community packages are provided by *PGDG*.

> **Note**
>
> If you are an EnterpriseDB customer, you should use the packages provided by EDB. This will ensure better compatibility with other EDB products.

There are four packages that make up the suite of Barman features: `barman`, `python3-barman`, `barman-cli` and `barman-cli-cloud`.

- `barman` is the main package and it must be installed.
- `python3-barman` is a package that holds the Python libraries of Barman. It is a package required by all the other Barman packages.
- `barman-cli` is an optional package that holds the `barman-wal-restore` and `barman-wal-archive` utilites. 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. Unless you have an EDB subscription, 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 version from 3.8 to 3.14
- Python modules:
  - `psycopg2` >= 2.4.2: Required to connect to the Postgres server
  - `python-dateutil`
  - `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.

2.5 Additional Requirements

Some Barman features — such as cloud storage integration and data compression — depend on optional Python libraries. These dependencies are not included by default when installing Barman through your system's package manager.

To enable functionalities such as `barman-cloud`, Snapshot Backups, or certain compression algorithms, you must install the appropriate extra dependencies listed in the table below. Make sure that these dependencies are installed for the same Python interpreter used by Barman.

Optional Libraries

Feature / Provider	Required Libraries
AWS S3	boto3
AWS EC2 Snapshots	boto3
Azure Blob Storage	azure-identity, azure-storage-blob
Azure VM Snapshots	azure-identity, azure-storage-blob, azure-mgmt-compute
Google Cloud Storage	google-cloud-storage
GCP Compute Snapshots	grpcio, google-cloud-storage, google-cloud-compute
Snappy Compression	python-snappy, cramjam >= 2.7.0
Zstandard Compression	zstandard
LZ4 Compression	lz4
Autocompletion	argcomplete

Tip

If you encounter `ModuleNotFoundError` when using `barman-cloud`, verify that:

1. You installed the libraries with the same Python interpreter used by Barman.
2. The installation completed successfully without permission or network errors.

CHAPTER 3

Quick start

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 *Database 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 *Database 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
```

CHAPTER 4

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:

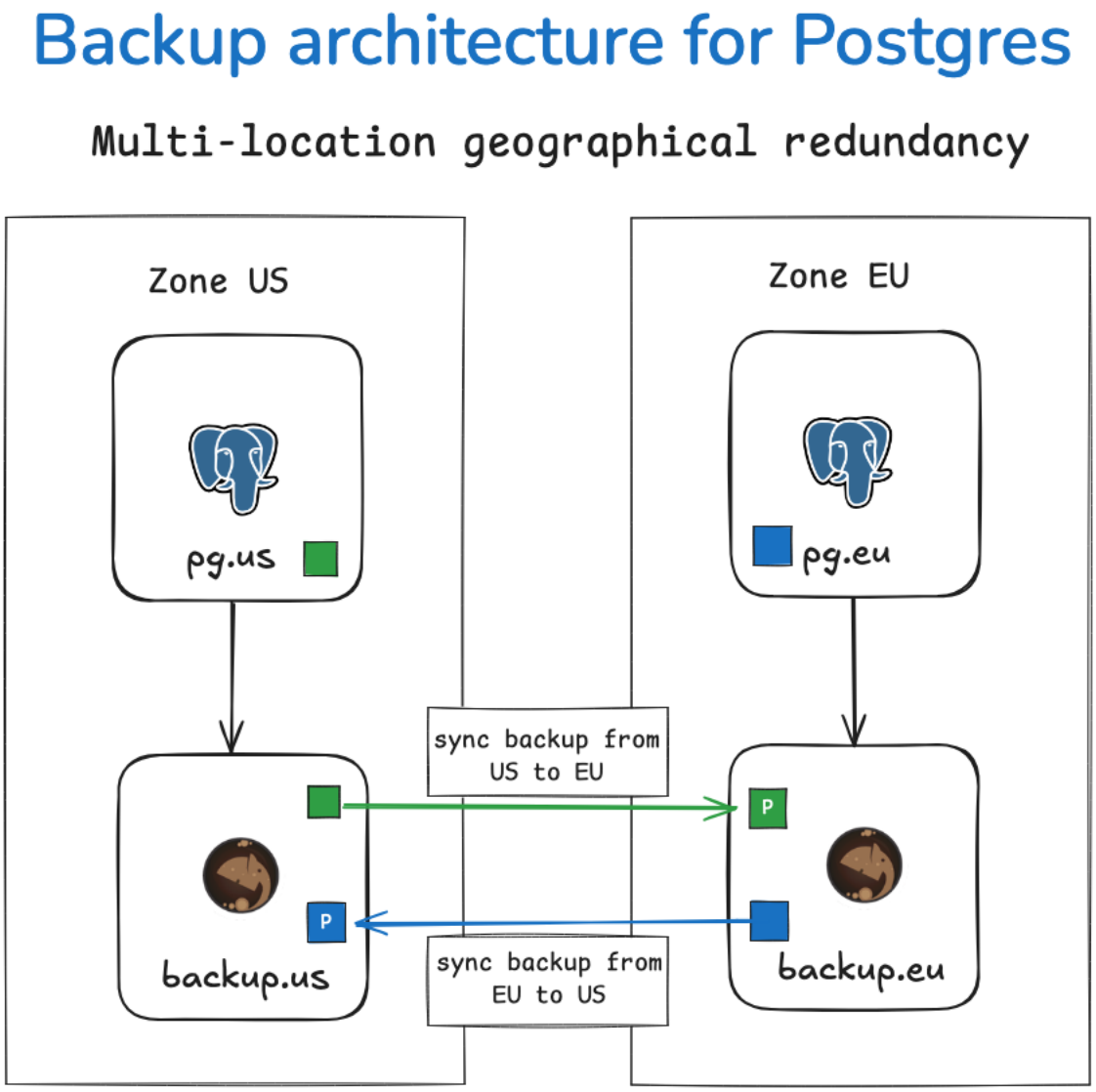
- Install Barman on a dedicated server.
- Do not share the same storage with your Postgres server.
- Integrate Barman with your infrastructure monitoring tools.
- Test everything before deploying to production.

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.



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

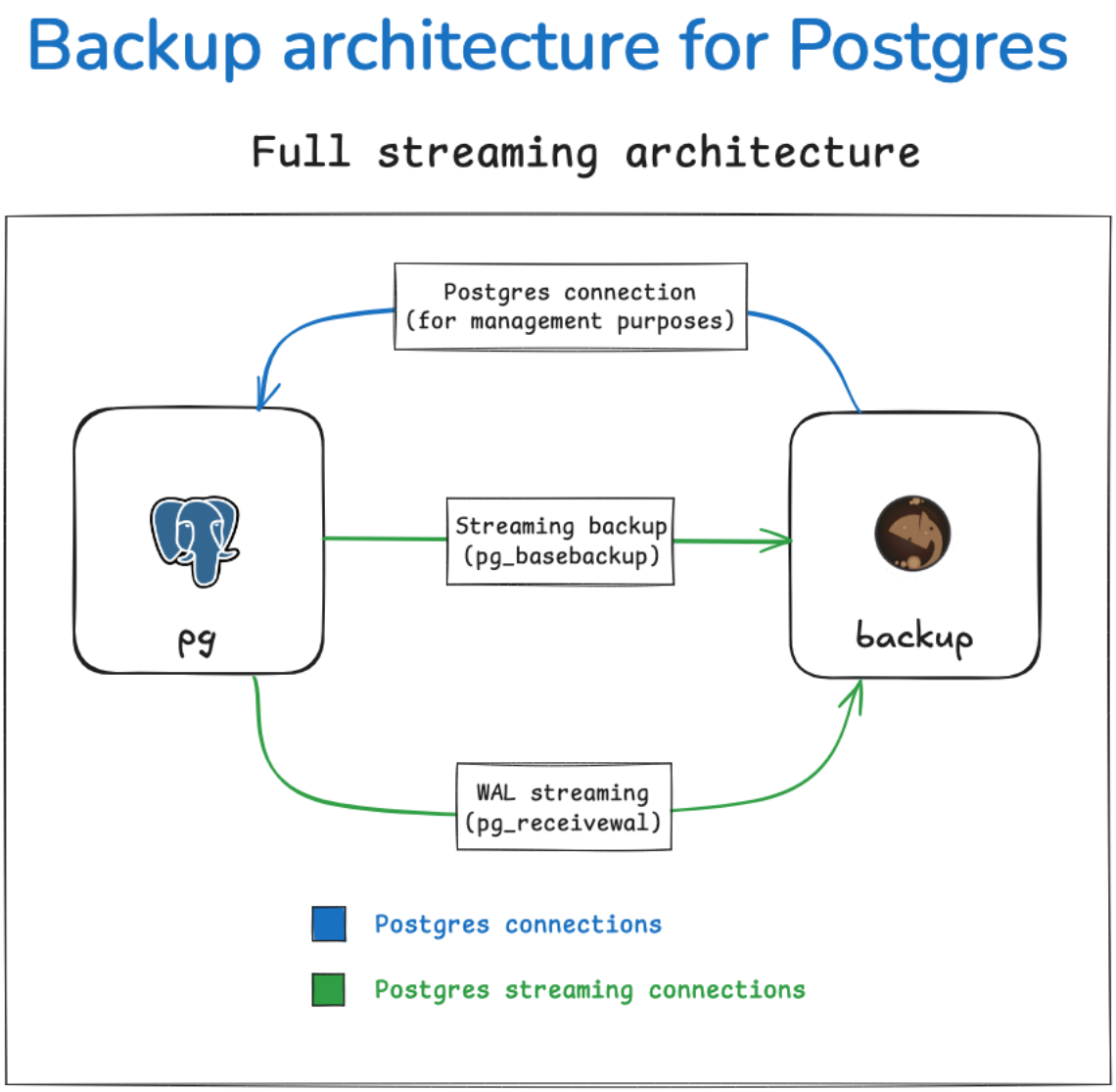
As stated in *Streaming Backups*, this approach uses the Postgres streaming protocol for transferring cluster files to your Barman server. This is done with the use of the `pg_basebackup` utility. In Barman, this method can be set by having `backup_method = postgres` in the configuration for the specific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

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.



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

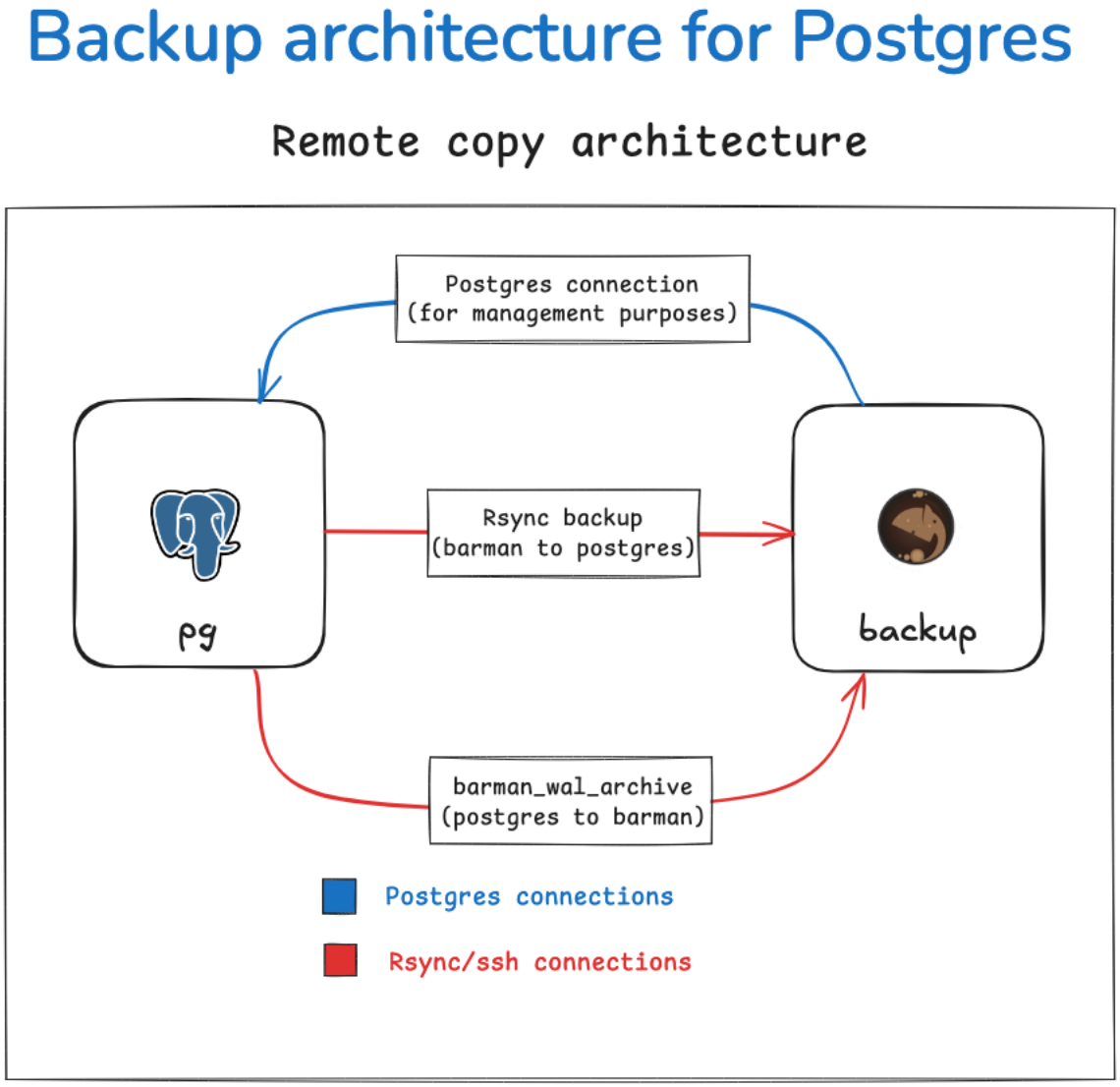
As stated in *rsync backups* concepts, this approach relies on Rsync to transfer backup files to your Barman server. This is done by putting your server in backup mode and transferring your cluster files using Rsync.

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.



In order to configure it, you will need:

1. A standard connection to Postgres for management, coordination, and monitoring purposes.

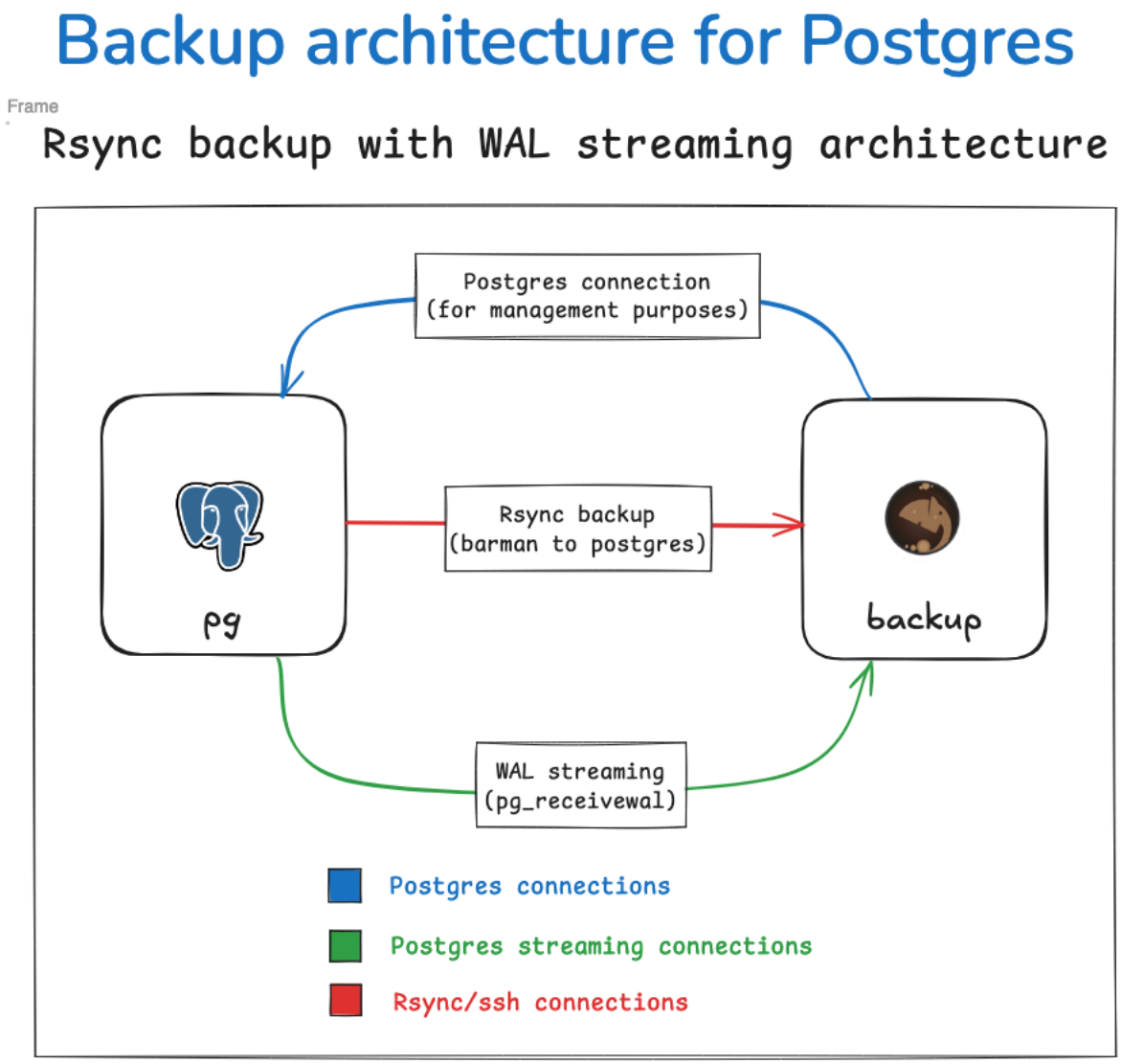
2. An SSH connection to be used by Rsync for base backup operations that allow the **barman** user on the Barman server to connect as the **postgres** user on the Postgres server.

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.



To accomplish this, you will need to configure the `backup_method` as `rsync`, and set `streaming_archiver` to `on` in the configuration of the specific Postgres server. 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 *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:

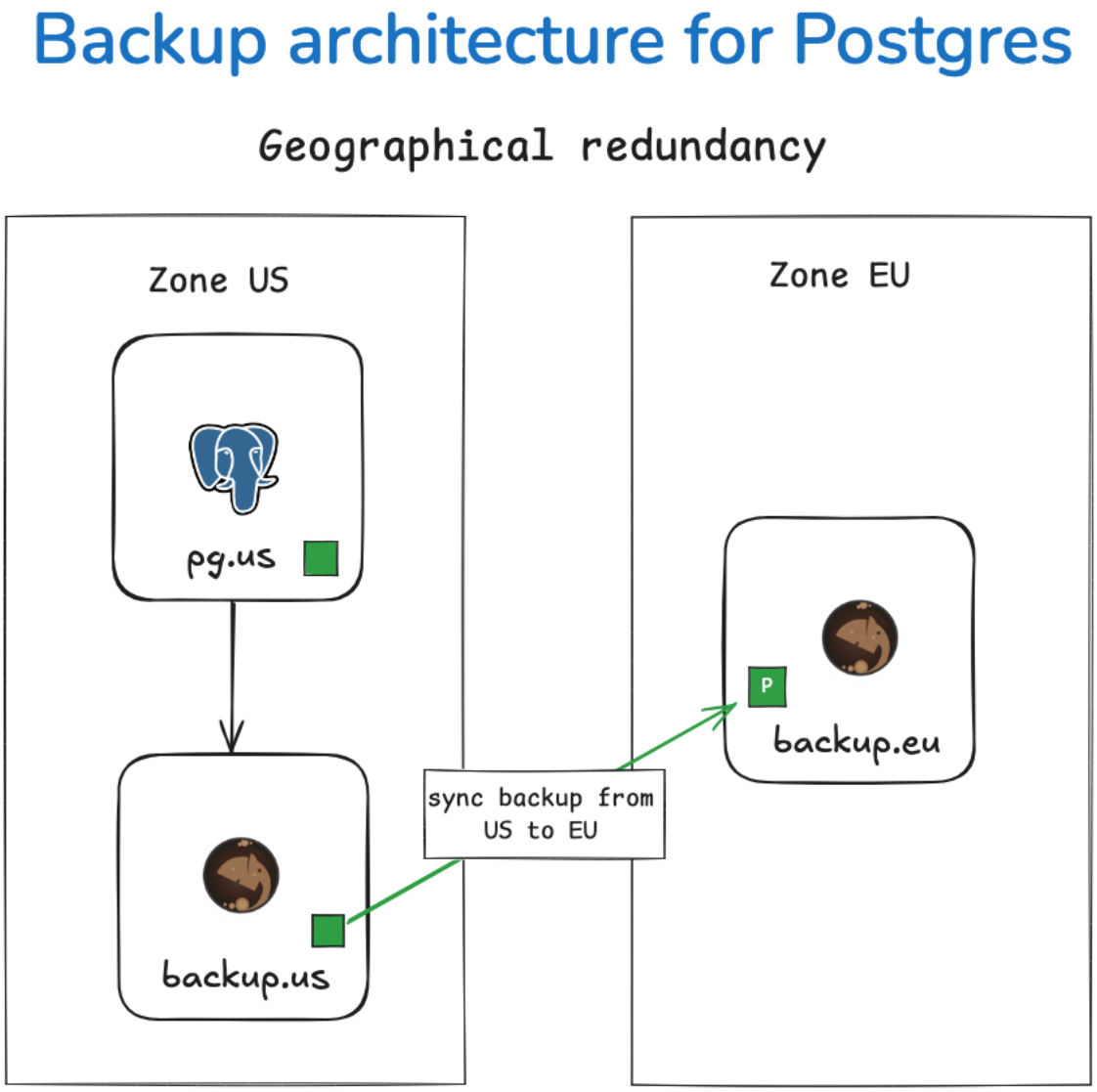
1. geographical redundancy (introduced in Barman 2.6)
2. hook scripts

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.



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 Database 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`. Aditionally, 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`. Aditionally, 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.

CHAPTER 6

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
```

Tip

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.

CHAPTER 7

Backups

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.

The most critical requirement for a Barman server is the amount of disk space available. You are recommended to plan the required disk space based on the size of the clusters to backup, number of WAL files generated per day, frequency

of backups, and retention policies.

Barman developers regularly test Barman with XFS and ext4 filesystems. Like PostgreSQL, Barman does nothing special for NFS mountpoints used for storing backups and WALs. The following points are required for safely using Barman with NFS:

- The `barman_lock_directory` should be on a local filesystem.
- Use at least NFS protocol version 4.
- The file system must be mounted using the hard and synchronous options (`hard`, `sync`).

> **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 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
```

7.2.1 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`.

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.

7.2.2 Streaming backups to the cloud

When configured with `backup_method = postgres` as well as `basebackups_directory` and `wals_directory` to cloud storage URLs, Barman supports streaming backups directly to a cloud storage. In this mode, backups are streamed from the Postgres server via `pg_basebackup` to the Barman server and dynamically pushed to the cloud in chunks, without ever storing the complete backup locally. WAL files are streamed from the Postgres server to the Barman server and uploaded to the cloud whenever archiving is triggered, which happens periodically or when manually running the `barman archive-wal` command.

This is currently the only method that enables block-level incremental backups in a cloud environment.

To configure this feature, set `backup_method = postgres` as well as `basebackups_directory` and `wals_directory` to cloud storage URLs. For example, if using an S3-compatible storage:

```
backup_method = postgres
basebackups_directory = s3://mybucket/barman/
wals_directory = s3://mybucket/barman/
```

Once configured, you can leverage all the capabilities of a Barman server, which includes incremental backups, retention policies, and more, while having your data safely stored in the cloud.

Note

Although it is possible to set different URLs for `basebackups_directory` and `wals_directory`, it is highly recommended to use the same URL, and possibly the same path, for both options. This ensures a consistent storage structure in the cloud as Barman organizes data of each server in dedicated subdirectories.

Warning

Changing the value of `basebackups_directory` and/or `wals_directory` on an active server is discouraged. Even if done with care, by deactivating the server, moving existing data to the new location, and reactivating it, there are potential risks of data loss or misconfiguration.

Instead, the safe option is to create a new server configuration with the new value for `basebackups_directory` or `wals_directory`, perform a new full backup to the new server, and then decommission the old server once the new one is fully operational.

When using this feature, Barman requires a local staging space to process chunks before uploading them to the cloud. The amount of staging space allowed to be used as well as its location can be configured with the options `cloud_staging_max_size` and `cloud_staging_directory`, or overridden at runtime with the `--cloud-staging-max-size` and `--cloud-staging-directory` CLI options.

This is an experimental feature. For this reason, a few limitations apply:

1. Currently, only S3-compatible storages are supported as destination;
2. Encryption of backups and WALs is not supported;

3. Compression of backups is not supported. WAL compression is supported except when using `pigz` or `custom` as compression methods;
4. Barman subcommands which require access to the backup or WAL files, such as `verify-backup`, `generate-manifest`, `rebuild-xlogdb` and `get-wal`, are not supported and will fail if executed;
5. The Barman *Geographical Redundancy* feature and its related commands are not supported.
6. For now this feature works only on Linux-based distributions and is not supported on BSD-derived systems (such as FreeBSD or OpenBSD).

7.3 Rsync Backups

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 configuration file for the specific Postgres server:

```
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.

7.3.1 File-Level Incremental Backups with Rsync

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.

7.4 Local Rsync Backups

Under special circumstances, Barman can be installed on the same server where the Postgres instance resides, with backed up data stored on a separate volume from `PGDATA` and, where applicable, tablespaces. Usually, these backup volumes reside on network storage appliances, with filesystems like NFS.

For these cases, Barman supports a local backup method called `local-rsync`. This method uses the same rsync-based backup approach as the remote `rsync` method, but without relying on SSH for the transport layer. Instead, it performs file copy operations locally on the same server where Postgres is running (for this reason it is required that Barman runs with the same user as Postgres).

The only requirement for local backups is that Barman runs with the same user as the Postgres server, which is normally `postgres`. Given that the community packages by default install Barman under the `barman` user, this use case requires manual installation procedures that include:

- cron configurations
- log configurations, including logrotate
- Barman home configuration.

The recommended way to set up Barman for local backups is to follow these steps after installing Barman:

1. Set `barman_user = postgres` (or the operating system user running the server if using a different one) in the global configuration file to ensure Barman has access to read the data files.
2. Configure the backup directory and other relevant directories to be accessible by the same user. This can be done by setting the appropriate options in the configuration file (e.g., `backup_directory`, `barman_home`, `log_filename`, etc.) to point to paths with the correct permissions. Don't use the default paths provided by the community packages, as they are owned by the `barman` user, and permissions and ownership of those directories will be restored to the original state during package updates, which can cause issues.

Refer to *Configuration options* for more details.

In order to use local backup for a given server in Barman, you need to set `backup_method` to `local-rsync`.

7.5 Local-to-Cloud Backups

Barman can perform backups directly to cloud object storage using the `local-to-cloud` backup method. This method reads data directly from the Postgres PGDATA directory on the local filesystem and uploads it directly to cloud storage without requiring intermediate local storage for the backup data.

Note

Currently, only **Amazon S3** and S3-compatible storage are supported. Support for Azure Blob Storage and Google Cloud Storage will be added in future versions of Barman.

Like other backup methods, this approach uses the Postgres's low-level backup API for coordination with the database server, but uploads the backup data directly to the cloud as it is read from disk.

Important

The `local-to-cloud` backup method requires:

- Postgres 9.6 or later (for concurrent backup support).
- Barman must have direct filesystem access to the PGDATA directory.
- The user executing Barman must be the owner of the database cluster.
- AWS credentials properly configured (or credentials for S3-compatible storage).
- Network access to a properly configured S3 or S3-compatible object storage bucket.

To configure local-to-cloud backups, set the `backup_method` to `local-to-cloud` and configure cloud storage path URLs:

```
backup_method = local-to-cloud
basebackups_directory = s3://bucket-name
wals_directory = s3://bucket-name
conninfo = host=localhost dbname=postgres user=postgres
```

The `basebackups_directory` must be an S3 storage URL (e.g., `s3://bucket/path`). Similarly, `wals_directory` should point to an S3 storage location for WAL archiving.

Note

It's recommended to use the same bucket and path for both `basebackups_directory` and `wals_directory` to maintain a consistent storage structure in the cloud, as Barman organizes data of each server in dedicated subdirectories.

Warning

Changing the value of `basebackups_directory` and/or `wals_directory` on an active server is discouraged. Even if done with care, by deactivating the server, moving existing data to the new location, and reactivating it, there are potential risks of data loss or misconfiguration.

Instead, the safe option is to create a new server configuration with the new value for `basebackups_directory` or `wals_directory`, perform a new full backup to the new server, and then decommission the old server once the new one is fully operational.

7.5.1 AWS S3 Configuration Options

The `local-to-cloud` backup method supports various AWS S3 configuration options:

Bandwidth Control:

- `bandwidth_limit`: Maximum transfer rate in kilobytes per second for cloud uploads (default: `0`, meaning no limit).

AWS S3 Options:

- `aws_region`: AWS region for the S3 bucket.
- `aws_profile`: AWS profile name for authentication.
- `aws_encryption`: S3 encryption method (`AES256` or `aws:kms`).
- `aws_sse_kms_key_id`: KMS key ID for server-side encryption.
- `aws_read_timeout`: S3 read timeout in seconds.

Cloud Options:

- `cloud_upload_max_archive_size`: Maximum size per cloud archive (default: `100G`).
- `cloud_upload_min_chunk_size`: Minimum chunk size for multipart uploads.
- `cloud_delete_batch_size`: Number of files to delete in a single batch operation.

Compression:

- `compression`: WAL compression algorithm (`gzip`, `bzip2`, `xz`, `snappy`, `zstd`, `lz4`, or `none`).
- `compression_level`: Compression level - can be set to `low`, `medium`, `high`, or a number according to the chosen algorithm.

You can find more information on compression settings in the *Backup Compression* section below.

7.5.2 When to Use Local-to-Cloud

The `local-to-cloud` backup method is appropriate when:

- Postgres and Barman run on the same machine, or Barman has direct local filesystem access to the PGDATA directory.
- You want to back up directly to AWS S3 or S3-compatible storage without intermediate local storage for the backup data itself.
- You want a simple, unified cloud backup configuration through the main `barman` CLI.

7.5.3 Limitations

The `local-to-cloud` backup method has the following limitations:

- It **does not** support incremental backups (unlike the `postgres` method with Postgres 17+, which supports block-level incremental backups).
- It **does not** support file-level deduplication through hard links (unlike `rsync` with `reuse_backup`).
- **Only AWS S3 and S3-compatible storage are currently supported** - Azure Blob Storage and Google Cloud Storage support will be added in future versions.
- It requires local filesystem access to PGDATA (unlike `rsync` or `postgres` methods for remote servers).

7.5.4 Example Configuration

Here is a complete example configuration for local-to-cloud backups with AWS S3:

```
[myserver]
backup_method = local-to-cloud
barman_user = postgres
basebackups_directory = s3://my-backup-bucket
wals_directory = s3://my-backup-bucket
conninfo = host=localhost dbname=postgres user=postgres
bandwidth_limit = 5120  ; 5 MB/s
compression = gzip
aws_region = us-west-2
aws_profile = barman
archiver = on
```

With this configuration, Barman will perform backups by reading data directly from the local PGDATA directory and uploading it to the specified S3 bucket with bandwidth throttling and gzip compression for WAL files.

7.5.5 WAL archival with local-to-cloud

When using the `local-to-cloud` backup method, WAL files can be archived to the cloud using the *barman cloud-wal-archive* command, which uploads WAL files via the `archive_command` to the cloud storage specified in the configuration.

```
archive_command = 'barman cloud-wal-archive myserver %p'
```

To reduce WAL archival backlog during periods of high WAL generation, the `--parallel` flag (or the `cloud_wal_archive_parallel` configuration option) can be used to upload additional WAL files concurrently:

```
archive_command = 'barman cloud-wal-archive --parallel 2 myserver %p'
```

When `--parallel N` is set (N > 1), up to `N - 1` extra WAL files that are ready in `pg_wal/archive_status` are uploaded in background processes after the primary WAL has been successfully archived.

7.6 General Backup Settings

7.6.1 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`.

7.6.2 Backup from a Standby Server

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.

7.6.3 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 = tbname:bwlimit[, tbname: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`, `postgres` and `local-to-cloud` backup methods, but the `tablespace_bandwidth_limit` option is only applicable when using `rsync`.

7.6.4 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`.

7.6.5 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.

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 `staging_path` and `staging_location` to enable recovery of compressed backups. Refer to the *Recovering Compressed backups* section for details.

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.

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.

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

7.6.6 Backup Encryption

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

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.

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.

7.6.7 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 or encrypted backups, defined by the `staging_path` and `staging_location` options (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.

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.

7.6.8 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.

7.6.9 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.

CHAPTER 8

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 snapshot backups, include the following parameters in the Barman configuration file for the specific Postgres server:

```
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.

8.1 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.

8.2 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
```

8.3 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
```

8.4 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`.
3. 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.

8.5 Backup Process

Here is an overview of the snapshot backup process:

1. **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`.
2. Barman initiates the backup using the Postgres backup API.
3. 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.
4. 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.

8.5.1 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

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

    device_name          : /dev/sdf
```

(continues on next page)

(continued from previous page)

```
    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            : 000000010000000000000001A
    End WAL              : 000000010000000000000001A
    Number of WALs       : 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:
    Number of files      : 1
    Disk usage           : 16.0 MiB
    WAL rate             : 5048.32/hour
    Last available       : 000000010000000000000001B

  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",
          "mount_point": "/opt/postgres"
        },
        "provider": {
          "device_name": "pgdata",
          "snapshot_name": "barman-av-ubuntu20-primary-pgdata-20230123t131430",
          "snapshot_project": "project_id"
        }
```

(continues on next page)

(continued from previous page)

```
      },
      {
        "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.

CHAPTER 9

WAL archiving

Barman also offers additional features regarding WAL archiving.

9.1 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 (uses Python's internal compression library);
- `pygzip`: for Gzip compression (currently the same as `gzip`)(Deprecated);
- `pigz`: for Pigz compression (requires the `pigz` utility);
- `bzip2`: for Bzip2 compression (uses Python's internal compression library);
- `pybzip2`: for Bzip2 compression (currently the same as `bzip2`)(Deprecated);
- `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. (Deprecated)

> **Note**
>
> Cloud WAL storage (when `wals_directory` points to a cloud object store) only supports in-memory compression algorithms: `gzip`, `bzip2`, `xz`, `zstd`, `lz4`, and `snappy`. `pigz` and `custom` are not supported because they rely on external processes and cannot compress in-memory.

Deprecated since version 3.16: The `pygzip` and `pybzip2` compression options are deprecated and will be removed in a future release. Use their equivalents `gzip` and `bzip2` instead.

Deprecated since version 3.17: The `custom` compression option and related configuration options (`custom_compression_filter`, `custom_decompression_filter`, and `custom_compression_magic`) are deprecated and will be removed in a future release. Use one of the built-in compression algorithms instead.

> **Note**
>
> When using `pigz` Barman forks 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.

9.2 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 `gpg` and `none` (no encryption) are accepted as values. A detailed guide on how to configure encryption is available in *Backup Encryption*.

9.3 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 configuration on the Barman server must 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 "Syncronous 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)
```

(continues on next page)

(continued from previous page)

```
    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)
```

CHAPTER 10

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.

10.1 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.

10.2 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.

> **Note**
>
> Backups can also be added to the catalog by importing a previously exported tarball with commands-barman-import-backup.

10.2.1 `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␣
↪MiB
```

- `svr_pg17` is the server name.
- `20240901T103000` is the backup ID.
- `F` is the backup type label.
- `Nov 1 11:12:03 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.

10.2.2 `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
```

(continues on next page)

(continued from previous page)

```
Server information:
  Checksums            : on
  WAL summarizer       : on

Base backup information:
  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
  Number of WALs       : 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:
  Number 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 compressed, the `Backup Compression` field is shown in the output, containing the compression algorithm that was used.
- 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.

CHAPTER 11

Recovery

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.

11.1 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.

11.2 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.

11.3 Delta Recovery

Use `--delta-restore` to allow barman to restore a backup into a pre-existing destination directory along with any custom tablespace by overwriting it.

Another option is to include `delta-restore` inside the `recovery_options` configuration at the global/server level prior to a recovery operation to perform a delta recovery without the need to specify the `--delta-restore` flag.

```
recovery_options = 'delta-restore'
```

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

> **Note**
>
> In case you are doing a delta recovery with `get-wal` enabled, you can add both in the configuration as `recovery_options = get-wal,delta-restore`.

Important

The `delta-restore` option is supported **only** in the following scenarios:

- **Local restore of plain backups** - incremental, compressed, encrypted, or multi-format backups are *not* supported.
- **Remote restore with** `staging_location=local` - using `staging_location=remote` is *not* supported.

11.4 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.

11.5 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

11.6 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.

11.6.1 Partial WAL files during recovery

When using `--no-get-wal`, Barman can optionally copy `.partial` WAL files to the recovery destination by specifying the `--partial-wal` flag. A `.partial` WAL file is a WAL segment that has been partially written by Postgres but not yet completed and archived.

```
barman restore SERVER_NAME BACKUP_ID DESTINATION_PATH --no-get-wal --partial-wal
```

Including `.partial` files in the recovery reduces the RPO (Recovery Point Objective) by ensuring that any transactions written since the last completed WAL segment are available during recovery.

Note

When using `--get-wal`, `.partial` files are already handled transparently during recovery — `barman-wal-restore` (remote) and `barman get-wal` (local) both support `.partial` files natively and do not require the `--partial-wal` flag.

Barman considers `.partial` files from two sources:

- **Streaming WALs directory** (`streaming_wals_directory`): the file being actively written by `pg_receivewal`. Only the segment immediately following the last archived WAL is considered; any other `.partial` files in the directory are ignored. This guarantees continuity between the archived WALs and the partial segment. Files from this source are always plain and require no decompression or decryption.
- **Main WAL archive** (`wals/`): `.partial` files archived with the suffix intact — typically when a standby is promoted and PostgreSQL calls `archive_command` with the in-progress segment. These files are yielded as part of the normal WAL range required for recovery. If `barman cron` has compressed or encrypted them, they are decompressed and/or decrypted before being written to the destination.

The following additional rules apply to both sources:

- `.partial` files on a different timeline are automatically excluded because the expected segment name encodes the target timeline.
- When `--target-lsn` is used, only a `.partial` file whose segment name is less than or equal to the WAL segment containing the target LSN is included; any segment beyond that point is skipped.
- When `--target-immediate` is used, `.partial` files are skipped entirely since recovery stops at the first consistent state.
- The `.partial` suffix is stripped from all filenames in the recovery destination so that PostgreSQL can locate them by their plain segment name.

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.

> **Note**
>
> In case you are doing a delta recovery with `get-wal` enabled, you can add both in the configuration as `recovery_options = get-wal,delta-restore`.

11.6.2 Using `get-wal` for local recovery

The barman get-wal command must always run as the `barman` user on local recovery, as it requires access to the WAL catalog managed by Barman.

```
restore_command = 'barman get-wal SERVER_NAME %f > %p'
```

This assumes that the PostgreSQL server will also run as the `barman` user.

If you choose to run PostgreSQL with another user (for example, `postgres`), you are responsible for:

- Updating the ownership of `PGDATA` and all tablespaces to match the user who will run the server.
- Adjusting the `restore_command` so that it can still invoke `barman get-wal` as the `barman` user (e.g. by configuring `sudo` or other mechanisms to become that user).

In other words, when using a user different from `barman` to run PostgreSQL on a local recovery, extra configuration is required to ensure that WAL files can still be retrieved from Barman, and that the server will be able to start up.

For 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 *
```

11.6.3 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.

11.6.4 Using `cloud-wal-restore` for cloud recovery

When restoring from a cloud backup, whether taken with `backup_method = local-to-cloud` or `backup_method = postgres`, `barman cloud-wal-restore` can be used as the `restore_command` to retrieve WAL files directly from the configured cloud provider, bypassing the Barman server.

> **Note**
>
> Restoring a cloud backup in `--no-get-wal` mode (default) is not supported. Hence, the only way to restore WALs stored in the cloud is by using `cloud-wal-restore` as the `restore_command`. An attempt to use `--no-get-wal` in `barman restore` for a cloud backup will be ignored and warned.

An example of a `restore_command` for cloud recovery is as follows:

```
restore_command = 'barman cloud-wal-restore --parallel 5 SERVER_NAME %f %p'
```

It goes without saying that `wals_directory` must point to the cloud location where WAL files are stored, e.g., `wals_directory = s3://barman/wals`.

11.7 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, regardless of whether `staging_location=remote` is set. This is because decryption is designed to occur locally on the Barman node. The directory used for decrypting is also defined by the `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. When additional operations are required, the staging directory is removed after the subsequent operation have finished.

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 `encrytion_passphrase_command` is invoked once for each WAL file being fetched through the `restore_command`.

Important

When `staging_location=remote` is used, `staging_path` must point to a location accessible for reading and writing on the local node as well. This is because decryption always takes place on the local node.

11.8 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. If restoring locally, the compressed backup files are decompressed directly into the restore destination directory. If restoring remotely, the behavior depends on the `staging_location` setting:
   - `staging_location=local`: The compressed backup is decompressed in the `staging_path` on the Barman server, then copied to the remote restore destination using rsync.
   - `staging_location=remote`: The compressed backup is copied to the remote server's `staging_path` using rsync and then decompressed in the remote restore destination.
2. 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.
3. When additional operations are required, the staging directory is removed after the subsequent operation have finished.

11.9 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.

The process involves the following steps:

1. Barman creates a synthetic backup by combining the chain of backups. If restoring locally, the chain of backups are combined directly into the restore destination directory. If restoring remotely, the behavior depends on the `staging_location` setting:
   - `staging_location=local`: The chain of backups is combined in the `staging_path` on the Barman server, then copied to the remote restore destination using rsync.
   - `staging_location=remote`: The chain of backups is copied to the remote server's `staging_path` using rsync and then combined in the remote restore destination.
2. When additional operations are required, the staging directory is removed after the subsequent operation have finished.

Important

If any backups in the chain were taken with checksums disabled, but the final backup has checksums enabled, the resulting syntethic backup may contain pages with invalid checksums. Please refer to the limitations in the

pg_combinebackup documentation for more details.

11.9.1 Combine Modes

The method used to create files in the combined backup can be controlled with the `--combine-mode` option of the barman restore command.

Supported modes are:

copy — Performs a standard file copy for each file in the chain. This is the default mode and works on all systems. It provides the highest level of safety since files in the resulting backup are fully independent from the originals, at the cost of more time, and extra disk space and I/O.

copy-file-range — Uses the `copy_file_range` system call for efficient file copying. On some file systems, this can achieve results similar to `--clone` by sharing physical disk blocks, while on others it still performs full copies but through an optimized path. This mode is currently supported on Linux and FreeBSD. If the backup manifest is not available or does not contain compatible checksums, files are still copied, but also read block-by-block for checksum verification.

link — Creates hard links, when possible, instead of copying files to the synthetic backup. This approach might be faster in some cases, and save disk space, but it introduces potential risks: changes made to the restore directory will also affect the original backup directories, and vice versa. Therefore, this mode should only be used when the input directories are disposable copies that will be deleted after combination. The input backups and output directory must reside on the same file system. If the backup manifest does not contain compatible checksums, hard links will still be created, but files will also be read block-by-block for checksum calculation.

clone — Uses file cloning capabilities (such as reflink) provided by some file systems to create lightweight copies of files that initially share data blocks. This approach combines the speed and space efficiency of link with the isolation of copy. Modifications to either copy do not affect the other. Support for this mode depends on the underlying file system (for example, XFS and Btrfs support cloning).

For more information about the behavior of these modes, see the pg_combinebackup documentation.

11.10 Recovery Pipeline for multi-format backups

Backups can be compressed, encrypted, incremental or a combination of these. Barman streamlines the recovery process by automatically handling decompression, decryption, and incremental backup chain combination as needed during restore. When you issue the `barman restore` command, Barman detects the backup type and performs all necessary operations in the correct order, removing any manual intervention.

For example:

- If the backup is encrypted, Barman decrypts it using the configured `encryption_passphrase_command`.
- If the backup is compressed, Barman decompresses it before restoring the data files.
- If the backup is a block-level incremental, Barman combines the backup chain using `pg_combinebackup` to produce a full, restorable backup.
- If the backup is both encrypted, compressed, and incremental, Barman will automatically process it in the following order: decryption, decompression, and incremental chain combination. For each step, Barman creates a temporary staging directory in the selected location. After a step is completed, the staging directory from the previous step is deleted. This means no more than two staging directories will use disk space at the same time.

You can control the location of these operations using the `staging_path` and `staging_location` options, either in the configuration file or as command-line arguments. This flexibility allows you to optimize for available disk space and network bandwidth, especially in remote recovery scenarios.

For example, if a backup is encrypted, compressed, and incremental and it is a remote recovery with `staging_location=remote` and `staging_path=/tmp`:

1. All backups in the chain are decrypted to a staging path in the Barman node, e.g. `/tmp/decrypt-staging-location`.
2. Then they are copied with rsync to the staging directory in the remote node, e.g. `/tmp/rsync-staging-location` on the remote node, and after the copy has finished, the staging path `/tmp/decrypt-staging-location` on the Barman node is deleted.
3. Next, the copied backups are decompressed into another remote staging directory, e.g. `/tmp/decompress-staging-location` on the remote node, and after decompression, the staging path `/tmp/rsync-staging-location` on the remote node is deleted.
4. Finally, the decompressed files are combined into a synthetic backup with the restore destination as its output.
5. After this, the decompression staging directory `/tmp/decompress-staging-location` is also deleted.

When `staging_location=local`, all operations are executed on the Barman server, but the order differs slightly: the rsync copy is deferred to the end. The combine operation uses its own `staging_path`, and the final step is to transfer the synthetic backup to the restore destination on the remote node.

11.11 Recovering Backups from the Cloud

Restoring a cloud backup, whether taken with `backup_method = local-to-cloud` or `backup_method = postgres`, follows the same process as restoring a local backup: a simple `barman restore` command. The main difference is that it offers some alternative restore flows, as described in the following sections.

For cloud WAL restore, check the *Using cloud-wal-restore for cloud recovery* section.

11.11.1 Restoring Cloud Backups on a New Server

The most efficient way to restore a cloud backup is to pull it directly to where your cluster will run, bypassing any centralized Barman server. To do this, run `barman restore` directly on the destination host where your cluster is going to be started.

In this approach, a Barman instance is installed and configured with the only purpose of restoring a cloud backup, being inactive for any other task. A minimal configuration for this Barman instance could look like the following:

```
[pg]
description = "Minimal configuration for cloud backup restore"
wals_directory = s3://barman/backups
basebackups_directory = s3://barman/backups
active = off
```

Note

`basebackups_directory` and `wals_directory` must point to an existing cloud location where your backups and WAL files are stored, respectively.

Important

You MUST set `active = off` to prevent this Barman instance from running maintenance tasks that could interfere with your data. Without this setting, Barman refuses to search for any backups in the cloud.

11.11.2 Restoring Cloud Backups from the same Barman Server

The easiest way to restore a cloud backup is by triggering the restore from the same Barman server that triggered the backup. This approach is as straightforward as running:

```
barman restore SERVER_NAME BACKUP_ID /path/to/destination
```

Naturally, this is only useful if you want to start your cluster on the same host where the Barman instance that triggered the backup is installed.

To restore it to a different host, you can use the `--remote-ssh-command` option and perform a remote copy via SSH:

```
barman restore SERVER_NAME BACKUP_ID /path/to/destination --remote-ssh-command "ssh␣
↪postgres@target-server"
```

> **Warning**
>
> This approach routes backup data through the Barman server: the backup is first pulled from the cloud to the Barman host and then transferred to the target host. This can lead to increased restore times and higher bandwidth usage on the Barman server.

11.12 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.

If recovering with `--no-get-wal`, Barman will copy all archived WALs to the destination node. To also recover transactions recorded in a `.partial` WAL segment, use the `--partial-wal` flag. See *Partial WAL files during recovery* for details.

11.13 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.

11.14 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.

11.14.1 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.

11.14.2 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.

11.14.3 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.

CHAPTER 12

Retention policies

12.1 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.

> **Note**
>
> Retention policies apply only to backups with status `DONE`. Backups that are incomplete (for example, those in the `WAITING_FOR_WALS` state) or `FAILED` are not accounted by retention rules.

12.2 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.

12.2.1 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.

12.2.2 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.

12.2.3 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.

12.3 Key Objectives of Retention Policies

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

12.3.1 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.

12.3.2 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.

12.3.3 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.

12.4 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.

12.5 Scope of retention policies

Barman allows you to define retention policies by two methods.

12.5.1 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.

12.5.2 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.

12.5.3 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.

> **Note**
>
> Incremental backups are not eligible for the `keep` command. This restriction exists because incremental backups rely on their parent backups and cannot be used on their own for recovery. Keeping an incremental without its base would serve no purpose, as it is unusable independently.
>
> If you attempt to apply keep to an incremental backup, Barman will return an error indicating that incremental backups cannot be kept.
>
> **Only the full base backup —the one that starts the backup chain— can be marked with keep.**

12.6 Use cases

12.6.1 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.

12.6.2 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.

12.6.3 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.

12.7 How retention policies are enforced

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

12.8 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.

12.9 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`.

12.10 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.

CHAPTER 13

Backup export and import

13.1 Overview

Barman can export an existing backup as a single self-contained tarball and later import it back into the Barman catalog. This bridges Barman with enterprise data-management workflows that rely on external archival systems, for example writing long-term copies to tape or moving backups across storage tiers as part of a Grandfather/Father/Son strategy.

The exchange format is the following: the export is a tarball of files; the import only requires that the tarball is presented on disk to the target Barman server. Barman does not integrate with any specific external product: any system that can archive a file from disk and later restore it back can carry an exported backup end-to-end.

The outer tarball can optionally be compressed with the supporetd algorithms (`gzip`, `bzip2` or `xz`). Compression applies only to the outer tarball: data files and WAL segments inside are not recompressed, and any compression they already carry (for example WAL or backup compression configured on the source server) is preserved as-is. On import, the compression format is detected automatically from the tarball and decompression is handled transparently.

13.2 Limitations

- Only full backups in `DONE` state can be exported or imported; incremental backup chains are currently not supported for this feature.
- The feature is supported only on Barman servers using local storage.
- The exported tarball is tied to the source cluster's `systemid`; importing into a different cluster is not supported.
- The original tarball filename must be preserved through the archival chain and renaming it breaks the integrity check.
- Encryption is preserved transparently; encrypted backups are exported and imported as-is.

13.3 Tarball layout

The exported tarball contains everything Barman needs to re-register and restore the backup:

```
backup-export-SERVER-BACKUP_ID-TIMESTAMP-CHECKSUM.tar[.gz|.bz2|.xz]
 ├─── identity.json            # server identity
 ├─── backup.info              # backup metadata
 ├─── barman.json              # additional Barman metadata
 ├─── backup/                  # backup data tree (PGDATA + tablespaces)
 ├─── wals/<hash>/<wal_name>   # WAL segments required for consistency
 └─── xlog.db                  # WAL metadata
```

The symlinks normally found in `pg_tblspc/` are not stored in the tarball: tablespace data is stored as plain subdirectories and the symlinks are reconstructed during import.

13.4 Tarball filename and integrity

The filename follows the pattern:

```
backup-export-{SERVER_NAME}-{BACKUP_ID}-{TIMESTAMP}-{CHECKSUM}.tar[.gz|.bz2|.xz]
```

where `TIMESTAMP` is the export time in UTC (`YYYYMMDDTHHMMSS`) and `CHECKSUM` is the first eight characters of the SHA-256 hash of the whole tarball. When `barman import-backup` is run, it recomputes the SHA-256 of the file and compares it against the value embedded in the filename, refusing the import on mismatch.

> **Warning**
>
> Do not rename the exported tarball when transferring it through your archival system. Renaming breaks the integrity check and `barman import-backup` will refuse it. If your archival system stores files under a different name, restore them to the original name before importing.

13.5 Usage

The following walk-through covers a full lifecycle: exporting a backup, handing it off to an external archival system, restoring it back to disk, and importing it into a Barman server for recovery.

1. **Pick a full backup** in `DONE` state on the source server:

   ```
   barman list-backups main
   barman show-backup main 20260513T101530
   ```

2. **Export the backup** to an existing directory. Optionally compress the outer tarball with `--compression {gzip|bzip2|xz}` and tune the level with `--compression-level`:

   ```
   barman export-backup main 20260513T101530 /var/tmp/barman-exports
   ```

   Barman writes a single file under `/var/tmp/barman-exports` whose name embeds the export timestamp and integrity checksum, for example:

   ```
   backup-export-main-20260513T101530-20260513T134207-9f3a1c20.tar
   ```

3. **Archive the tarball** with your external backup product. Treat the file as opaque; the only requirement is that the original filename is preserved through the archival chain.
4. **Restore the tarball back to disk** on the target Barman server using your external product, into any directory of your choice.
5. **Import the backup** into the target catalog:

```
barman import-backup main \
    /var/tmp/barman-imports/backup-export-main-20260513T101530-20260513T134207-
↪9f3a1c20.tar
```

   The import is atomic: data is extracted into a staging directory under `basebackups_directory`, verified end-to-end, and only then moved into place. If any step fails, the staging directory and any partial WAL changes are cleaned up automatically and the catalog is left untouched.

   On success, the backup is annotated with `KEEP:STANDALONE` so that retention policies will not remove it, and a warning is printed to make this visible. Use `barman keep main BACKUP_ID --release` if you want the imported backup to participate in retention again (see commands-barman-keep).

6. **Verify and restore** to confirm the imported backup is fully usable:

```
barman list-backups main
barman show-backup main 20260513T101530
barman restore main 20260513T101530 /var/lib/postgresql/recovered
```

CHAPTER 14

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.

14.1 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.

14.2 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
    ...
```

14.3 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)
```

14.4 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.

CHAPTER 15

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.

15.1 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.

Barman searches for its configuration file in a predefined order and uses the first one it finds. The search order is as follows:

- `~/.barman.conf` - per-user configuration file.
- `/etc/barman.conf` - main global configuration file.
- `/etc/barman/barman.conf` - alternative global configuration file.

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 configuration overrides that can be applied to backups of Postgres 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.

> **Note**
>
> **Comment Handling**: Only line comments are supported (lines starting with `#` or `;`). Inline comments (`#` or `;` appearing after configuration values on the same line) are not supported and will be parsed as part of the configuration value, which may cause unexpected behavior.

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`.

15.2 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.

15.2.1 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 | local-to-cloud`.

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. If using NFS, or any other network filesystem to store the backups and WAL files, make sure that the lock directory lives in a 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.

> **Note**
>
> Multi-host connection strings (e.g. `host=primary_db,replica_db`) and `target_session_attrs` are not supported. Each connection string must point to a single, specific host.

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.

> **Note**
>
> Multi-host connection strings are not supported. See `conninfo` for details.

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`.

> **Note**
>
> Multi-host connection strings are not supported. See `conninfo` for details.

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.

15.2.2 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.

warehousepg_dbid

Specifies the integer identifier for a WarehousePG database. When set, this value is passed to `pg_basebackup` via the `--target-gp-dbid` parameter for WarehousePG compatibility. This option is required when backing up WarehousePG segments.

> **Important**
>
> This option is only relevant for WarehousePG databases and should not be set for standard PostgreSQL databases.
>
> Furthermore, Barman does not handle backups and restores for WarehousePG clusters. The orchestration of backup and restore operations for WarehousePG clusters must be managed externally by the user.

> **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

Destination for base backup files. It can be a local path if storing backups locally, or a cloud storage URL if streaming backups to the cloud as described in *Streaming backups to the cloud*. Defaults to `<backup_directory>/base`.

> **Warning**
>
> Changing the value of `basebackups_directory` on an active server is discouraged. Even if done with care, by deactivating the server, moving existing data to the new location, and reactivating it, there are potential risks of data loss or misconfiguration.
>
> Instead, the safe option is to create a new server configuration with the new value for `basebackups_directory`, perform a new full backup to the new server, and then decommission the old server once the new one is fully operational.

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.

15.2.3 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`, or when `backup_method` is `postgres` or `local-to-cloud` and `basebackups_directory` and/or `wals_directory` point to S3.

Scope: Global / Server / Model.

aws_region

When using `backup_method = snapshot`, indicates the AWS region where the EC2 VM and storage volumes, as defined by `snapshot_instance` and `snapshot_disks`, are located.

When using `backup_method = postgres` or `local-to-cloud` pointing to S3, indicates the AWS region where the S3 bucket is located.

> **Note**
>
> Only supported when `backup_method = snapshot` and `snapshot_provider = aws`, or when `backup_method` is `postgres` or `local-to-cloud` and `basebackups_directory` and/or `wals_directory` point to S3.

Scope: Global / Server / Model.

aws_check_object_lock

When set to `true`, Barman will check for S3 Object Lock before attempting to delete base backup files from S3 storage. If an object is locked, the deletion is aborted. Defaults to `false`.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` points to S3. If enabled with a non-S3 provider, a warning will be logged and no lock checks will be performed.

Scope: Global / Server / Model.

aws_encryption

Specifies the server-side encryption algorithm to use when storing objects in S3. This value is passed to the S3 client for all upload operations to cloud object storage.

Allowed options:

- `AES256`: Use S3-managed encryption keys (SSE-S3).
- `aws:kms`: Use AWS Key Management Service encryption keys (SSE-KMS).

If not set, S3's default encryption settings for the bucket will be used.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` and/or `wals_directory` point to S3.

Scope: Global / Server / Model.

aws_sse_kms_key_id

Specifies the AWS Key Management Service (KMS) key ID to use for server-side encryption of S3 objects. This option is only valid when `aws_encryption` is set to `aws:kms`.

Can be specified using the key ID on its own or using the full ARN for the key.

If not set when using `aws_encryption = aws:kms`, S3 will use the default AWS managed KMS key for S3.

Example:

```
[myserver]
backup_method = local-to-cloud
basebackups_directory = s3://my-bucket/barman/backups
aws_encryption = aws:kms
aws_sse_kms_key_id = arn:aws:kms:us-east-1:123456789012:key/abcd1234-5678-90ab-cdef-
↪EXAMPLEKEY
```

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` and/or `wals_directory` point to S3.

Scope: Global / Server / Model.

aws_read_timeout

Specifies the read timeout in seconds for S3 operations when using cloud storage. This value is passed to the S3 client for all commands that interact with backup data on cloud object storage. If not set, the default boto3 read timeout behavior applies.

This option is useful for advanced and non-standard S3 workflows where the default timeout may not be sufficient.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` and/or `wals_directory` point to S3.

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.

cloud_delete_batch_size

Specifies the number of files to delete in a single batch when performing delete operations on cloud storage. This option is relevant for both backup and WAL file deletions. The default value is `1000` for S3, `256` for Azure Blob, and `100` for GCS.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` and/or `wals_directory` point to cloud storage.

Scope: Global / Server / Model.

cloud_upload_max_archive_size

Specifies the maximum size in bytes for individual archive files during cloud uploads. When uploading backups to cloud storage, Barman splits the backup into multiple archive files if it exceeds this size. The default value is `100Gi`.

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.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` points to cloud storage.

Scope: Global / Server / Model.

cloud_upload_min_chunk_size

Specifies the minimum size in bytes for individual chunks during multipart uploads to cloud storage. This controls the size of each part when uploading large files to cloud object storage. When not set, defaults to the chunk size of the cloud provider: `5Mi` for S3, and `64Ki` for Azure Blob. Not applicable for GCS.

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.

> **Note**
>
> Only supported when `backup_method` is `postgres` or `local-to-cloud`, and `basebackups_directory` points to cloud storage.

cloud_staging_directory

A staging directory for when sending backups to a cloud object storage using `backup_method = postgres`. It is used as a temporary location for storing chunks of the backup before they are sent to the cloud. Defaults to `/tmp/barman/cloud-staging`.

Scope: Global / Server / Model.

cloud_staging_max_size

The maximum size that `cloud_staging_directory` can grow to before Barman stops generating new backup chunks. This is used to prevent the staging directory from growing in case the speed of uploading chunks does not keep up with the speed of streaming them from Postgres. The default value is `30Gi`.

Extremely low values are discouraged as they may lead to performance degradation. We recommend a minimum of at least `10Gi`. The optimal value will depend on the transfer speed of the backup to Barman and from Barman to the cloud, as well as the size of the backup.

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.

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

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.

15.2.4 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.

15.2.5 Write-Ahead Logs (WAL)

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

compression

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

Deprecated since version 3.16: The `pygzip` and `pybzip2` compression options are deprecated and will be removed in a future release. Use their equivalents `gzip` and `bzip2` instead.

Deprecated since version 3.17: The `custom` compression option and related configuration options (`custom_compression_filter`, `custom_decompression_filter`, and `custom_compression_magic`) are deprecated and will be removed in a future release. Use one of the built-in compression algorithms instead.

> **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.

Deprecated since version 3.17: This configuration option is deprecated and will be removed in a future release.

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.

Deprecated since version 3.17: This configuration option is deprecated and will be removed in a future release.

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.

Deprecated since version 3.17: This configuration option is deprecated and will be removed in a future release.

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`, `pygzip` 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.

cloud_wal_archive_parallel

Controls the number of WAL files that `barman cloud-wal-archive` will upload in parallel. When set to `N` (where N > 1), up to `N - 1` additional WAL files that are ready in `pg_wal/archive_status` are uploaded concurrently in background processes after the primary WAL has been successfully archived. Defaults to `0` (disabled).

> **Note**
>
> This option only applies when using the `local-to-cloud` backup method with `barman cloud-wal-archive` as the `archive_command`.

Scope: Global / Server.

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

Destination for archived WAL files. It can be a local path if storing WAL files locally, or a cloud storage URL if storing WAL files in the cloud as described in *Streaming backups to the cloud*. Defaults to `<backup_directory>/wals`.

> **Warning**
>
> Changing the value of `wals_directory` on an active server is discouraged. Even if done with care, by deactivating the server, moving existing data to the new location, and reactivating it, there are potential risks of data loss or misconfiguration.
>
> Instead, the safe option is to create a new server configuration with the new value for `wals_directory`, perform a new full backup to the new server, and then decommission the old server once the new one is fully operational.

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.

15.2.6 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`.

Deprecated since version 3.15: `local_staging_path` is deprecated and will be removed in a future release. Use `staging_path` and `staging_location` instead.

Scope: Global / Server / Model.

recovery_options

Options for recovery operations. Currently, only `get-wal` and `delta-restore` are supported. `get-wal` 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. `delta-restore` allows barman to restore a backup into a pre-existing destination directory along with existing custom tablespaces by overwriting it. 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.

Deprecated since version 3.15: `recovery_staging_path` is deprecated and will be removed in a future release. Use `staging_path` and `staging_location` instead.

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. The default is `/tmp`.

Scope: Global / Server / Model.

staging_location

Specifies whether `staging_path` is a local or remote path. Valid values are `local` and `remote`. The default is `local`.

Scope: Global / Server / Model.

combine_mode

Specifies a copy mode for `pg_combinebackup` when combining incremental backups during a restore.

Options include:

- `copy` (default): Use standard file copying when combining incremental backups.
- `link`: Use hard links when combining incremental backups. Reconstruction of the synthetic backup might be faster (no file copying) and use less disk space.
- `clone`: Use efficient file cloning (also known as "reflinks" on some systems) instead of copying files to the new data directory, which can result in near-instantaneous copying of the data files.

- `copy-file-range`: Use the `copy_file_range` system call for efficient copying. On some file systems this gives results similar to `clone`, sharing physical disk blocks, while on others it may still copy blocks, but do so via an optimized path.

Refer to the pg_combinebackup documentation for more details and restrictions of each mode.

Scope: Global / Server / Model.

15.2.7 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.

15.3 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.

15.3.1 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.

15.3.2 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.

15.3.3 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 Postgres 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.

15.3.4 Benefits

- Consistency: Ensures uniform configuration across multiple Postgres 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.

CHAPTER 16

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:

16.1 barman

16.1.1 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`.

16.1.2 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.

16.2 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 `rsync` or `postgres`.
last-failed	Most recent backup that failed, in chronological order.

16.3 Exit Statuses

Status code **0** means **success**, while status code **Non-Zero** means **failure**.

16.4 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.

16.4.1 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.

16.4.2 barman backup

Synopsis

```
backup
    [ --bwlimit KBPS ]
    [ --cloud-staging-directory PATH ]
    [ --cloud-staging-max-size SIZE ]
    [ { -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 PostreSQL 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.

--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.

--cloud-staging-directory
: A staging directory for when sending backups to a cloud object storage using `backup_method = postgres`. It is used as a temporary location for storing chunks of the backup before they are sent to the cloud. Defaults to `/tmp/barman/cloud-staging`.

--cloud-staging-max-size
: The maximum size that `--cloud-staging-directory` can grow to before Barman stops generating new backup chunks. This is used to prevent the staging directory from growing in case the speed of uploading chunks does not keep up with the speed of streaming them from Postgres. The default value is `30Gi`.

  Extremely low values are discouraged as they may lead to performance degradation. We recommend a minimum of at least `10Gi`. The optimal value will depend on the transfer speed of the backup to Barman and from Barman to the cloud, as well as the size of the backup.

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.

`-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.

16.4.3 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.

16.4.4 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.

16.4.5 barman cloud-wal-archive

Synopsis

```
cloud-wal-archive
    [ { -h | --help } ]
    [ { -p | --parallel } PARALLEL ]
    SERVER_NAME WAL_PATH
```

Description

Push a WAL file from the local disk to a configured cloud object storage. This command is intended to be used in the `archive_command` of a Postgres server when using the `local-to-cloud` backup method for taking base backups.

> **Note**
> The compression algorithm and level used for the WAL files can be configured in the Barman server configuration file. If not specified, the WAL file will be uploaded without compression.

Parameters

SERVER_NAME
The name of the Postgres server for which the WAL file is being archived.

WAL_PATH
The value of the `%p` keyword (according to `archive_command`).

-h / --help
Show a help message and exit. Provides information about command usage.

-p / --parallel
Number of WALs to archive in parallel. When set to a value greater than 1, up to `parallel - 1` additional WAL files that are ready in `pg_wal/archive_status` are uploaded concurrently in background processes after the primary WAL has been successfully archived. This can reduce WAL archival backlog during periods of high WAL generation. Defaults to the `cloud_wal_archive_parallel` server configuration value, which itself defaults to `0` (disabled).

16.4.6 barman cloud-wal-restore

Synopsis

```
cloud-wal-restore
    [ { -h | --help } ]
    [ { -p | --parallel } PARALLEL ]
    [ --spool-dir SPOOL_DIR ]
    SERVER_NAME WAL_NAME WAL_DEST
```

Description

Pull a WAL file from a configured cloud object storage to the local disk. This command is intended to be used as `restore_command` in Postgres when restoring WALs from a cloud object storage.

This command requires the server to have cloud storage configured via `wals_directory` pointing to a cloud URL (e.g., `s3://barman/wals`).

Parameters

SERVER_NAME
The name of the Postgres server for which the WAL file is being restored.

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
Show a help message and exit.

-p / --parallel
Specifies the number of WAL files to peek and download in parallel. When set to a value greater than 1, fetches the requested WAL file along with the next `N - 1` files simultaneously. The additional files are staged in the spool directory so that subsequent restore requests can be served immediately from local storage. Default is `0` (disabled).

--spool-dir
Directory used for staging extra WALs fetched when using `--parallel`. Default is `/var/tmp/walrestore`.

> **Tip**
>
> When using `--parallel` and both the `pg_wal` directory and the spool directory are on the same filesystem, WAL files are served faster because they are renamed rather than copied.

16.4.7 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.

16.4.8 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" }]'
```

16.4.9 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.

16.4.10 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.

16.4.11 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.

16.4.12 barman export-backup

Synopsis

```
export-backup
    [ { -h | --help } ]
    [ --compression { gzip | bzip2 | xz } ]
    [ --compression-level COMPRESSION_LEVEL ]
    SERVER_NAME BACKUP_ID OUTPUT_DIRECTORY
```

Description

Export the specified backup as a single tarball, suitable for hand-off to an external archival system. The tarball is self-contained: it bundles the backup data files, any tablespaces, the WAL segments required to bring the backup to a consistent state, and the Barman metadata needed to re-register the backup with commands-barman-import-backup later.

The generated file is named using the pattern `backup-export-SERVER_NAME-BACKUP_ID-TIMESTAMP-CHECKSUM.tar[.gz|.bz2|.xz]`. The filename is part of the integrity contract — do not rename the tarball, or commands-barman-import-backup will refuse it.

Only full backups in `DONE` state on a server using local storage can be exported. See *Backup export and import* for the full feature description.

Parameters

SERVER_NAME
Name of the server in barman node.

BACKUP_ID
Id of the backup in barman catalog.

OUTPUT_DIRECTORY
Existing directory where the exported tarball will be written. Must be writable and executable by the user running Barman.

`--compression {gzip | bzip2 | xz}`
Compress the outer tarball using the given algorithm. By default the tarball is not compressed. Compression applies only to the outer tarball; data files and WAL segments stored inside are not recompressed, and any pre-existing WAL compression is preserved through the associated `xlog.db` metadata.

`--compression-level COMPRESSION_LEVEL`
Compression level to pass to the chosen algorithm. Valid levels are `1-9` for `gzip` and `bzip2`, and `0-9` for `xz`. Defaults to the algorithm's own default level. Requires `--compression` to be set.

> **Note**
>
> Passing an explicit `--compression-level` requires Python 3.12 or newer for `gzip/bzip2` and Python 3.14 or newer for `xz`. On older interpreters the option is accepted, a warning is emitted, and the algorithm's default level is used instead.

16.4.13 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.

16.4.14 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 decompress 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.

16.4.15 barman import-backup

Synopsis

```
import-backup
    [ { -h | --help } ]
    SERVER_NAME INPUT_TARBALL
```

Description

Register a previously exported backup tarball back into the Barman catalog of `SERVER_NAME`. The tarball is expected to be the output of commands-barman-export-backup (or a file restored to disk verbatim by an external archival system from such an export). After a successful import, the backup is fully usable: it appears in commands-barman-list-backups, its metadata is available via commands-barman-show-backup, and it can be restored with commands-barman-restore.

Imported backups are automatically protected from retention by applying the `KEEP:STANDALONE` annotation (see commands-barman-keep). If you want the imported backup to participate in retention again, remove the annotation explicitly.

Import is performed atomically: the tarball is extracted into a staging directory under `basebackups_directory`, verified end-to-end, and only then moved into place. If any step fails, the staging area is cleaned up and the catalog and WAL archive are left untouched. See this section *Backup export and import* for the full feature description.

Parameters

SERVER_NAME
: Name of the server in barman node.

INPUT_TARBALL
: Path to the export tarball to import.

16.4.16 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`.

> **Note**
>
> Incremental backups are not eligible for the `keep` command. This restriction exists because incremental backups rely on their parent backups and cannot be used on their own for recovery. Keeping an incremental without its base would serve no purpose, as it is unusable independently.
>
> If you attempt to apply keep to an incremental backup, Barman will return an error indicating that incremental backups cannot be kept.

Only the full base backup —the one that starts the backup chain— can be marked with keep.

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.

16.4.17 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.

16.4.18 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.

16.4.19 barman list-processes

Synopsis

```
list-processes
    [ { -h | --help } ]
    SERVER_NAME
```

Description

The `list-processes` sub-command outputs all active subprocesses for a Postgres server. It displays the process identifier (PID) and the corresponding barman task for each active subprocess.

Parameters

SERVER_NAME
Name of the Postgres server for which to list active subprocesses.

-h / --help
Displays a help message and exits.

16.4.20 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.

16.4.21 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.

16.4.22 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.

16.4.23 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.

16.4.24 barman recover

> **Note**
> This command is deprecated. Use the commands-barman-restore command instead.

16.4.25 barman receive-wal

Synopsis

```
receive-wal
    [ --create-slot ]
    [ --if-not-exists ]
    [ --drop-slot ]
```

(continues on next page)

(continued from previous page)

```
    [ { -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.

--if-not-exists
Do not error out when `--create-slot` is specified and a slot with the specified name already exists.

--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.

16.4.26 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 ]
```

(continues on next page)

(continued from previous page)

```
    [ --local-staging-path PATH ]
    [ { --network-compression | --no-network-compression } ]
    [ --no-retry ]
    [ --partial-wal ]
    [ --recovery-conf-filename FILENAME ]
    [ --recovery-staging-path PATH ]
    [ --staging-path STAGING_PATH ]
    [ --staging-location STAGING_LOCATION ]
    [ --combine-mode COMBINE_MODE ]
    [ --remote-ssh-command STRING ]
    [ --restore-command COMMAND ]
    [ --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 PostreSQL 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.

--delta-restore / --no-delta-restore
Enable/disable usage of `delta-restore` for a delta recovery. Default is based on `recovery_options` setting.

--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.

> **Note**
> When `wals_directory` is configured to use cloud storage (e.g. S3, Azure, GCS), restores always require `get-wal` for WAL fetching. If you specify `--no-get-wal` or omit `get-wal` from `recovery_options`, Barman will automatically enable `get-wal` for the restore operation and issue a warning.

--partial-wal
Copy `.partial` WAL files to the recovery destination with the `.partial` suffix stripped. Barman considers two sources: `.partial` files in the `streaming_wals_directory` (written by `pg_receivewal`) and `.partial` files in the main WAL archive (archived during standby promotion, possibly compressed or encrypted). Only applicable when using `--no-get-wal`. When `--get-wal` is used, `.partial` files are already handled transparently and this option has no effect. The flag also has no effect when `--target-immediate` is used, since recovery stops at the first consistent state before any partial WAL would be needed. See *Partial WAL files during recovery* for details.

-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.

Deprecated since version 3.15: `--local-staging-path` is deprecated and will be removed in a future release. Use `--staging-path` and `--staging-location` instead.

--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 decompressed 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.

Deprecated since version 3.15: `--recovery-staging-path` is deprecated and will be removed in a future release. Use `--staging-path` and `--staging-location` instead.

--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`.

--combine-mode
Specifies a copy mode for `pg_combinebackup` when combining incremental backups during a restore.

Options include:

- `copy` (default): Use standard file copying when combining incremental backups.
- `link`: Use hard links when combining incremental backups. Reconstruction of the synthetic backup might be faster (no file copying) and use less disk space.
- `clone`: Use efficient file cloning (also known as "reflinks" on some systems) instead of copying files to the new data directory, which can result in near-instantaneous copying of the data files.
- `copy-file-range`: Use the `copy_file_range` system call for efficient copying. On some file systems this gives results similar to `clone`, sharing physical disk blocks, while on others it may still copy blocks, but do so via an optimized path.

Refer to the pg_combinebackup documentation for more details and restrictions of each mode.

--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'`.

--restore-command
Specify a custom restore command to override Barman's default restore_command in the PostgreSQL recovery configuration. While the default restore_command built by Barman is suitable for most cases, this option allows

you to specify an alternative command if your setup requires it. Can be used only when running a restore with `get-wal` mode.

`--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.

16.4.27 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.

16.4.28 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.

16.4.29 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.

16.4.30 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.

16.4.31 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.

16.4.32 barman switch-xlog

Description

Alias for the `switch-wal` command.

16.4.33 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.

16.4.34 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).

16.4.35 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.

16.4.36 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.

16.4.37 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.

16.4.38 barman verify

Description

Alias for `verify-backup` command.

16.5 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.

16.5.1 barman-wal-archive

Synopsis

```
barman-wal-archive
    [ { -h | --help } ]
    [ { -V | --version } ]
    [ { -U | --user } USER ]
    [ --port PORT ]
    [ { { -z | --gzip } | { -j | --bzip2 } | --xz | --snappy | --zstd | --lz4 } ]
    [ --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.

16.5.2 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
Specifies the number of WAL files to fetch in parallel. Default is `0` (disabled). When set to a value greater than `1`, fetches the requested WAL file along with the next `N - 1` files simultaneously. The additional files are staged in a local spool directory (see `--spool-dir`) so that subsequent restore requests can be served immediately from local storage.

--spool-dir
Directory used for staging extra WALs fetched when using `--parallel`. Default is `/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 trasnfered 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.

CHAPTER 17

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).

17.1 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.

17.2 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
```

17.3 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.

17.4 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>
```

CHAPTER 18

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`.

18.1 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.

18.2 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.

18.3 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.

18.4 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**.

18.5 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).

18.6 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`.

> **Warning**
>
> When using `barman-cloud-backup` as a hook script, it is not possible to set a backup name. It's also not possible to use it on backups with compression. Attempting to do so will result in an error.

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
```

CHAPTER 19

Barman for the cloud

Barman provides various ways for backing up Postgres clusters to the cloud:

- **Using a Barman server**

  Having a Barman server implies that you have installed and configured Barman via configuration files. In this case, all metadata about backups and WALs is stored locally and maintenance operations are performed periodically.

  In this mode you can:

  1. Set up a standard Barman server which stores backups and WALs locally. Then use the Barman *Hook Scripts* to copy the backups and WALs to a cloud object storage. If you choose this approach, please consult the *Using barman-cloud-* scripts as hooks in barman* section for further details.
  2. Set up a Barman server to store metadata and WAL files locally, while your backups are created as disk volume snapshots in the cloud. If you choose this approach, please consult the *cloud snapshots backups* section for details.
  3. Set up a Barman server which streams backups from the Postgres server and sends them directly to a cloud object storage, without ever storing the full backup locally. WAL files are streamed from the Postgres server and stored temporarily in the Barman server until successfully archived to the cloud. This is currently the only method that allows you to have incremental backups (block-level) in the cloud. If you choose this approach, please consult the *Streaming backups to the cloud* section for details.
  4. Set up Barman locally on the Postgres server, and use the `local-to-cloud` backup method to create backups directly in the cloud, without ever storing them locally. WAL files can be archived directly to the object store using `barman cloud-wal-archive` as the `archive_command`. If you choose this approach, please consult the backup-local-to-cloud section for details.

- **Using the cloud scripts, without the need of a Barman server**

  The Barman cloud scripts allow you to manage and interact with your backups directly via the command line utilities provided by the `barman-cli-cloud` package, without the need of a Barman server. In this case, data and metadata are always stored in the cloud object storage and Barman holds no local state.

  In this mode you can:

1. Use 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 metadata in the cloud object storage. Check the *Commands Reference* section below for details about the commands available in this mode.
2. Use the utility provided by the barman cloud client package in the Postgres host, without a Barman server. 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. Check the *Commands Reference* section below for details about the commands available in this mode, particularly with the `--snapshot-*` options.

This section of the documentation is focused on the cloud script commands, which 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.

Important

S3 Compatibility Statement

Barman Cloud utilizes the Boto3 SDK to integrate with S3-compatible object stores. While this ensures Barman Cloud functions correctly with the AWS S3 reference implementation and supports Barman Cloud integration with S3-compatible object stores, Barman Cloud does not directly support the underlying storage infrastructure of third-party vendors. If operational inconsistencies occur exclusively on third-party object stores which cannot be reproduced on AWS S3, this is considered API incompatibilities. Such issues must be resolved by the user in conjunction with their specific storage vendor.

19.1 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.

Note

Some S3 third-party providers are not compatible with the new Data Integrity Protection checks provided by default with `boto3>=1.36`. If your provider falls in this group, there is a workaround which may help you overcome the incompatibility with the AWS S3.

For this, you will have to set `when_required` to the following two environment variables shown below in order for the `barman-cloud` tools to work (there is no guarantee it will work, but we've seen that setting these make `barman-cloud` work in such environments):

```
AWS_REQUEST_CHECKSUM_CALCULATION=when_required
AWS_RESPONSE_CHECKSUM_VALIDATION=when_required
```

Important

Starting with AWS boto3 1.36, the behavior of **Data Integrity Protection checks** has changed. Some methods used by Barman no longer require the `Content-MD5` header.

This means that **S3-compatible storage providers that have not updated their server-side code may fail** when used with newer boto3 versions. For example, MinIO addressed this change shortly after the boto3 1.36 announcement.

If you are using MinIO, you **must upgrade** to the latest release (or at least `RELEASE.2025-02-03T21-03-04Z` or newer) to ensure compatibility and avoid failures when deleting backups or releasing a keep annotation.

19.2 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.

19.3 Object Lock Support

Barman cloud can work with object locking mechanisms that provide Write-Once-Read-Many (WORM) protection for backups stored in cloud object storage. Object locking helps ensure compliance with data retention requirements and prevents accidental or malicious deletion of backup files, protecting critical backup data from ransomware attacks or unauthorized deletions.

Object locking prevents objects from being deleted or overwritten for a fixed amount of time (retention period) or indefinitely (legal hold). This is particularly useful for regulatory compliance requirements.

Note

Object Lock verification is currently only supported for AWS S3 and S3-compatible storage providers.

Important

Barman does not create or manage object locks. It is the user's responsibility to configure the cloud storage bucket with the appropriate retention policies. Barman can verify and respect these locks during deletion operations to ensure compliance with the configured policies.

19.3.1 S3 Object Lock

Barman can work with AWS S3 Object Lock to help maintain backup immutability. To use this feature, you must configure your S3 bucket with the necessary Object Lock settings:

- **Object versioning enabled**: Required for Object Lock to function.
- **Default retention policy**: Configure a default retention policy for new objects in your bucket. This policy is managed entirely within AWS S3, not by Barman.

When these settings are configured in your S3 bucket, AWS automatically applies the retention policy to objects uploaded through `barman-cloud-backup` and `barman-cloud-wal-archive`. The objects are protected according to the bucket's retention configuration and cannot be deleted or modified until the retention period expires.

When deleting backups and WALs with `barman-cloud-backup-delete`, you can use the `--check-object-lock` option to verify whether objects are protected by Object Lock before attempting deletion. This flag instructs Barman to check the lock status of backup files and respect the configured retention policies, preventing deletion attempts that would fail and ensuring compliance with your data retention requirements.

For more information about AWS S3 Object Lock, refer to the AWS S3 Object Lock documentation.

See the barman-cloud-barman-cloud-backup-delete command reference for more details on how the `--check-object-lock` option works, including information about the deletion process and performance considerations.

19.4 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`.

19.4.1 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 | --lz4 } ]
                [ { -h | --host } HOST ]
                [ { -p | --port } PORT ]
                [ { -U | --user } USER ]
                [ { -d | --dbname } DBNAME ]
                [ { -n | --name } BACKUP_NAME ]
                [ { -J | --jobs } JOBS ]
                [ { -S | --max-archive-size } MAX_ARCHIVE_SIZE ]
                [ --immediate-checkpoint ]
```

(continues on next page)

(continued from previous page)

```
                [ --min-chunk-size MIN_CHUNK_SIZE ]
                [ --max-bandwidth MAX_BANDWIDTH ]
                [ --snapshot-instance SNAPSHOT_INSTANCE ]
                [ --snapshot-disk NAME [ --snapshot-disk NAME ... ] ]
                [ --snapshot-zone GCP_ZONE ]
                [ -snapshot-gcp-project GCP_PROJECT ]
                [ --tag KEY,VALUE [ --tag KEY,VALUE ... ] ]
                [ --endpoint-url ENDPOINT_URL ]
                [ --addressing-style { auto | virtual | path } ]
                [ { -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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

> **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).

--lz4
lz4-compress the backup while uploading to the cloud (requires optional `lz4` Python 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.

--tag
Tag to be added to all uploaded files in cloud storage, and/or to snapshots created, if snapshots are used.

--tags
Tags to be added to all uploaded files in cloud storage, and/or to snapshots created, if snapshots are used.

> **Note**
> If you are using `--tags` before positional arguments, you must insert `--` after it to indicate the end of optional arguments. This tells the parser to treat everything after `--` as positional arguments. Without the `--`, Barman may misinterpret positional arguments as values for the last option.

Deprecated since version 3.15: `--tags` is deprecated. Use `--tag` instead.

Extra options for the AWS cloud provider

--endpoint-url
Override default S3 endpoint URL with the given one.

--addressing-style
The addressing style to use for S3 requests. This is useful when connecting to S3-compatible services that require a specific addressing style.

Allowed options are:

- `auto` (default): Uses the addressing style determined by the underlying library.
- `virtual`: Uses virtual-hosted style addressing (e.g., bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (e.g., s3.amazonaws.com/bucket-name).

-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`.

19.4.2 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { -r | --retention-policy } RETENTION_POLICY ]
                [ { -m | --minimum-redundancy } MINIMUM_REDUNDANCY ]
                [ { -b | --backup-id } BACKUP_ID]
                [ --dry-run ]
                [ --check-object-lock ]
                [ { -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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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.

Important

Starting with AWS boto3 1.36, the behavior of **Data Integrity Protection checks** has changed. Some methods used by Barman no longer require the `Content-MD5` header.

This means that **S3-compatible storage providers that have not updated their server-side code may fail** when used with newer boto3 versions. For example, MinIO addressed this change shortly after the boto3 1.36 announcement.

If you are using MinIO, you **must upgrade** to the latest release (or at least `RELEASE.2025-02-03T21-03-04Z` or newer) to ensure compatibility and avoid failures.

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

--check-object-lock
Check whether backup files are protected by S3 Object Lock before attempting deletion. Enable this when your S3 bucket has Object Lock configured and you want to ensure deletion operations respect object-level retention policies. If any base backup file is locked (either by an active retention period or legal hold), the deletion is aborted with an error.

For performance reasons, lock checks are only performed for base backup files, since once the base backup is removed, the corresponding WAL files are no longer useful. Note that this check requires one request per file to check the lock status, which may increase deletion time for backups with many files. This feature is only available for AWS S3-compatible object stores and it also requires the `s3:GetObjectRetention` and `s3:GetObjectLegalHold` IAM permissions.

--endpoint-url
Override default S3 endpoint URL with the given one.

--addressing-style
The addressing style to use for S3 requests. This is useful when connecting to S3-compatible services that require a specific addressing style.

Allowed options are:

- `auto` (default): Uses the addressing style determined by the underlying library.
- `virtual`: Uses virtual-hosted style addressing (e.g., bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (e.g., s3.amazonaws.com/bucket-name).

-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.

19.4.3 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { --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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the GOOGLE_CLOUD_UNIVERSE_DOMAIN environment variable to the desired universe domain.

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).

--addressing-style
The S3 addressing style to use when connecting to S3-compatible cloud storage. This option is useful when using custom S3 endpoints (such as MinIO) that may require a specific addressing style.

Allowed options:

- `auto` (default): Uses the addressing style determined by the boto3 library.
- `virtual`: Uses virtual-hosted-style addressing (bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (s3.amazonaws.com/bucket-name).

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`.

19.4.4 barman-cloud-backup-list

Synopsis

```
barman-cloud-backup-list
                [ { -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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { --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.

Backups in any state are included in the listing. Completed backups (status `DONE`) show their end time and, where applicable, their archival keep target (e.g. `KEEP:FULL`). In-progress or failed backups show their status (e.g. `STARTED`) in the *Archival Status* column and an empty *End Time*.

> **Note**
>
> For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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).

--addressing-style
The S3 addressing style to use when connecting to S3-compatible cloud storage. This option is useful when using custom S3 endpoints (such as MinIO) that may require a specific addressing style.

Allowed options:

- `auto` (default): Uses the addressing style determined by the boto3 library.
- `virtual`: Uses virtual-hosted-style addressing (bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (s3.amazonaws.com/bucket-name).

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.

19.4.5 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { --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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

> **Important**
>
> Starting with AWS boto3 1.36, the behavior of **Data Integrity Protection checks** has changed. Some methods used by Barman no longer require the `Content-MD5` header.
>
> This means that **S3-compatible storage providers that have not updated their server-side code may fail** when used with newer boto3 versions. For example, MinIO addressed this change shortly after the boto3 1.36 announcement.
>
> If you are using MinIO, you **must upgrade** to the latest release (or at least `RELEASE.2025-02-03T21-03-04Z` or newer) to ensure compatibility and avoid failures when releasing a keep annotation.

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).

--addressing-style
The S3 addressing style to use when connecting to S3-compatible cloud storage. This option is useful when using custom S3 endpoints (such as MinIO) that may require a specific addressing style.

Allowed options:

- `auto` (default): Uses the addressing style determined by the boto3 library.

- `virtual`: Uses virtual-hosted-style addressing (bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (s3.amazonaws.com/bucket-name).

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`.

19.4.6 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { --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**
>
> The `barman-cloud-check-wal-archive` command performs an initial `HeadBucket` call to verify whether the target bucket already exists in the S3 storage. If the bucket does not exist, the command will attempt to automatically create it.
>
> This is the only Barman command that performs these operations (bucket existence check and automatic creation).

> **Note**
>
> For GCP, only authentication with `GOOGLE_APPLICATION_CREDENTIALS` env is supported. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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).

`--addressing-style`
The S3 addressing style to use when connecting to S3-compatible cloud storage. This option is useful when using custom S3 endpoints (such as MinIO) that may require a specific addressing style.

Allowed options:

- `auto` (default): Uses the addressing style determined by the boto3 library.
- `virtual`: Uses virtual-hosted-style addressing (bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (s3.amazonaws.com/bucket-name).

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`.

19.4.7 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { -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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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.

`--addressing-style`
The addressing style to use for S3 requests. This is useful when connecting to S3-compatible services that require a specific addressing style.

Allowed options are:

- `auto` (default): Uses the addressing style determined by the underlying library.
- `virtual`: Uses virtual-hosted style addressing (e.g., bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (e.g., s3.amazonaws.com/bucket-name).

`-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`.

19.4.8 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 ]
                [ --tag KEY,VALUE [ --tag KEY,VALUE ... ] ]
                [ --history-tag KEY,VALUE [ --history-tag KEY,VALUE ... ] ]
                [ --endpoint-url ENDPOINT_URL ]
                [ --addressing-style { auto | virtual | path } ]
                [ { -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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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*.

--tag
Tag to be added to archived WAL files in cloud storage.

--tags
Tag to be added to archived WAL files in cloud storage.

> **Note**
>
> If you are using `--tags` before positional arguments, you must insert `--` after it to indicate the end of optional arguments. This tells the parser to treat everything after `--` as positional arguments. Without the `--`, Barman may misinterpret positional arguments as values for the last option.

Deprecated since version 3.15: `--tags` is deprecated. Use `--tag` instead.

`--history-tag`
Tag to be added to archived history files in cloud storage.

`--history-tags`
Tags to be added to archived history files in cloud storage.

> **Note**
>
> If you are using `--history-tags` before positional arguments, you must insert `--` after it to indicate the end of optional arguments. This tells the parser to treat everything after `--` as positional arguments. Without the `--`, Barman may misinterpret positional arguments as values for the last option.

Deprecated since version 3.15: `--history-tags` is deprecated. Use `--history-tag` instead.

Extra options for the AWS cloud provider

`--endpoint-url`
Override default S3 endpoint URL with the given one.

`--addressing-style`
The addressing style to use for S3 requests. This is useful when connecting to S3-compatible services that require a specific addressing style.

Allowed options are:

- `auto` (default): Uses the addressing style determined by the underlying library.
- `virtual`: Uses virtual-hosted style addressing (e.g., bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (e.g., s3.amazonaws.com/bucket-name).

`-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.

19.4.9 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 ]
                [ --addressing-style { auto | virtual | path } ]
                [ { -P | --aws-profile } AWS_PROFILE ]
                [ --profile AWS_PROFILE ]
                [ --read-timeout READ_TIMEOUT ]
                [ { --azure-credential | --credential } { azure-cli | managed-identity
                  | default } ]
                [ --no-partial ]
                [ { -p, --parallel } PARALLEL ]
                [ --spool-dir SPOOL_DIR ]
                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.

> **Note**
>
> Partial WAL files (with a `.partial` suffix) can appear in the object store when a standby server is archiving WAL segments and is then promoted to primary. At the moment of promotion, the in-progress WAL segment has not been completed yet, so Postgres archives it as a `.partial` file before switching to a new timeline. These files represent an incomplete segment and are not suitable for crash recovery on their own.
>
> In most cases, downloading `.partial` files during WAL restore is harmless, because Postgres knows how to handle them correctly. However, when running `barman-cloud-wal-restore` on a cluster that has experienced a promotion, the `restore_command` may retrieve a `.partial` (after renaming it without the suffix) file where a complete WAL is expected.
>
> Use `--no-partial` to suppress the download of `.partial` files in this scenario, which will cause `restore_command` to signal to Postgres that the WAL is not available, prompting it to switch to the correct timeline via a timeline history file (`.history`) instead.

> **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. To use an alternative GCP universe (e.g. S3NS/T-Systems), set the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable to the desired universe domain.

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

-p / --parallel
Specifies the number of WAL files to fetch in parallel. Default is `0` (disabled). When set to a value greater than `1`, fetches the requested WAL file along with the next `N - 1` files simultaneously. The additional files are staged in a local spool directory (see `--spool-dir`) so that subsequent restore requests can be served immediately from local storage.

--spool-dir
Directory used for staging extra WALs fetched when using `--parallel`. Default is `/var/tmp/walrestore`.

Extra options for the AWS cloud provider

--endpoint-url
Override default S3 endpoint URL with the given one.

--addressing-style
The addressing style to use for S3 requests. This is useful when connecting to S3-compatible services that require a specific addressing style.

Allowed options are:

- `auto` (default): Uses the addressing style determined by the underlying library.
- `virtual`: Uses virtual-hosted style addressing (e.g., bucket-name.s3.amazonaws.com).
- `path`: Uses path-style addressing (e.g., s3.amazonaws.com/bucket-name).

-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`.

CHAPTER 20

FAQ

20.1 General

Can Barman perform physical backups of Postgres instances?

Yes. Barman is an application for physical backups of Postgres servers that manages base backups and WAL archiving. It is a disaster recovery application. Barman does not support logical backups (aka dumps).

I am already using pg_dump. What is the difference between pg_dump and Barman and why should I use Barman instead?

If you already use `pg_dump`, it is a good thing. However, if your business is based on your database, logical backups (the ones performed by `pg_dump`) are not enough. These dumps are snapshots of your database at a particular point in time. Usually, people perform these activities at night. If a crash occurs during the day, all your transactions between when the dump started and the crash will be lost forever. In this context, you need to put in place a more robust solution for disaster recovery, based on physical backups, which allows for point in time recovery.

I manage several Postgres instances. Can Barman manage multiple database servers?

Yes. Barman has been designed to allow remote backups of Postgres servers. It allows *DBAs* to manage backups of multiple servers from a centralized host. Barman allows you to define a catalogue of backup servers for base backups and continuous archiving of WAL segments.

Does Barman manage replication and high availability as well? How does it compare with repmgr, OmniPITR, walmgr and similar tools?

No. Barman aims to be a pure disaster recovery solution. It is responsible for the sole backup of a cluster of Postgres servers. If high availability is what you are looking for, we encourage you to use a tool like Patroni or repmgr. Barman specifically targets disaster recovery only, because it requires a simpler and less invasive design than the one required to cover high availability. This is why Barman does not duplicate existing HA tools like the ones mentioned above. You do not need to install anything on the server. The only requirement is to configure access for the Barman client, which is less invasive than what is required for high availability.

Can I define retention policies for my backups?

Yes. Barman 1.2.0 introduced support for retention policies both for base backups and WAL segments, for every server you have. You can specify a retention policy in the server configuration file. Retention policies can be based on time

(e.g. `RECOVERY WINDOW OF 30 DAYS`) or number of base backups (e.g. `REDUNDANCY 3`).

Does Barman guarantee data protection and security?

Barman communicates with your remote/local Postgres server using SSH connections (for process and file management) and the standard Postgres connection (for querying the database). It is the system administrator's duty to make sure that SSH communications occur in a secure way. Similarly, it is the database administrator's duty to make sure communications with the database server occur in a secure way.

Is the Barman interface complex or hard to understand?

Barman has a simple console interface through which you can run its several commands e.g. listing servers, listing backups, show detailed information of a backup, etc. Launching a new base backup for a server is also trivial, as well as restoring it, either locally or remotely.

Can Barman manage backups in the cloud e.g. using AWS S3 or Azure Blob Storage?

Yes. Barman can currently manage backups in Amazon, Azure and Google cloud providers. This backup method is called snapshot backups in Barman terminology. In this scenario, backups are taken via a snapshot of the storage volume where your database server resides. In this case, Barman act mainly as a storage server for your WAL files and backups catalog. You can check the *Barman Cloud* and *Cloud Snapshot Backups* sections for further details.

Do you have packages for RedHat and Debian based distributions?

Barman official packages are provided by *PGDG*. Barman packages can be found in alternative repositories, but we recommend using *PGDG* repositories because it ensures compatibility, stability and access to the latest updates. Refer to the *installation* section for further details.

Does Barman allow me to limit the bandwidth usage for backup and recovery?

Yes. Barman 1.2.1 introduces support for bandwidth limitation at global, server and tablespace level.

20.2 Backup

Does Barman support incremental backups

Yes. Barman currently supports file-level and block-level incremental backups, depending on the backup method in use and the Postgres version to backup.

How many backups can Barman handle for each Postgres instance?

Barman allows the storage of multiple base backups per database server. The only limit on the number of backups is the disk space on the Barman host.

I have continuous archiving in place, but managing WAL files and understanding which ones "belong" to a particular base backup is not obvious. Can Barman simplify this?

Fortunately, yes. The way Barman works is by keeping separate the base backups from WAL segments of a specific server. Even though they are much related, Barman sees a WAL archive as a continuous stream of data from the first base backup to the last available. A neat feature of Barman is to link every WAL file to a base backup so that it is possible to determine the size of a backup in terms of two components: base backup and WAL segments.

Can Barman compress base backups?

Currently, Barman can compress backups using `backup_method = postgres`, thanks to `pg_basebackup` compression features. This can be enabled using the `backup_compression` config option. For Rsync-based backups, at the moment there is no compression method, but it is feasible and the current design allows it. You can check the *Backup Compression* section for further details.

Can Barman compress WAL segments?

Yes. You can specify a compression filter for WAL segments, and significantly reduce the size of your WAL files by 5/10 times. This is done by setting the `compression` option in the configurations. Refer to the *WAL configuration* section for further details.

Can Barman back up tablespaces?

Yes. Tablespaces are handled transparently and automatically by Barman.

Can I backup from a Postgres standby server?

Yes, Barman natively supports backup from standby servers for both `postgres` and `rsync` backup methods.

What's the difference between Full and Incremental backups when using the rsync backup method in Barman?

With the `rsync` backup method, the on-disk backup has no clear distinction between a full and an incremental backup. In practice, all backups created with `rsync` are full backups, but they may share common files using hard-links, which reduces storage space and speeds up backup creation.

When using `rsync` with `reuse_backup = link`, files that are exactly the same since the last backup are not copied again; instead, hard links to the existing files are created. This makes the backups appear to full as all files are completely available in the backup folder, yet the real size used on-disk and transfered is less because unchanged files are linked rather than duplicated. For this reason, each rsync backup is a full "snapshot", independent of previous backups, and deleting any of the backups would not alter any of the others.

> **Note**
>
> A hard link is a pointer to the actual data of a file stored on the disk. When a hard link is created, it points to the same underlying data blocks on the storage device, so the same data can be accessed via multiple file names.

In comparison, with the `backup_method = postgres` method (Postgres 17+), incremental backups depend on a chain of backups, and restoring an incremental backup requires combining it with its full backup and any intermediate incremental backups. In This case, deleting an incremental backup would invalidate the following incremental backups.

To summarize, while rsync backups are file-level incremental in that they avoid duplicating unchanged files, each backup remains a full "snapshot", independent of previous ones.

20.3 Installation & Configuration

Does Barman have to be installed on the same server as Postgres?

No. Barman does not necessarily need to be on the same host where Postgres is running. It is your choice to install it locally or on another server (usually dedicated for backup purposes). We strongly recommend having a dedicated server for Barman.

Can I have multiple configurations for different users in a Barman server?

Yes. Barman needs a configuration file. You can have a system wide configuration (`/etc/barman.conf`) or a user configuration (`~/.barman.conf`). For instance, you could set up several users in your system that use Barman, each of those working on a subset of your managed Postgres servers. This way you can protect your backups on a user basis.

20.4 Recovery

Does Barman manage recovery?

Yes. With Barman, you can recover a Postgres instance on your backup server or a remote node. Recovering remotely is just a matter of specifying an SSH command in the `recover` command, which Barman will use to connect to the destination host in order to restore the backup.

Does Barman manage recovery to a specific transaction or to a specific time?

Yes. Barman allows you to perform point-in-time recovery by specifying a timestamp, transaction ID, a Log Sequence Number (LSN) or a named restore point created previously. It is just a matter of adding an extra option to your `recover` command. You can refer to *Point-in-Time Recovery* under Recovery section for further details.

Does Barman support timelines?

Yes, Barman handles Postgres timelines for recovery.

Does Barman handle tablespaces and their mapping during recovery operations?

Yes. By default, tablespaces are restored to the same path they had on the source server. You can remap them as you wish by specifying the `--tablespace` option in your `recover` command.

During recovery, does Barman allow me to relocate the PGDATA directory? What about tablespaces?

Yes. When recovering a server, you can specify different locations for your `PGDATA` directory and all your tablespaces, if any. This allows you to set up temporary sandbox servers. This is particularly useful in cases where you want to recover a table that you have unintentionally dropped from the master by dumping the table from the sandbox server and then recreating it in your master server.

20.5 Requirements

Does Barman work on Windows?

Barman can take backups of your Postgres servers on Windows. The recovery part is not supported. Additionally, Barman will have to run on a UNIX box.

CHAPTER 21

License

Barman is owned by EnterpriseDB UK Limited and is licensed under the GNU General Public License v3.

Barman has initially received partial funding from 4CaaSt, a research project supported by the European Commission's Seventh Framework Programme.

We welcome contributions to Barman. These contributions will be acknowledged in the release notes. To ensure that all Barman code remains free, EnterpriseDB UK Limited requires contributors to complete a copyright assignment and to disclaim any work-for-hire claims from their employers. For a Copyright Assignment Form, please contact barman@enterprisedb.com.

© Copyright EnterpriseDB UK Limited 2011-2025

CHAPTER 22

Barman release notes

© Copyright EnterpriseDB UK Limited 2025 - All rights reserved.

22.1 3.19.0 (2026-05-20)

22.1.1 Notable changes

- Add support for restoring backups from cloud storage

  Barman can now restore backups taken with `backup_method = local-to-cloud` or `backup_method = postgres` when stored in cloud object storage.

  The standard `barman restore` command can be used to restore such backups.

  For WAL restore, a new `barman cloud-wal-restore` command was introduced to fetch WAL files directly from cloud storage during recovery. This command can be used as the `restore_command` in PostgreSQL and supports parallel fetching for improved performance.

  Also, when a server has `wals_directory` configured to use cloud storage, `get-wal` is now always required for restore operations. If `--no-get-wal` is specified or `get-wal` is absent from `recovery_options`, Barman will automatically enable `get-wal` and issue a warning so the restore can proceed without interruption.

  This completes the cloud backup lifecycle, enabling seamless backup and recovery workflows entirely in the cloud.

  References: BAR-929.

- Add new barman export-backup command for exporting backups to tarball

  Introduced a new `barman export-backup` command that exports a backup to a tarball format. The command takes a server name, backup ID, and destination directory as arguments, and creates a tarball containing the backup data. This provides an easy way to share or archive backups.

  References: BAR-1180.

- Add the `barman import-backup` command for importing backups from tarball

Introduced a new `barman import-backup` command that imports a backup from a tarball format. The command takes a server name and tarball path as arguments, and provides a way to register previously exported backups back into a Barman catalog.

References: BAR-1223.

22.1.2 Minor changes

- Support alternative GCP universes via GOOGLE_CLOUD_UNIVERSE_DOMAIN environment variable

  Barman now supports alternative GCP universes (e.g. S3NS/T-Systems) by reading the `GOOGLE_CLOUD_UNIVERSE_DOMAIN` environment variable. When set, the value is passed as `client_options` to the GCS storage client, allowing `barman-cloud-*` commands to operate against non-standard GCP endpoints. This feature was contributed by community member @mgarstecki.

  References: BAR-1017.

- Add support for parallel WAL restore in `barman-cloud-wal-restore`

  Support for parallelism has been added to the `barman-cloud-wal-restore` command, allowing users to specify a number of WALs to be downloaded in parallel during the recovery of WALs. This enhancement improves the overall performance of WAL restoration by prefetching multiple WAL files concurrently. Check the documentation of `barman-cloud-wal-restore` for more details.

  References: BAR-1253.

- Add aws_check_object_lock config option to check S3 object lock before deletion

  Introduced a new `aws_check_object_lock` configuration option. When enabled, Barman checks for S3 Object Lock before attempting to delete base backup files from S3 storage. If an object is locked, the deletion is aborted. This aligns Barman' s behavior with `barman-cloud-delete` for locked objects. Only applies to S3 storage; a warning is logged if this option is enabled with a non-S3 cloud provider.

  References: BAR-1113.

- Make in-progress cloud backups visible in the backup catalog

  In-progress cloud backups are now immediately visible in `barman-cloud-backup-list` with `STARTED` in the archival status column and an empty end time. Previously, a backup only became visible once it completed successfully. `barman-cloud-backup-delete`, `barman-cloud-restore`, and the `last/latest/first/oldest` shortcuts all treat in-progress backups safely by excluding them from redundancy counts, auto-restore selection, and shortcut resolution respectively.

  References: BAR-1235.

- Removed setuptools as a runtime dependency

  Setuptools is no longer required as a runtime dependency for Barman. Setuptools was previously included to provide the deprecated distutils module for Python 3.12+ compatibility, but it has been replaced with another implementation. Setuptools remains available during build time for packaging purposes.

  References: BAR-1170.

- Add –partial-wal flag to barman restore for opt-in .partial WAL file copying

  A new `--partial-wal` flag has been added to `barman restore --no-get-wal`. When set, Barman includes `.partial` WAL files in the recovery, stripping the `.partial` suffix so that PostgreSQL can find them during replay. Two sources are supported:

  - The streaming WALs directory: the in-progress segment being written by `pg_receivewal` at the time of the restore. Including it reduces RPO by making any transactions written since the last completed WAL segment available during recovery.

  - The WAL archive (`wals/`): segments archived with the `.partial` suffix, e.g. when a standby is promoted and PostgreSQL calls `archive_command` with an in-progress segment. Compressed and encrypted files are handled transparently.

  The flag has no effect when `--get-wal` is used, as `.partial` files are already handled transparently in that case.

  References: BAR-695.

- Add parallel WAL archiving support to `barman cloud-wal-archive`

  `barman cloud-wal-archive` now supports a `--parallel` flag (and the corresponding `cloud_wal_archive_parallel` configuration option) that enables opportunistic prefetching of additional WAL files during archival. When `--parallel N` is set (N > 1), up to N - 1 extra WAL files that are ready in `pg_wal/archive_status` are identified and uploaded concurrently in background worker processes after the primary WAL has been successfully archived. This can significantly reduce WAL archival backlog during periods of high WAL generation. If a WAL file was already uploaded by a previous invocation, subsequent calls to `archive_command` for that WAL will detect it and exit immediately with 0, avoiding redundant uploads.

  References: BAR-1101.

22.1.3 Bugfixes

- Fix backup metadata not being updated in the cloud storage

  NOTE: This fix is for the cloud integration of Barman servers i.e. those configured with `backup_method = local-to-cloud` or `backup_method = postgres` which uses a cloud storage as backup destination. This is not an issue in the `barman-cloud-*` scripts (e.g. barman-cloud-backup).

  Backup metadata (backup.info files) was not being updated in the cloud storage, which led, for example, to the status of the backup in the cloud being outdated compared to the local one. This can be a problem during the restore phase, where the metadata from the cloud might be the primary source of information. This issue is now fixed.

  References: BAR-1227.

- Fix `barman restore --no-get-wal` copying unusable .partial WAL files

  When running `barman restore --no-get-wal`, `.partial` WAL files present in the WAL archive were copied to the recovery destination. However, the `restore_command` generated by Barman uses `cp` from the `barman_wal` directory, which does not account for `.partial` suffixes — so these files were never used by PostgreSQL during recovery. Barman now skips `.partial` files by default. To explicitly include them (with the suffix stripped so PostgreSQL can find them), use the new `--partial-wal` flag.

  References: BAR-1265.

- Fix `--no-get-wal` restore not copying WALs from newer timelines on PostgreSQL 12+

  Since PostgreSQL 12, the default for `recovery_target_timeline` changed from `current` to `latest`. When restoring with PITR options (e.g., `--target-time`) in `--no-get-wal` mode, Barman was only copying WAL files from the backup's original timeline. This could cause restores to fail when the recovery target is on a newer timeline.

  Barman now correctly copies WAL files from all timelines up to the latest available when performing PITR on PostgreSQL 12+, matching the default Postgres recovery behavior.

  References: BAR-1054.

- Fix multiple warnings about intermediary files not being removed

  When using WAL compression, Barman creates intermediary files in the archive directory during the compression process. Intermediary file paths are stored in a list and removed at the end of the file archival.

However, when archiving multiple files in batch, the list of intermediary files was not being cleared after each processed file, causing later archivals to attempt to remove files that no longer exist, resulting in warnings like the below:

```
WARNING: Could not remove intermediary file /home/gustavo/pg/streaming/
↪000000010000001300000FD: [Errno 2] No such file or directory: '/home/gustavo/pg/
↪streaming/000000010000001300000FD'
```

This bug was introduced in version 3.18, commit 5db8ca9b. It is now fixed and such warnings should no longer occur.

References: BAR-1228, CP-59700.

22.2 3.18.0 (2026-03-12)

22.2.1 Notable changes

- Add support for incremental backups in cloud storages

  Added support for block-level incremental backups in cloud storages, allowing users to efficiently manage backup storage and reduce costs by only uploading changed data.

  This new feature relies on streaming native Postgres backups (taken with `pg_basebackup`) to Barman and from Barman directly to a cloud storage dynamically, without ever storing the full backup in the Barman host, only using a small staging area. The staging area location and size can be configured via the new `cloud_staging_directory` and `cloud_staging_max_size` configuration options, respectively.

  To configure this feature, set `backup_method = postgres` and `basebackups_directory` and `wals_directory` to a cloud storage URL (e.g., `s3://my-bucket/barman`).

  Example configuration:

```
[myserver]
backup_method = postgres
basebackups_directory = s3://my-bucket/barman
wals_directory = s3://my-bucket/barman
cloud_staging_directory = /tmp/barman/cloud-staging
cloud_staging_max_size = 30Gi
```

  The backup structure in the cloud storage follows the same patterns as the current barman-cloud-* scripts i.e. backup files are archived with `tar` and live under directories named after the server name and the backup ID, e.g. `my-bucket/barman/<server_name>/base/<backup_id>/...`, while WAL files are stored in a `wals` subdirectory under the server directory, i.e. `my-bucket/barman/<server_name>/wals/...`. In this sense, `basebackups_directory` and `wals_directory` act as a prefix for the backup and WAL paths in the cloud storage. This ensures compatibility with existing backup management practices.

  Important notes:

  - This is an experimental feature, and there are some limitations to be aware of (see the documentation for details).
  - Restoring backups taken with this method is currently not supported directly in Barman, and it's the user's responsibility to perform this manually or through custom scripts/processes. Restoring such backups will come in a future release.
  - This is the first step toward unifying the barman cloud experience and simplifying cloud backup operations, and more features and improvements will be added in future releases.

References: BAR-928.

- Add `local-to-cloud` backup method to `barman` CLI

  Introduced support for direct cloud backups through the main `barman` CLI by adding a new `local-to-cloud` backup method. Previously, cloud backups were only available through standalone `barman-cloud-*` scripts.

  With this enhancement, you can now configure cloud backups directly in the barman configuration file using `backup_method = local-to-cloud` along with a cloud storage URL in `basebackups_directory` (e.g., `s3://bucket/path`). This provides a unified interface for all backup methods and is the first step toward deprecating the standalone `barman-cloud-*` scripts.

  The new `CloudBackupExecutor` integrates seamlessly with the existing barman backup infrastructure, coordinating with PostgreSQL using the low-level backup API and uploading backups directly to cloud object storage.

  Example configuration:

```
[myserver]
backup_method = local-to-cloud
basebackups_directory = s3://my-bucket/barman/backups
conninfo = host=postgres-host dbname=postgres
```

  This is the first step toward unifying the barman cloud experience and simplifying cloud backup operations.

  The following configuration options were reused or added to the Barman configuration to provide feature parity with the `barman-cloud` scripts:

  - `cloud_upload_min_chunk_size`
  - `aws_region`
  - `aws_read_timeout`
  - `aws_encryption`
  - `aws_sse_kms_key_id`
  - `bandwidth_limit`
  - `cloud_delete_batch_size`
  - `cloud_upload_max_archive_size`

  References: BAR-1095, BAR-1116, BAR-1122, BAR-1109, BAR-1120, BAR-1121, BAR-1117, BAR-1112, BAR-1115.

- Add 'barman cloud-wal-archive' command for local-to-cloud WAL archiving

  Introduced a new `barman cloud-wal-archive` command that enables direct WAL file archiving from the PostgreSQL `pg_wal` directory to cloud object storage when using the 'local-to-cloud' backup method. This command is designed to be used as the PostgreSQL `archive_command` when the Barman server runs on the same host as the PostgreSQL instance.

  WAL files sent to cloud object storage can also be compressed before uploading, using the existing `compression` and `compression_level` configuration options. Compressed WAL files are stored with the appropriate file extension (e.g. `.gz`, `.lz4`), consistent with the convention already used by `barman-cloud-wal-archive`.

  The following in-memory compression algorithms are supported: `gzip`, `bzip2`, `xz`, `snappy`, `zstd`, and `lz4`. `pigz` and `custom` are not supported for cloud WAL storage because they rely on external processes.

  Example configuration:

```
[myserver]
backup_method = local-to-cloud
archiver = on
wals_directory = s3://my-bucket/barman/wals
compression = gzip
compression_level = 6
```

References: BAR-1098, BAR-1124.

22.2.2 Minor changes

- Add `--addressing-style` option for S3-compatible storage providers

  A new `--addressing-style` option is now available for barman-cloud commands when using AWS S3 or S3-compatible storage providers. This option allows users to explicitly set the S3 addressing style to `auto`, `virtual`, or `path`. This is useful when working with S3-compatible storage systems that require virtual-hosted-style addressing, as `boto3` automatically defaults to path-style addressing for non-S3 endpoints.

  Special thanks to @ChandonPierre who reported and contributed to this feature in the PR #1134.

  References: BAR-1011.

- Add Python 3.14 compatibility support for cloud operations

  Added support for Python 3.14 by fixing a breaking change in `functools.partial` that affected cloud backup operations. In this version, `functools.partial` became a method descriptor which changed its behavior when used as a class attribute. This update ensures Barman cloud operations continues to work seamlessly when upgrading to Python 3.14.

  This patch is based on the original idea by @pvbiesen, who suggested the feature and contributed in the PR #1143. A similar fix was also suggested by @mjlshen in PR #1109.

  References: BAR-1012.

- Prevent usage of barman-cloud-backup as hook for backups with compression

  The usage of `barman-cloud-backup` as a hook for backups taken with compression is now prevented by erroring out with an appropriate message.

  This change ensures that backups with compression are not inadvertently used with cloud backup hooks, given that such usage was never supported.

  The behavior is also documented with a warning note.

  References: BAR-1015.

- Add LZ4 compression support for Barman cloud

  Added support for LZ4 compression in `barman-cloud-backup` command. You can now specify the –lz4 option to compress backup files using LZ4 compression before uploading to cloud storage. This option is mutually exclusive with other compression options (–gzip, –bzip2, –snappy), and it requires the `lz4` Python library to be installed.

  This feature was suggested and implemented by @vietcgi in the PR #1144.

  References: BAR-1013.

- Experimental support for WarehousePG database ID configuration

  Addition of experimental support for WarehousePG, which includes a new configuration option `warehousepg_dbid` that allows users to specify the database ID of the WarehousePG segment when using the

PostgreSQL backup method. This configuration is intended to enable better integration with WarehousePG for backup and restore operations.

There's also an improvement in the collection of Postgres parameters due to `archive_timeout` not being exposed in `pg_settings` on WarehousePG segments.

References: BAR-1092.

- Add custom restore command support for barman recover operations

  Added a new `--restore-command` option to the `barman recover` command that allows users to override Barman's default restore command with a custom command. While the default restore command built by Barman is suitable for most cases, this option allows you to specify an alternative command if your setup requires it.

  Previously, users had to manually edit PostgreSQL recovery configuration files after recovery to change the restore command, which was error-prone and required additional manual intervention. With this new option, users can now specify a custom restore command directly through the command line.

  This enhancement was applied due to an issue reported in #881 by @PaganMuffin.

  References: BAR-201.

22.2.3 Bugfixes

- Fix file handle leak for long-running WAL archiving services

  Fixed a resource leak in the cloud WAL archiving functionality where file handles were not being properly closed after uploading WAL files to cloud storage. This issue could primarily affect long-running WAL archiving services that process multiple WAL files in sequence, where the accumulating open file handles could eventually cause "Too many open files" errors, preventing further WAL archiving operations.

  The fix ensures that file handles are properly closed immediately after each upload operation completes, regardless of whether compression is used and even when upload failures occur. This prevents handle exhaustion in services that perform continuous or high-frequency WAL archiving to cloud providers.

  Special thanks to @ethiebautgeorge-nasuni who contributed to this feature in the PR #1129.

  References: BAR-973.

22.3 3.17.0 (2026-01-07)

22.3.1 Notable changes

- Enable query, fetch, and selective operations on inactive/disabled servers

  Multiple Barman commands now work with inactive and disabled servers based on their safety profile:

  Query commands:

  - `list-backups`: view backups from inactive/disabled servers
  - `status`: check status of inactive/disabled servers
  - `show-backup`: display backup details from inactive/disabled servers
  - `list-files`: list files from backups on inactive/disabled servers
  - `verify-backup`: verify backups on inactive/disabled servers

  Fetch commands:

  - `restore`: restore backups from inactive servers
  - `get-wal`: retrieve WAL files from inactive servers

Modification commands:

- `delete`: delete backups from inactive servers
- `config-switch`: switch configuration models for inactive/disabled servers

This change improves operational flexibility by allowing users to query backup information and perform restore operations even when servers are manually disabled (inactive) or have configuration issues (disabled). The different behavior for disabled servers provides safety against operations on servers with potential configuration conflicts or path issues.

References: BAR-976.

- Deprecate custom compression for WAL files

  The `custom` compression option and its related configuration options (`custom_compression_filter`, `custom_decompression_filter`, and `custom_compression_magic`) are now deprecated and will be removed in a future release.

  Users currently using custom compression are advised to migrate to one of the built-in compression algorithms supported by Barman: `gzip`, `bzip2`, `lz4`, `xz`, `zstd`, or `pigz`.

  The custom compression feature was designed to provide flexibility for users with specific compression requirements, but maintaining compatibility with arbitrary external compression tools has proven to be complex and error-prone. The built-in compression options provide better performance, reliability, and maintainability.

  Existing configurations using custom compression will continue to work in the current release, but users should plan to migrate their configurations before the feature is removed.

  References: BAR-993.

22.3.2 Minor changes

- Handle non-UTF8 output when checking for file encryption

  In certain scenarios, such as when recovering WALs via `barman-wal-restore` or running `barman rebuild-xlogdb`, Barman executes the `file` command to check whether a file is encrypted.

  On macOS systems (which are not officially supported), this command was reported to be raising a `UnicodeDecodeError` exception if its output contained bytes that could not be decoded using UTF-8.

  To prevent this, Barman now handles undecodable bytes gracefully, allowing the process to continue without interruption.

  Github issue: #1117

  References: BAR-926.

- Add `--recovery-option-port` to `barman restore` command

  When doing a remote restore with `get-wal` mode, Barman writes a `restore_command` which invokes `barman-wal-restore` to fetch WALs. However, there was currently no way to specify a custom port to the written `barman-wal-restore` command.

  This change introduces the `--recovery-option-port` flag to the `barman restore` command, allowing users to specify a value for the `--port` option of `barman-wal-restore` command during remote restores.

  References: BAR-943.

- Add S3 Object Lock support for the deletion of backups in barman-cloud

  `barman-cloud-backup-delete` now supports checking for Object Lock before deleting backups in AWS S3. This feature helps prevent deletion of objects protected by compliance or governance retention policies.

A new `--check-object-lock` option has been added, which enables Object Lock verification before deletion. When enabled, the deletion operation for all objects of a base backup will be aborted if any of those objects is locked. For performance reasons, lock checks are only performed for base backup files, since once the base backup is removed, the corresponding WAL files are no longer useful.

Note that this feature adds overhead to the delete operation, as it requires an additional API call for each object being deleted. This feature is only available for AWS S3-compatible object stores and requires the appropriate permissions to function correctly.

References: BAR-946.

22.3.3 Bugfixes

- Increase coverage of non-compliant S3 object stores for backup deletion

  With this fix we increase the coverage of non-compliant S3 object stores for backup deletion.

  We’ve reduced the expression we were matching for the message returned on failure to catch any error code that returns a message that contains “content-MD5”.

  We believe with this, almost all non-compliant S3 object stores for backup deletion will be covered

  References: BAR-974.

- Fix passive node not syncing

  In Barman 3.16 the “name” field was removed from the “config” key of the `barman diagnose` command. Turns out the same “config” data is also used when syncing a passive node, and the absence of the “name” field caused the sync to fail.

  This change adds back the “name” field to the “config” key, fixing the sync of passive nodes.

  Reported in GitHub issue #1125.

  References: BAR-971.

- Fix delta restore not working

  The delta restore feature was not functioning correctly due to an oversight where Barman was removing the destination directory before performing a restore. The problem was fixed by avoiding the removal of the destination directory on certain operations. Reported in GitHub issue #1139.

  References: BAR-1002.

- Fix “rm execution failed” errors during restore

  As a consequence of a refactoring implemented during the release of version 3.15, Barman started removing the destination directory of the restore operation before copying backup files. This was to ensure that the files were always being copied to a clean directory.

  However, when the destination directory is a mount point, or a subdirectory where the user does not have sufficient permissions on its parent, the removal of the directory fails, leading to errors like “rm execution failed”.

  The issue is now fixed by avoiding the removal of the destination directory for PGDATA and tablespaces on the restore final destination.

  References: BAR-975.

22.4 3.16.2 (2025-11-04)

22.4.1 Bugfixes

- Improve barman-cloud delete workaround to inspect error message content

  Some S3-compatible storages return an "InvalidRequest" error instead of "MissingContentMD5" during delete operations. Since we cannot rely on the error code alone, this updates the logic to also inspect the error message text to detect MissingContentMD5-like issues and apply the appropriate fallback.

  References: BAR-947.

22.5 3.16.1 (2025-10-14)

22.5.1 Notable changes

- Automatic bucket creation added to barman-cloud-check-wal-archive

  The `barman-cloud-check-wal-archive` command now verifies whether the target bucket exists in the configured cloud storage (e.g., AWS S3) and automatically creates it if it does not.

  Previously, automatic bucket creation was handled by `barman-cloud-backup`. This change restores compatibility with CloudNativePG (CNPG) automations that depend on this behavior, which were affected after the release of Barman 3.16.0.

  In collaboration with the CloudNativePG team, we have reintroduced this functionality specifically in `barman-cloud-check-wal-archive` to ensure seamless operation of CNPG workflows and internal tests — all while preserving the intended behavior and simplicity for regular Barman users.

  References: BAR-931.

22.6 3.16.0 (2025-10-02)

22.6.1 Notable changes

- The `restore_command` for local restores no longer includes `sudo`

  When using `barman restore --get-wal` for a local restore (i.e. without `--remote-ssh-command`), the generated `restore_command` no longer includes a `sudo -u $USER` prefix. This simplifies the process for the common case where the locally restored Postgres cluster is run by the `barman` user, whom already owns the files by the end of the `barman restore` operation.

  Action Required: If you intend to run the restored cluster as a different user (e.g., `postgres`), you must now manually add the appropriate `sudo -u postgres` prefix to the `restore_command` in the recovery configuration and ensure file ownership is correct.

  References: BAR-675.

- Deprecate `pygzip` and `pybzip2` compression options

  The `pygzip` and `pybzip2` compression options are now deprecated and will be removed in a future release. Users should migrate to the `gzip` and `bzip2` options, which now use Python's internal libraries.

  This change was made to improve performance and code consistency. Previously, the `gzip` and `bzip2` options relied on external subprocesses, while their `py` counterparts used native Python libraries. The core compression logic for `gzip` and `bzip2` has been updated to also use Python's built-in libraries, making the `pygzip` and `pybzip2` options redundant.

  References: BAR-878.

- Add delta-restore flag and recovery option to barman restore

  Introduces the –delta-restore flag for barman restore, enabling delta restore mode. This mode restores a backup by reusing identical data already present in the destination directory, which drastically reduces restore time and network traffic.

  This feature is also configurable per-server via 'recovery_options' as a new field called delta-restore in the server configuration file.

  References: BAR-788.

- S3 Object Deletion Fallback for Object Stores non-compliant with boto3 >=1.36

  When using `delete_objects()` with some non-compliant S3-compatible object stores, users could encounter a `MissingContentMD5 ClientError` due to a breaking change in boto3 1.36 that Botocore will no longer automatically compute and populate the `Content-MD5` header.

  To address this, Barman now implements a fallback mechanism:

  - Bulk deletion via `delete_objects()` is still attempted first.
  - If `delete_objects()` fails with `MissingContentMD5`, Barman automatically falls back to deleting objects individually using `delete_object()`, which does not require a `Content-MD5` header.
  - Other errors continue to raise exceptions as before.

  This change ensures Barman remains compatible with non-compliant object stores without requiring changes on the user's side.

  Users with older S3-compatible storage can continue performing backup cleanups without failures, while newer object stores are unaffected.

  References: BAR-909.

- Cloud connectivity checks are now only performed with the `-t/--test` flag.

  Previously, Barman always tested cloud connectivity and bucket existence for most of the commands, raising errors if connectivity failed or if the bucket did not exist. Both checks are now executed only when the `-t/--test` flag is specified. The only exception was `barman-cloud-check-wal-archive`, which still checks connectivity and the existence of the bucket.

  Additionally, buckets are no longer created automatically — Barman assumes the target bucket already exists, making it the user's responsibility to provision the bucket in advance.

  References: BAR-895, BAR-908, BAR-918.

22.6.2 Minor changes

- Add `combine-mode` option to `barman restore` command

  Added a `--combine-mode` option to the `barman restore` command, which can be used to specify a copy mode for `pg_combinebackup` when combining incremental backups during a restore. Available modes are `copy`, `link`, `clone`, and `copy-file-range` which are equivalent to the `pg_combinebackup` options `--copy`, `--link`, `--clone`, and `--copy-file-range`, respectively. An equivalent `combine_mode` configuration option can be specified in the configuration file.

  This feature allows for greater optimization of the restore process based on the underlying filesystem capabilities.

  References: BAR-870.

- Reorganize fields in barman diagnose JSON

  In the barman diagnose JSON, some fields appeared under the server's "config" section even though they were not actual configuration fields.

The fields "barman_lock_directory", "lock_directory_cleanup", and "config_changes_queue" were relocated under the "global" key, since they represent global configurations. The "name" field was removed since the server name is already used as the key. Additionally, "msg_list" is now stored under the server's name key.

References: BAR-134.

- Add '-p JOBS' to barman-wal-restore command when `--get-wal` is `True`

  Adds '-p JOBS' to barman-wal-restore command when `--get-wal` or `get_wal` configuration option is `True`, `JOBS` being the number of jobs chosen by the user via `-p` flag or `parallel_jobs` configuration option.

  References: BAR-857.

- The `barman show-backup`` command now displays the backup compression method

  The output of the `barman show-backup` command has been enhanced to include a new `Backup Compression` field. If a backup was taken with compression enabled, this field will now display the compression algorithm that was used (e.g., `zstd`). This makes it easier to quickly verify the compression status of a backup.

  References: BAR-901.

22.6.3 Bugfixes

- Fix parallelism (–parallel option) in barman-wal-restore

  There has been a regression in `barman-wal-restore` when introducing the `--keep-compression` option in Barman 3.12.0, which made the `--parallel` option not invoke processes as expected. This issue has now been fixed.

  References: BAR-865.

- Fix `pg_combinebackup` not being found in `PATH` when `staging_location` is `remote`

  When recovering an incremental backup to a remote server with `staging_location` set to `remote`, Barman checks if `pg_combinebackup` is available in the remote `PATH`. However, a previous issue was making it incorrectly check the local `PATH` as well, leading to errors if `pg_combinebackup` was not installed locally. The problem has been fixed by ensuring that the check is only performed on the server where `pg_combinebackup` is actually executed.

  References: BAR-903.

22.7 3.15.0 (2025-08-05)

22.7.1 Notable changes

- Allow incremental backups to be taken with compression and encryption

  Previously, Barman prevented incremental backups to be taken if compression was enabled or if the parent backup was already compressed. Since encryption in Barman requires compression, this also meant that incremental backups could not be taken with encryption.

  This limitation existed due to the complexity of restoring such backups, which involves multiple staging phases for decryption, decompression, and combination of backups.

  After a significant refactoring effort, Barman now fully supports taking incremental backups with both compression and/or encryption. This allows for more flexible backup strategies, enabling users to take advantage of all available features without restrictions.

  References: BAR-764, BAR-801, BAR-841.

- Deprecate `local_staging_path` and `recovery_staging_path` configuration options

  The `local_staging_path` and `recovery_staging_path` configuration options have been deprecated in favor of the new `staging_path` and `staging_location` options. The old options will continue to work for backwards compatibility, but users are encouraged to migrate to the new configuration options for better clarity and flexibility.

  References: BAR-571, BAR-801.

- Improve flexibility and consistency of staging area configuration

  Whenever restoring a compressed, encrypted or incremental backup, Barman needs a staging area to handle intermediate files, such as decompressed or decrypted files, or the combined incremental backup data.

  Previously, Barman used to have different configuration options for each scenario:

  - For incremental backups, the staging area was specified via the `local_staging_path` in the Barman configuration, which could only be a path in the Barman host.
  - For compressed backups, the staging area was specified via the `recovery_staging_path`, which could be a path in the Barman host (in case of local recovery) or a path in the target server (in case of remote recovery to a different server).
  - For encrypted backups, the staging area was specified via the `local_staging_path` in the Barman configuration, which could only be a path in the Barman host. Note that, since encryption requires compression to be enabled, when restoring an encrypted backup, users needed to specify both `recovery_staging_path` and `local_staging_path`.

  Barman now provides a single, unified and more flexible staging area configuration:

  - The new `staging_path` configuration option can be used to specify an absolute path for the staging area, where intermediate files will be stored. The default value is `/tmp`.
  - The new `staging_location` configuration option can be used to specify whether the staging area is on the Barman host (`local`) or on the remote server (`remote`). The default value is `local`.

  This means that, besides having a unified configuration for the staging path, users are now also able to specify with more flexibility where the staging area should be located, either on the Barman host or on the remote Postgres server.

  Storage usage has also been optimized to reduce resource consumption whenever possible. This includes:

  - Removing intermediate files as soon as they are no longer needed.
  - Avoid using the staging area when not necessary, e.g. when restoring a compressed backup locally, no staging is needed and the backup is decompressed directly to its final destination.

  This change simplifies configuration and improves consistency and resource usage, making the recovery management more transparent and predictable for end users.

  References: BAR-764, BAR-765, BAR-801, BAR-841.

22.7.2 Minor changes

- Add `--staging-path` and `--staging-location` to `barman restore`

  Added `--staging-path` and `--staging-location` options to `barman restore` command to allow the user to specify a custom path for the staging area during the restore process. `--staging-path` is the absolute path to be used while `--staging-location` defines its location i.e. `local` or `remote`. They also have its equivalent configuration options in `barman.conf` as `staging_path` and `staging_location` respectively.

  References: BAR-570.

- Add option `--if-not-exists` to `barman receive-wal --create-slot`

  A new `--if-not-exists` flag has been added to the `barman receive-wal --create-slot` command. This prevents the command from failing when attempting to create a replication slot that has already been created, making it more reliable for use in scripts.

  Special thanks to @crazybolillo antonio@zoftko.com a.k.a @antonag32 for creating a PR and contributing to Barman.

  References: BAR-527.

- Fail gracefully when an unexpected field is found in the backup metadata.

  This update improves how Barman handles backup metadata that includes fields unknown to the current version. This situation can occur when attempting to restore a backup created with a newer version of Barman using an older one. For example, if a newer version adds metadata about encryption, and the older version doesn't recognize that field. That's a case where this change is relevant.

  Previously, this would cause an unhelpful exception. Now, Barman will display a clearer message explaining that the backup was created with a newer version and may not be compatible with the current version.

  References: BAR-763.

- Improve orphan backup warning for proper cleanup

  The warning message for orphan backups has been improved to help users fully clean up incomplete or manually removed backups. Previously, the message did not mention the location of the `backup.info` file, which led to confusion. The updated message now clearly indicates the location of the `backup.info` file to help users properly remove it.

  References: BAR-793.

- Improve tag handling in cloud commands to fix parsing errors

  The `--tags` command-line option in `barman-cloud-backup` and `barman-cloud-wal-archive` has been replaced to improve parsing reliability and align with standard CLI practices. The previous space-separated format could cause parsing errors, especially when placed before positional arguments.

  The commands now accept multiple `--tag` flags, one for each key-value pair.

  Before: `barman-cloud-backup ... --tags key1,val1 key2,val2`

  After: `barman-cloud-backup ... --tag key1,val1 --tag key2,val2`

  The `--tags` option is now deprecated. While existing scripts using the old `--tags` format will continue to work, users are encouraged to update their scripts to the new format. This change is backward compatible and does not introduce a breaking change.

  References: BAR-441.

- Clarify `WAL Number` label in barman show-backup output

  Rename misleading `WAL Number` label in `barman show-backup` output to `Number of WALs`.

  References: BAR-791.

- Change "No of files" to "Number of files" in show-backup command

  In the Wal Information section of the show-backup command, the output of "No of files" was changed to "Number of files".

  References: BAR-798.

- Validate options passed for servers and models through `config-update`

The `barman config-update` command now performs extra validation on each provided section before enqueuing changes. It verifies, among other things, that all required fields are present, that the provided values conform to the expected types, and that each value also passes its respective parser. In addition, the command will now error out if any of the checks fail. This prevents partial or malformed updates from being applied.

References: BAR-142.

- Add `--check-timeout` argument to `barman check` and `barman backup` commands

  A new CLI option has been introduced for the `barman check` and `barman backup` commands to address situations where the check operation duration exceeds the configured `check_timeout`, preventing backups from being scheduled. This option allows users to disable the timeout or override the default or configured `check_timeout` value.

  References: BAR-219.

- Add `--list-empty-directories` to `barman list-files` command

  Adds a new `--list-empty-directories` argument to the `list-files` command to add empty directories to the listing.

  References: BAR-220.

- Improve archiving check with automatic WAL switching and archiving

  To improve the user experience, especially during initial server configuration, the `barman check` command is now more robust.

  Previously, when the WAL archive was empty, the check would fail and require manual intervention.

  Barman now intelligently handles this by first attempting to archive any existing WAL present in the Barman server. If none exists, it automatically triggers a WAL switch in the Postgres server and tries to archive the resulting file. This enhancement automates a common manual step and makes the initial health check more reliable out-of-the-box.

  References: BAR-718.

- Support shortcuts as backup IDs in Barman cloud scripts

  It is now possible to specify the shortcuts `first/oldest`, `last/latest` and `last-failed` when referencing backups in the Barman cloud commands. Barman now automatically parses these shortcuts and returns their associated backup ID.

  References: BAR-717.

22.7.3 Bugfixes

- Ignore data files of temporary tables during backups

  Files that match the pattern of temporary relation files will now be ignored during a `rsync` backup, similar to the exclusion implicitly performed by `pg_basebackup`. This is needed to avoid errors when running Barman in rsync-concurrent mode against a Postgres instance that creates and drops many temporary tables, since rsync might try to copy a temp file that had already been removed, causing Barman to hang until manually unblocked.

  References: BAR-266.

- Improve error message when `pg_combinebackup` is not found

  Previously, when restoring an incremental backup and `pg_combinebackup` was not available in the PATH, Barman did not handle the exception properly, which made it difficult to understand and fix the error. Now, a proper exception is raised with a clear message indicating that `pg_combinebackup` could not found in the target server.

  References: BAR-840.

- Fail on restore if the target data or tablespace directories are not empty.

  Previously, Barman did not verify whether tablespace destination directories were empty before restoring, which could result in their contents being silently overwritten — potentially causing data loss or corruption. Now, the restore process ensures that all destination directories, including those for data and tablespaces, are empty before proceeding. If any are not, Barman will halt the operation and display an appropriate error message, significantly improving safety and data integrity for both local and remote restores.

  References: BAR-787.

- Fix errors when deleting orphaned and obsolete backups

  Resolved an issue where applying retention policies could fail if a backup was both orphaned and obsolete. Barman now verifies that a base backup directory exists before attempting to delete it, preventing errors when the directory has already been removed.

  References: BAR-794.

- Ensure correct WAL segment naming for PG17 boundary LSNs

  Addresses a backup hang issue experienced with PostgreSQL 17 and later versions when using barman backup `<server>` –wait on low-activity instances and modifies the logic within the methods and properties that interact with pg_walfile_name() and pg_walfile_name_offset().

  As described in barman#1041, PostgreSQL 17 altered the behavior where the end WAL of a backup is now the currently written WAL file, rather than the last completed one. On servers with minimal activity, this current WAL file might not be completed or archived promptly, leading to the backup process stalling indefinitely until a pg_switch_wal() occurs.

  More details on the PostgreSQL change can be found at `https://www.postgresql.org/docs/release/17.0/`.

  References: BAR-519.

- Snapshot Backup Now Supports Non-Root EBS Volumes of Nitro instances on AWS

  Resolved an issue where snapshot backups failed to detect non-root EBS volumes on NVMe (Nitro) devices in AWS. Snapshot backups now work reliably with all non-root EBS volumes attached to the instance.

  References: BAR-500.

22.8 3.14.1 (2025-06-18)

22.8.1 Bugfixes

- Fix WAL archiving performance issues

  The Barman WAL archiving process was suffering from a performance degradation due to processing of WAL files that would not be archived in the current run. There was also an oversight in the encryption logic introduced in 3.14, that caused Barman to check for encryption in WAL files streamed to the server, which is unnecessary as streamed WALs are never GPG-encrypted, nor compressed. With both issues fixed, the archiving process will be significantly faster and more resource-efficient.

  Thanks to @thealex55 for the detailed analysis in the issue #1087.

  References: BAR-775.

22.9 3.14.0 (2025-05-15)

22.9.1 Notable changes

- Implementation of GPG encryption for tar backups and WAL files

  Implement GPG encryption of tar backups. Encryption starts at the end of the backup, encrypting the backup of PGDATA and tablespaces present in the backup directory. Encrypted backup files will have the `.gpg` extension added.

  Barman supports the decryption and restoration of GPG-encrypted backups using a passphrase obtained through the new `encryption_passphrase_command` configuration option. During the restore process, decrypted files are staged in the `local_staging_path` setting on the Barman host, ensuring a reliable and safe restore process.

  New configuration options required for encryption and decryption of backups and WAL files needed to be added. The new options are `encryption`, `encryption_key_id`, and `encryption_passphrase_command`.

  WAL files are all encrypted with GPG when `encryption = gpg`. This includes changing the way that xlogdb records are read and written (maintaining backwards compatibility), and a new logic to detect when files are encrypted and the encryption process itself.

  Decryption of GPG-encrypted WAL files during the restore process when using the get-wal and no-get-wal flags of the barman restore command. This extends the functionality added for decrypting backups via the `encryption_passphrase_command` configuration option.

  There's a new field in `show-backup` to expose if a backup was encrypted, and specifies the encryption method that was used, if any.

  The `barman check` command verifies if the user's encryption settings are correctly configured in the Barman server and functioning as expected.

  References: BAR-683, BAR-687, BAR-693, BAR-669, BAR-671, BAR-692, BAR-685, BAR-680, BAR-670, BAR-681, BAR-702.

- Deprecation of Python versions 3.6 and 3.7

  As of version 3.14 of Barman, we are deprecating support for Python 3.6 and 3.7, which are not supported by the Python community anymore. Barman still supports Python 3.8 as it's the newest version available on RHEL 8 systems which contains all needed modules.

  References: BAR-737.

22.9.2 Minor changes

- Allow compression level to be specified for WAL compression in Barman server

  Add a new `compression_level` parameter to the Barman configuration. This option accepts a valid integer value or one of the predefined options: `low`, `medium`, and `high`. Each option corresponds to a different level depending on the compression algorithm chosen.

  References: BAR-540.

- Add client-side compression to `barman-wal-archive`

  Client-side compression options have been added to `barman-wal-archive`, supporting the same algorithms that are available on a Barman server. When enabled, compression is applied on the client side before sending the WAL to the Barman server. The `--compression-level` parameter allows specifying a desired compression level for the chosen algorithm.

  References: BAR-262.

- Add –compression-level parameter to barman-cloud-wal-archive

  A parameter called `compression-level` was added to `barman-cloud-wal-archive`, which allows a level to be specified for the compression algorithm in use.

  References: BAR-557.

- Add Snappy compression algorithm to Barman server

  The Snappy compression, previously only available in `barman-cloud-wal-archive`, is now also available for standard Barman server. As with all other algorithms, it can be configured by setting `snappy` in the `compression` configuration parameter.

  References: BAR-557.

- Introduce the new `list-processes` sub-command for listing the server processes

  Add a new `list-processes` command that outputs all active subprocesses for a Barman server. The command displays each process' s PID and task.

  References: BAR-664.

- Introduce the new `terminate-process` sub-command for terminating Barman subprocesses

  Add a new `terminate-process` command that allows users to terminate an active Barman subprocess for a given server by specifying its task name. Barman will terminate the subprocess as long as it belongs to the specified server and it is currently active.

  References: BAR-665.

- Remove the pin from boto3 version used in cloud scripts

  After thorough investigation of issues with boto3 >= 1.36, we' ve decided to remove the pin that kept the dependency at version 1.35.

  Both AWS and MinIO object stores work correctly with the latest version, and using a version of boto3 that is >= 1.36 ensures the Barman cloud scripts work in a FIPS-compliant environment.

  References: BAR-637.

22.9.3 Bugfixes

- Ensure minimum redundancy check considers only 'non-incremental backups'

  An issue was reported where the `minimum_redundancy` rule could be violated due to the inclusion of incremental backups in the redundancy count. As an example: in a scenario where the catalog contained one full backup and two dependent incremental backups, and the user had `minimum_redundancy = 2`, the rule was incorrectly considered satisfied. As a result, deleting the full backup triggered cascading deletion of its incremental dependents, leaving zero backups in the catalog.

  This issue has been fixed by updating the `minimum_redundancy` logic to consider only non-incremental backups (i.e. only full, rsync, snapshot). This ensures that full backups cannot be deleted if doing so would violate the configured minimum redundancy level.

  References: BAR-707.

- Fix usage of `barman-wal-restore` with `--keep-compression` using `gzip`, `bzip2`, and `pigz` compression algorithms

  Fix an issue in `barman-wal-restore` where, when trying to restore WALs compressed with `gzip`, `bzip2` or `pigz` while having `--keep-compression` specified, leading to unexpected errors.

  References: BAR-722.

22.10 3.13.3 (2025-04-24)

22.10.1 Bugfixes

- Fix local restore of block-level incremental backups

  When performing a local restore of block-level incremental backups, Barman was facing errors like the following:

  ```
  ERROR: Destination directory '/home/vagrant/restore/internal_no_get_wal' must be␣
  ↪empty
  ```

  That error was caused by a regression when the option `--staging-wals-directory` was introduced in version 3.13.0. Along with it came a new check to ensure the WAL destination directory was empty before proceeding. However, when restoring block-level incremental backups locally, Barman was setting up the WAL destination directory before performing this check, triggering the error above.

  References: BAR-655.

- Fix regression when running `barman-cloud-backup` as a hook

  Barman 3.13.2 changed the location of the `backup.info` metadata file as part of the work delivered to fix issues in WORM environments.

  However, those changes introduced a regression when using `barman-cloud-backup` as a backup hook in the Barman server: the hook was not aware of the new location of the metadata file.

  This update fixes that issue, so `barman-cloud-backup` becomes aware of the new folder structure, and properly locates the `backup.info` file, avoiding runtime failures.

  References: BAR-696.

- Avoid decompressing partial WAL files when custom compression is configured

  Fixed an issue where Barman attempted to decompress partial WAL files when custom compression was configured. Partial WAL files are never compressed, so any attempt to decompress them is incorrect and caused errors when using the `--partial` flag with `barman-wal-restore` or `barman get-wal`.

  References: BAR-697.

- Fixed `barman-cloud-backup` not recycling temporary part files

  This fixes a `barman-cloud-backup` problem where temporary part files were not deleted after being uploaded to the cloud, leading to disk space exhaustion. The issue happened only when using Python >= 3.12 and it was due to a change in Python that removed the `delete` attribute of named-temporary file objects, which Barman used to rely on when performing internal checks.

  References: BAR-674.

- Fixed backup annotations usage in WORM environments

  Barman previously stored backup annotation files, used to track operations like `barman keep` and `barman delete`, inside the backup directory itself. These annotations help determine whether a backup should be kept or marked for deletion. However, in WORM environments, files in the backup directory cannot be modified or deleted after a certain period, which caused issues with managing backup states. This fix relocates annotation files to a dedicated metadata directory, as to ensure that such operations function correctly in WORM environments.

  References: BAR-663.

22.11 3.13.2 (2025-03-27)

22.11.1 Minor changes

- Fix errors when using an immutable storage

  Added a new `worm_mode` configuration to enable WORM (Write Once Read Many) handling in Barman, allowing it to support backups on immutable storage.

  This fix also provides automatic relocation of the backup.info file in a new directory `meta` inside `backup_directory`. This will let Barman update it in future when needed.

  Barman will also *not* purge the wals directory for WAL files that are not needed when running the first backup. This will add some extra space which will be reclaimed when this first backup is obsolete and removed (by that time, the backups and the WALs will be outside the retention policy window).

  Added additional notes to the documentation explaining limitations when running with an immutable storage for backups. In particular the need for a grace period in the immutability of files and the fact that `barman keep` is not supported in these environments.

  References: BAR-649, BAR-645, BAR-650, BAR-651, BAR-652.

22.12 3.13.1 (2025-03-20)

22.12.1 Minor changes

- Improve behavior of the backup shortcuts `last-full` / `latest-full`

  The shortcuts `last-full` / `latest-full` were retrieving not the last full backup of the server, but the last full backup of the server which was eligible as the parent for an incremental backup.

  While this was the expected behavior, the feedback from the community has shown that it was confusing for the users.

  From now on, the shortcuts `last-full` / `latest-full` will retrieve the last full backup of the Barman server, independently if that backup is eligible as the parent for an incremental backup or not.

  The eligibility of the full backup as the parent of an incremental backup will still be validated by Barman in a later step, and a proper message will be displayed in case it doesn't suit as a parent.

  References: BAR-555.

22.12.2 Bugfixes

- Fix error message when parsing invalid `--target-time` in `barman restore`

  When using the `barman restore` command, the error message when parsing invalid `--target-time` string was:

  ```
  EXCEPTION: local variable 'parsed_target' referenced before assignment
  ```

  That exception was replaced with an understandable error message.

  References: BAR-627.

- Fix mutual exclusive arguments in the cloud restore command

  In the `barman-cloud-restore` command, we were checking that `target_tli` and `target_lsn` were mutually exclusive arguments, where the correct pair to check would be `target_time` and `target_lsn`.

  References: BAR-624.

- Fix Barman not honoring `custom_decompression_filter`

  Fixed an issue where Barman was not honoring the configured `custom_decompression_filter` if the compression algorithm specified was natively supported by Barman. Custom filters now take priority over native handlers when decompressing WAL files.

  References: BAR-584.

- Fix barman restore with –no-get-wal and –standby

  Fixed an issue where Barman was removing the `pg_wal` directory during recovery if `--no-get-wal` and `--standby-mode` were specified together. The issue happened due to Barman incorrectly filling the recovery parameters referencing `pg_wal`, including `recovery_end_command`, which led to this issue. Barman will now ignore filling such parameters as they are not required for this specific case.

  References: BAR-630.

- Fix argument parsing issue in `barman restore` and `barman-cloud-restore`

  In Barman 3.13.0, a regression was introduced causing errors when using `barman restore` and `barman-cloud-restore` commands. Specifically, the `backup_id` positional argument, which was made optional in that version, conflicted with other arguments, causing unrecognized arguments and errors.

  For example, running `barman-cloud-restore` like this:

  ```
  barman-cloud-restore source_url server_name backup_id --cloud-provider aws-s3␣
  ↪recovery_dir
  ```

  Would trigger an error like this:

  ```
  barman-cloud-restore: error: unrecognized arguments: recovery_dir
  ```

  This fix resolves the issue by making `backup_id` a required argument again. Additionally, a new "auto" value is now accepted as a `backup_id`, allowing Barman to automatically choose the best backup for restoration without needing a specific `backup_id`. This update fixes argument handling and still allows a smooth and flexible restoration process for the user.

  References: BAR-596.

22.13 3.13.0 (2025-02-20)

22.13.1 Notable changes

- Add new xlogdb_directory configuration

  Introduces a new `xlogdb_directory` configuration option. This parameter can be set either globally or per-server, and allows you to specify a custom directory for the `xlog.db` file. This file stores metadata of archived WAL files and is used internally by Barman in various scenarios. If unset, it defaults to the value of `wals_directory`. Additionally, the file was also renamed to contain the server name as a prefix.

  References: BAR-483.

- Make "backup_id" optional when restoring a backup

  Historically, Barman always required a "backup_id" to restore a backup, and would use that backup as the source for the restore.

  This feature removes the need for specifying which backup to use as a source for restore, making it optional.

  This change applies to both Barman and the barman-cloud scripts.

Now the user is able to restore a backup in the following ways:

1. Provide a "backup_id"
2. Do not provide a "backup_id". It will retrieve the most recent backup
3. Do not provide a "backup_id", but provide a recovery target, such as: - "target_time" (mutually exclusive with target_lsn) Will get the closest backup prior to the "target_time" - "target_lsn" (mutually exclusive with "target_time") Will get the closest backup prior to the "target_lsn" - "target_tli" (can be used combined with "target_time" or "target_lsn") Will get the most recent backup that matches the timeline. If combined with other recovery targets, it will get the most recent backup prior to the target_time or target_lsn that matches the timeline

The recovery targets `--target-xid`, `--target-name` and `--target-immediate` are not supported, and will error out with a message if used.

This feature will provide flexibility and ease when restoring a postgres cluster.

References: BAR-541, BAR-473.

22.13.2 Minor changes

- Add current active model to `barman show-server` and `barman status`

  Previously, after applying a configuration model, the only way to check which model is currently active for a server was via the `barman diagnose` command. With this update, the `barman status` and `barman show-server` commands now also display the current active configuration model for a server, if any.

  References: BAR-524, BAR-400.

- Add `--staging-wal-directory` option to `barman restore` command to allow alternative WAL directory on PITR

  A new command line option `--staging-wal-directory` was added to the `restore/recover` command to allow an alternative destination directory for WAL files when performing PITR. Previously, WAL files were copied to a `barman_wal` directory within the restore destination directory. This enhancement provides greater flexibility, such as storing WALs on separate partitions during recovery.

  References: BAR-224.

- Pin boto3 version to any version <= 1.35.99

  Boto3 version 1.36 has changed the way S3 integrity is checked making this version incompatible with the current Barman code, generating the following error:

  An error occurred (MissingContentLength) when calling the PutObject operation

  As a temporary workaround, the version for boto3 is pinned to any version <= 1.35.99 until support for 1.36 is implemented in Barman.

  References: BAR-535.

- Make barman-wal-archive smarter when dealing with duplicate WAL files

  Under some corner cases, Postgres could attempt to archive the same WAL twice. For example: if `barman-wal-archive` copies the WAL file over to the Barman host, but the script is interrupted before reporting success to Postgres. New executions of `barman-wal-archive` could fail when trying to archive the same file again because the WAL was already copied from Postgres to Barman, but not yet processed by the asynchronous Barman WAL archiver.

  This minor change deals with this situation by verifying the checksum of the existing and the incoming file. If the checksums match the incoming file is ignored, otherwise an output info message is sent and the incoming

file is moved to the errors directory. The code will exit with 0 in both situations, avoiding WALs piling up in the Postgres host due to a failing `archive_command`.

References: BAR-225.

- Document procedure to clear WAL archive failure check

  While redesigning the Barman docs we missed adding a note advising users to run a `switch-wal` command if the server is idle and `barman check` returns a failure on "WAL archiving".

  This addresses the gap left from the previous documentation.

  References: BAR-521.

- Delete WALs by deleting the entire directory at once, when possible

  Previously, when WAL files needed to be deleted (e.g., due to deletion of a backup), Barman would iterate over every WAL file and delete them individually. This could cause performance issues, mainly in systems which use ZFS filesystem. With this change, the entire directory will be deleted whenever noticed that all files in the directory are no longer needed by Barman.

  References: BAR-511.

- Add support for `DefaultAzureCredential` option on Azure authentication

  Users can now explicitly use Azure's `DefaultAzureCredential` for authentication by using the `default` option for `azure_credential` in the server configuration or the `--azure-credential default` option in the case of `barman-cloud-*`. Previously, that could only be set as a fallback when no credential was provided and no environment variables were set.

  References: BAR-539.

- Improve diagnose output for retention policy info

  Improves the output of the barman diagnose command to display a more user-friendly string representations. Specifically, "REDUNDANCY 2" is shown instead of "redundancy 2 b" for the 'retention_policy' attribute, and "MAIN" is shown instead of "simple-wal 2 b" for the 'wal_retention_policy' attribute.

  References: BAR-100.

22.13.3 Bugfixes

- Fix PITR when using `barman restore` with `--target-tli`

  Barman was not creating the `recovery.signal` nor filling `recovery_target_timeline` in `postgresql.auto.conf` in these cases:

  - The only recovery target passed to `barman restore` was `--target-tli`; or
  - `--target-tli` was specified with some other `--target-*` option, but the specified target timeline was the same as the timeline of the chosen backup.

  Now, if any `--target-*` option is passed to `barman restore`, that will be correctly treated as PITR.

  References: BAR-543.

- Fix bug when AWS 'profile' variable is referenced before assignment

  An issue was introduced by BAR-242 as part of the Barman 3.12.0 release. The issue was causing `barman-cloud-backup-delete` (and possibly other commands) to fail with errors like this when `--aws-profile` argument or `aws_profile` configuration were not set:

```
ERROR: Barman cloud backup delete exception: local
variable 'profile' referenced before assignment`
```

References: BAR-518.

- Fix –zstd flag on barman-cloud-wal-archive

  Fixed a bug with the `--zstd` flag on `barman-cloud-wal-archive` where it was essentially being ignored and not really compressing the WAL file before upload.

  References: BAR-567.

22.14 3.12.1 (2024-12-09)

22.14.1 Bugfixes

- Add isoformat fields for backup start and end times in json output

  This patch modifies the json output of the infofile object adding two new fields: `begin_time_iso` and `end_time_iso`. The new fields allow the use of a more standard and timezone aware time format, preserving compatibility with previous versions. It is worth noting that in the future the iso format for dates will be the standard used by barman for storing dates and will be used everywhere non human readable output is requested.

  As part of the work, this patch reverts BAR-316, which was introduced on Barman 3.12.0.

  References: BAR-494.

22.15 3.12.0 (2024-11-21)

22.15.1 Minor changes

- Add FIPS support to Barman

  The `md5` hash algorithm is not FIPS compliant, so it is going to be replaced by `sha256`. `sha256` is FIPS compliant, vastly used, and is considered secure for most practical purposes. Up until this release, Barman's WAL archive client used `hashlib.md5` to generate checksums for tar files before they were sent to the Barman server. Here, a tar file is a file format used for bundling multiple files together with a `MD5SUMS` file that lists the checksums and their corresponding paths. In this release, the `md5` hashing algorithm is replaced by `sha256` as the default. As a result, checksums for the tar files will be calculated using `sha256`, and the `MD5SUMS` file will be named `SHA256SUMS`. Barman still has the ability to use the nondefault `md5` algorithm and the `MD5SUMS` file from the client if there is a use case for it. The user just needs to add the `--md5` flag to the `barman-wal-archive` `archive_command`.

  References: BAR-155, CP-34954, CP-34391.

- Removed el7, debian10, and ubuntu1804 support; updated Debian and SLES.

  Support for el7, debian10, and ubuntu1804 has been removed. Additionally, version 12 and version name "bookworm" has been added for Debian, addressing a previously missing entry. The SLES image version has also been updated from sp4 to sp5.

  References: BAR-389.

- Add support for Postgres Extended 17 (PGE) and Postgres Advanced Server 17 (EPAS)

  Tests were conducted on Postgres Extended 17 (PGE) and Postgres Advanced Server 17 (EPAS), confirming full compatibility with the latest features in Barman. This validation ensures that users of the latest version of PGE and EPAS can leverage all the new capabilities of Barman with confidence.

  References: BAR-331.

- Improve WAL compression with `zstd`, `lz4` and `xz` algorithms

Introduced support for xz compression on WAL files. It can be enabled by specifying `xz` in the `compression` server parameter. WALs will be compressed when entering the Barman's WAL archive. For the cloud, it can be enabled by specifying `--xz` when running `barman-cloud-wal-archive`.

Introduced support for zstandard compression on WAL files. It can be enabled by specifying `zstd` in the `compression` server parameter. WALs will be compressed when entering the Barman's WAL archive. For the cloud, it can be enabled by specifying `--zstd` when running `barman-cloud-wal-archive`.

Introduced support for lz4 compression on WAL files. It can be enabled by specifying `lz4` in the `compression` server parameter. WALs will be compressed when entering the Barman's WAL archive. For the cloud, it can be enabled by specifying `--lz4` when running `barman-cloud-wal-archive`.

References: BAR-265, BAR-423, BAR-264.

- Improve WAL upload performance on S3 buckets by avoiding multipart uploads

  Previously, WAL files were being uploaded to S3 buckets using multipart uploads provided by the boto3 library via the `upload_fileobj` method. It was noticed that multipart upload is slower when used for small files, such as WAL segments, compared to when uploading it in a single PUT request. This has been improved by avoiding multipart uploads for files smaller than 100MB. The average upload time of each WAL file is expected to be reduced by around 15% with this change.

  References: BAR-374.

- Modify behavior when enforcing retention policy for `KEEP:STANDALONE` full backups

  When enforcing the retention policy on full backups created with `backup_method = postgres`, Barman was previously marking all dependent (child) incremental backups as `VALID`, regardless of the KEEP target used. However, this approach is incorrect:

  - For backups labeled `KEEP:STANDALONE`, Barman only retains the WAL files needed to restore the server to the exact state of that backup. Because these backups are self-contained, any dependent child backups are no longer needed once the root backup is outside the retention policy.
  - In contrast, backups marked `KEEP:FULL` are intended for point-in-time recovery. To support this, Barman retains all WALs, as well as any child backups, to ensure the backup's consistency and allow recovery to the latest possible point.

  This distinction ensures that `KEEP:STANDALONE` backups serve as snapshots of a specific moment, while `KEEP:FULL` backups retain everything needed for full point-in-time recovery.

  References: BAR-366.

- Update documentation and user-facing features for Barman's recovery process.

  Barman docs and the tool itself used to use the terms "recover"/"recovery" both for referencing:

  - The Postgres recovery process;
  - The process of restoring a backup and preparing it for recovery.

  Both the code and documentation have been revised to accurately reflect the usage of the terms "restore" and "recover"/"recovery".

  Also, the `barman recover` command was renamed to `barman restore`. The old name is still kept as an alias for backward compatibility.

  References: BAR-337.

- Add –keep-compression flag to barman-wal-restore and get-wal

  A new `--keep-compression` option has been added to both `barman-wal-restore` and `get-wal`. This option controls whether compressed WAL files should be decompressed on the Barman server before being fetched. When specified with `get-wal`, default decompression is skipped, and the output is the WAL file content in its

original state. When specified with `barman-wal-restore`, the WAL file is fetched as-is and, if compressed, decompressed on the client side.

References: BAR-435.

- Ransomware protection - Add AWS Snapshot Lock Support

  Barman now supports AWS EBS Snapshot Lock, a new integrated feature to prevent accidental or malicious deletions of Amazon EBS snapshots. When a snapshot is locked, it can't be deleted by any user but remains fully accessible for use. This feature enables you to store snapshots in WORM (Write-Once-Read-Many) format for a specified duration, helping to meet regulatory requirements by keeping the data secure and tamper-proof until the lock expires.

  Special thanks to Rui Marinho, our community contributor who started this feature.

  References: BAR-242.

- Prevent orphan files from being left from a crash while deleting a backup

  This commit fixes an issue where backups could leave behind files if the system crashed during the deletion of a backup.

  Now, when a backup is deleted, it will get a "delete marker" at the start. If a crash happens while the backup is being deleted, the marker will help recognize incomplete backup removals when the server restarts.

  The Barman cron job has been updated to look for these deleted markers. If it finds a backup with a "delete marker", it will complete the process.

  References: BAR-244.

- Add support for using tags with snapshots

  Barman now supports tagging the snapshots when creating backups using the barman-cloud-backup script command. A new argument called –tags was added.

  Special thanks to Rui Marinho, our community contributor who started this feature.

  References: BAR-417.

- Use ISO format instead of ctime when producing JSON output of Barman cloud commands

  The ctime format has no information about the time zone associated with the timestamp. Besides that, that format is better suited for human consumption. For machine consumption the ISO format is better suited.

  References: BAR-316.

22.15.2 Bugfixes

- Fix barman check which returns wrong results for Replication Slot

  Previously, when using architectures which backup from a standby node and stream WALs from the primary, Barman would incorrectly use `conninfo` (pointing to a standby server) for replication checks, leading to errors such as:

  ```
  replication slot (WAL streaming):  FAILED (replication slot 'barman' doesn't exist.
  Please execute 'barman receive-wal --create-slot pg17')
  ```

  This fixes the following issue #1024 by ensuring `wal_conninfo` is used for WAL replication checks if it's set.

  `wal_conninfo` takes precedence over `wal_streaming_conninfo`, when both are set. With this change, if only `wal_conninfo` is set, it will be used and will not fall back to `conninfo`.

  Also, in the documentation, changes were made so it is explicit that when `conninfo` points to a standby server, `wal_conninfo` must be set and used for accurate replication status checks.

References: BAR-409.

- Fix missing options for `barman keep`

  The error message that the Barman CLI emitted when running `barman keep` without any options suggested there were shortcut aliases for status and release. These aliases, -s and -r, do not exist, so the error message was misleading. This fixes the issue by including these short options in the Barman CLI, aligning it with other tools like `barman-cloud-backup-keep`, where these shortcuts already exist.

  References: BAR-356.

- Lighten standby checks related to conninfo and primary_conninfo

  When backing up a standby server, Barman performs some checks to assert that `conninfo` is really pointing to a standby (in recovery mode) and that `primary_conninfo` is pointing to a primary (not in recovery).

  The problem, as reported in the issues #704 and #744, is that when a failover occurs, the `conninfo` will now be pointing to a primary instead and the checks will start failing, requiring the user to change Barman configs manually whenever a failover occurs.

  This fix solved the issue by making such checks non-critical, which means they will still fail but Barman will keep operating regardless. Essentially, Barman will ignore `primary_conninfo` if `conninfo` does not point to a standby. Warnings about this misconfiguration will also be emitted whenever running any Barman command so the user can be aware.

  References: BAR-348.

- Check for USAGE instead of MEMBER when calling pg_has_role in Barman

  To work correctly Barman database user needs to be included in some roles. Barman was verifying the conditions was satisfied by calling `pg_has_role` in Postgres. However, it was check for the `MEMBER` privilege instead of `USAGE`. This oversight was fixed.

  This change is a contribution from @RealGreenDragon.

  References: BAR-489.

22.16 3.11.1 (2024-08-22)

22.16.1 Bugfixes

- Fix failures in `barman-cloud-backup-delete`. This command was failing when applying retention policies due to a bug introduced by the previous release.

22.17 3.11.0 (2024-08-22)

22.17.1 Notable changes

- Add support for Postgres 17+ incremental backups. This major feature is composed of several small changes:
  - Add `--incremental` command-line option to `barman backup` command. This is used to specify the parent backup when taking an incremental backup. The parent can be either a full backup or another incremental backup.
  - Add `latest-full` shortcut backup ID. Along with `latest`, this can be used as a shortcut to select the parent backup for an incremental backup. While `latest` takes the latest backup independently if it is full or incremental, `latest-full` takes the latest full backup.
  - `barman keep` command can only be applied to full backups when `backup_method = postgres`.

  - Retention policies do not take incremental backups into consideration. As incremental backups cannot be recovered without having the complete chain of backups available up to the full backup, only full backups account for retention policies. If a full backup has dependent incremental backups and the retention policy is applied, the full backup will propagate its status to the associated incremental backups. When the full backup is flagged with any `KEEP` target, Barman will set the status of all related incremental backups to `VALID`.
  - When deleting a backup all the incremental backups depending on it, if any, are also removed.
  - `barman recover` needs to combine the full backup with the chain of incremental backups when recovering. The new CLI option `--local-staging-path`, and the corresponding `local_staging_path` configuration option, are used to specify the path in the Barman host where the backups will be combined when recovering an incremental backup.
- Changes to `barman show-backup` output:
  - Add the "Estimated cluster size" field. It's useful to have an estimation of the data directory size of a cluster when restoring a backup. It's particularly useful when recovering compressed backups or incremental backups, situations where the size of the backup doesn't reflect the size of the data directory in Postgres. In JSON format, this is stored as `cluster_size`.
  - Add the "WAL summarizer" field. This field shows if `summarize_wal` was enabled in Postgres at the time the backup was taken. In JSON format, this is stored as `server_information.summarize_wal`. This field is omitted for Postgres 16 and older.
  - Add "Data checksums" field. This shows if `data_checkums` was enabled in Postgres at the time the backup was taken. In JSON format, this is stored as `server_information.data_checksums`.
  - Add the "Backup method" field. This shows the backup method used for this backup. In JSON format, this is stored as `base_backup_information.backup_method`.
  - Rename the field "Disk Usage" as "Backup Size". The latter provides a more comprehensive name which represents the size of the backup in the Barman host. The JSON field under `base_backup_information` was also renamed from `disk_usage` to `backup_size`.
  - Add the "WAL size" field. This shows the size of the WALs required by the backup. In JSON format, this is stored as `base_backup_information.wal_size`.
  - Refactor the field "Incremental size". It is now named "Resources saving" and it now shows an estimation of resources saved when taking incremental backups with `rsync` or `pg_basebackup`. It compares the backup size with the estimated cluster size to estimate the amount of disk and network resources that were saved by taking an incremental backup. In JSON format, the field was renamed from `incremental_size` to `resource_savings` under `base_backup_information`.
  - Add the `system_id` field to the JSON document. This field contains the system identifier of Postgres. It was present in console format, but was missing in JSON format.
  - Add fields related with Postgres incremental backups:
    * "Backup type": indicates if the Postgres backup is full or incremental. In JSON format, this is stored as `backup_type` under `base_backup_information`.
    * "Root backup": the ID of the full backup that is the root of a chain of one or more incremental backups. In JSON format, this is stored as `catalog_information.root_backup_id`.
    * "Parent backup": the ID of the full or incremental backup from which this incremental backup was taken. In JSON format, this is stored as `catalog_information.parent_backup_id`.
    * "Children Backup(s)": the IDs of the incremental backups that were taken with this backup as the parent. In JSON format, this is stored as `catalog_information.children_backup_ids`.

    * "Backup chain size": the number of backups in the chain from this incremental backup up to the root backup. In JSON format, this is stored as `catalog_information.chain_size`.

- Changes to `barman list-backup` output:
  - It now includes the backup type in the JSON output, which can be either `rsync` for backups taken with rsync, `full` or `incremental` for backups taken with `pg_basebackup`, or `snapshot` for cloud snapshots. When printing to the console the backup type is represented by the corresponding labels `R`, `F`, `I` or `S`.
  - Remove tablespaces information from the output. That was bloating the output. Tablespaces information can still be found in the output of `barman show-backup`.
- Always set a timestamp with a time zone when configuring `recovery_target_time` through `barman recover`. Previously, if no time zone was explicitly set through `--target-time`, Barman would configure `recovery_target_time` without a time zone in Postgres. Without a time zone, Postgres would assume whatever is configured through `timezone` GUC in Postgres. From now on Barman will issue a warning and configure `recovery_target_time` with the time zone of the Barman host if no time zone is set by the user through `--target-time` option.
- When recovering a backup with the "no get wal" approach and `--target-lsn` is set, copy only the WAL files required to reach the configured target. Previously Barman would copy all the WAL files from its archive to Postgres.
- When recovering a backup with the "no get wal" approach and `--target-immediate` is set, copy only the WAL files required to reach the consistent point. Previously Barman would copy all the WAL files from its archive to Postgres.
- `barman-wal-restore` now moves WALs from the spool directory to `pg_wal` instead of copying them. This can improve performance if the spool directory and the `pg_wal` directory are in the same partition.
- `barman check-backup` now shows the reason why a backup was marked as `FAILED` in the output and logs. Previously for a user to know why the backup was marked as `FAILED`, they would need to run `barman show-backup` command.
- Add configuration option `aws_await_snapshots_timeout` and the corresponding `--aws-await-snapshots-timeout` command-line option on `barman-cloud-backup`. This specifies the timeout in seconds to wait for snapshot backups to reach the completed state.
- Add a keep-alive mechanism to rsync-based backups. Previously the Postgres session created by Barman to run `pg_backup_start()` and `pg_backup_stop()` would stay idle for as long as the base backup copy would take. That could lead to a firewall or router dropping the connection because it was idle for a long time. The keep-alive mechanism sends heartbeat queries to Postgres through that connection, thus reducing the likelihood of a connection getting dropped. The interval between heartbeats can be controlled through the new configuration option `keepalive_interval` and the corresponding CLI option `--keepalive-interval` of the `barman backup` command.

22.17.2 Bugfixes

- When recovering a backup with the "no get wal" approach and `--target-time` set, copy all WAL files. Previously Barman would attempt to "guess" the WAL files required by Postgres to reach the configured target time. However, the mechanism was not robust enough as it was based on the stats of the WAL file in the Barman host (more specifically the creation time). For example: if there were archiving or streaming lag between Postgres and Barman, that could be enough for recovery to fail because Barman would miss to copy all the required WAL files due to the weak check based on file stats.
- Pin `python-snappy` to `0.6.1` when running Barman through Python 3.6 or older. Newer versions of `python-snappy` require `cramjam` version `2.7.0` or newer, and these are only available for Python 3.7 or newer.
- `barman receive-wal` now exits with code `1` instead of `0` in the following cases:

  - Being unable to run with `--reset` flag because `pg_receivewal` is running.
  - Being unable to start `pg_receivewal` process because it is already running.
- Fix and improve information about Python in `barman diagnose` output:
  - The command now makes sure to use the same Python interpreter under which Barman is installed when outputting the Python version through `python_ver` JSON key. Previously, if an environment had multiple Python installations and/or virtual environments, the output could eventually be misleading, as it could be fetched from a different Python interpreter.
  - Added a `python_executable` key to the JSON output. That contains the path to the exact Python interpreter being used by Barman.

22.18 3.10.1 (2024-06-12)

22.18.1 Bugfixes

- Make `argcomplete` optional to avoid installation issues on some platforms.
- Load `barman.auto.conf` only when the file exists.
- Emit a warning when the `cfg_changes.queue` file is malformed.
- Correct in documentation the postgresql version where `pg_checkpoint` is available.
- Add `--no-partial` option to `barman-cloud-wal-restore`.

22.19 3.10.0 (2024-01-24)

22.19.1 Notable changes

- Limit the average bandwidth used by `barman-cloud-backup` when backing up to either AWS S3 or Azure Blob Storage according to the value set by a new CLI option `--max-bandwidth`.
- Add the new configuration option `lock_directory_cleanup` That enables cron to automatically clean up the barman_lock_directory from unused lock files.
- Add support for a new type of configuration called `model`. The model acts as a set of overrides for configuration options for a given Barman server.
- Add a new barman command `barman config-update` that allows the creation and the update of configurations using JSON

22.19.2 Bugfixes

- Fix a bug that caused `--min-chunk-size` to be ignored when using barman-cloud-backup as hook script in Barman.

22.20 3.9.0 (2023-10-03)

22.20.1 Notable changes

- Allow `barman switch-wal --force` to be run against PG>=14 if the user has the `pg_checkpoint` role (thanks to toydarian for this patch).
- Log the current check at `info` level when a check timeout occurs.

- The minimum size of an upload chunk when using `barman-cloud-backup` with either S3 or Azure Blob Storage can now be specified using the `--min-chunk-size` option.
- `backup_compression = none` is supported when using `pg_basebackup`.
- For PostgreSQL 15 and later: the allowed `backup_compression_level` values for `zstd` and `lz4` have been updated to match those allowed by `pg_basebackup`.
- For PostgreSQL versions earlier than 15: `backup_compression_level = 0` can now be used with `backup_compression = gzip`.

22.20.2 Bugfixes

- Fix `barman recover` on platforms where Multiprocessing uses spawn by default when starting new processes.

22.21 3.8.0 (2023-08-31)

22.21.1 Notable changes

- Clarify package installation. barman is packaged with default python version for each operating system.
- The `minimum-redundancy` option is added to `barman-cloud-backup-delete`. It allows to set the minimum number of backups that should always be available.
- Add a new `primary_checkpoint_timeout` configuration option. Allows define the amount of seconds that Barman will wait at the end of a backup if no new WAL files are produced, before forcing a checkpoint on the primary server.

22.21.2 Bugfixes

- Fix race condition in barman retention policies application. Backup deletions will now raise a warning if another deletion is in progress for the requested backup.
- Fix `barman-cloud-backup-show` man page installation.

22.22 3.7.0 (2023-07-25)

22.22.1 Notable changes

- Support is added for snapshot backups on AWS using EBS volumes.
- The `--profile` option in the `barman-cloud-*` scripts is renamed `--aws-profile`. The old name is deprecated and will be removed in a future release.
- Backup manifests can now be generated automatically on completion of a backup made with `backup_method = rsync`. This is enabled by setting the `autogenerate_manifest` configuration variable and can be overridden using the `--manifest` and `--no-manifest` CLI options.

22.22.2 Bugfixes

- The `barman-cloud-*` scripts now correctly use continuation tokens to page through objects in AWS S3-compatible object stores. This fixes a bug where `barman-cloud-backup-delete` would only delete the oldest 1000 eligible WALs after backup deletion.
- Minor documentation fixes.

22.23 3.6.0 (2023-06-15)

22.23.1 Notable changes

- PostgreSQL version 10 is no longer supported.
- Support is added for snapshot backups on Microsoft Azure using Managed Disks.
- The `--snapshot-recovery-zone` option is renamed `--gcp-zone` for consistency with other provider-specific options. The old name is deprecated and will be removed in a future release.
- The `snapshot_zone` option and `--snapshot-zone` argument are renamed `gcp_zone` and `--gcp-zone` respectively. The old names are deprecated and will be removed in a future release.
- The `snapshot_gcp_project` option and `--snapshot-gcp-project` argument are renamed to `gcp_project` and `--gcp-project`. The old names are deprecated and will be removed in a future release.

22.23.2 Bugfixes

- Barman will no longer attempt to execute the `replication-status` command for a passive node.
- The `backup_label` is deleted from cloud storage when a snapshot backup is deleted with `barman-cloud-backup-delete`.
- Man pages for the `generate-manifest` and `verify-backup` commands are added.
- Minor documentation fixes.

22.24 3.5.0 (2023-03-29)

22.24.1 Notable changes

- Python 2.7 is no longer supported. The earliest Python version supported is now 3.6.
- The `barman`, `barman-cli` and `barman-cli-cloud` packages for EL7 now require python 3.6 instead of python 2.7. For other supported platforms, Barman packages already require python versions 3.6 or later so packaging is unaffected.
- Support for PostgreSQL 10 will be discontinued in future Barman releases; 3.5.x is the last version of Barman with support for PostgreSQL 10.
- Backups and WALs uploaded to Google Cloud Storage can now be encrypted using a specific KMS key by using the `--kms-key-name` option with `barman-cloud-backup` or `barman-cloud-wal-archive`.
- Backups and WALs uploaded to AWS S3 can now be encrypted using a specific KMS key by using the `--sse-kms-key-id` option with `barman-cloud-backup` or `barman-cloud-wal-archive` along with `--encryption=aws:kms`.
- Two new configuration options are provided which make it possible to limit the rate at which parallel workers are started during backups with `backup_method = rsync` and recoveries. `parallel_jobs_start_batch_size` can be set to limit the amount of parallel workers which will be started in a single batch, and `parallel_jobs_start_batch_period` can be set to define the time in seconds over which a single batch of workers will be started. These can be overridden using the arguments `--jobs-start-batch-size` and `--jobs-start-batch-period` with the `barman backup` and `barman recover` commands.
- A new option `--recovery-conf-filename` is added to `barman recover`. This can be used to change the file to which Barman should write the PostgreSQL recovery options from the default `postgresql.auto.conf` to an alternative location.

22.24.2 Bugfixes

- Fix a bug which prevented `barman-cloud-backup-show` from displaying the backup metadata for backups made with `barman backup` and uploaded by `barman-cloud-backup` as a post-backup hook script.
- Fix a bug where the PostgreSQL connection used to validate backup compression settings was left open until termination of the Barman command.
- Fix an issue which caused rsync-concurrent backups to fail when running for a duration greater than `idle_session_timeout`.
- Fix a bug where the backup name was not saved in the backup metadata if the `--wait` flag was used with `barman backup`.
- Thanks to mojtabash78, mhkarimi1383, epolkerman, barthisrael and hzetters for their contributions.

22.25 3.4.0 (2023-01-26)

22.25.1 Notable changes

- This is the last release of Barman which will support Python 2 and new features will henceforth require Python 3.6 or later.
- A new `backup_method` named `snapshot` is added. This will create backups by taking snapshots of cloud storage volumes. Currently only Google Cloud Platform is supported however support for AWS and Azure will follow in future Barman releases. Note that this feature requires a minimum Python version of 3.7. Please see the Barman manual for more information.
- Support for snapshot backups is also added to `barman-cloud-backup`, with minimal support for restoring a snapshot backup added to `barman-cloud-restore`.
- A new command `barman-cloud-backup-show` is added which displays backup metadata stored in cloud object storage and is analogous to `barman show-backup`. This is provided so that snapshot metadata can be easily retrieved at restore time however it is also a convenient way of inspecting metadata for any backup made with `barman-cloud-backup`.
- The instructions for installing Barman from RPMs in the docs are updated.
- The formatting of NFS requirements in the docs is fixed.
- Supported PostgreSQL versions are updated in the docs (this is a documentation fix only - the minimum supported major version is still 10).

22.26 3.3.0 (2022-12-14)

22.26.1 Notable changes

- A backup can now be given a name at backup time using the new `--name` option supported by the `barman backup` and `barman-cloud-backup` commands. The backup name can then be used in place of the backup ID when running commands to interact with backups. Additionally, the commands to list and show backups have been been updated to include the backup name in the plain text and JSON output formats.
- Stricter checking of PostgreSQL version to verify that Barman is running against a supported version of PostgreSQL.

22.26.2 Bugfixes

- Fix inconsistencies between the barman cloud command docs and the help output for those commands.
- Use a new PostgreSQL connection when switching WALs on the primary during the backup of a standby to avoid undefined behaviour such as `SSL error` messages and failed connections.
- Reduce log volume by changing the default log level of stdout for commands executed in child processes to `DEBUG` (with the exception of `pg_basebackup` which is deliberately logged at `INFO` level due to it being a long-running process where it is frequently useful to see the output during the execution of the command).

22.27 3.2.0 (2022-10-20)

22.27.1 Notable changes

- `barman-cloud-backup-delete` now accepts a `--batch-size` option which determines the maximum number of objects deleted in a single request.
- All `barman-cloud-*` commands now accept a `--read-timeout` option which, when used with the `aws-s3` cloud provider, determines the read timeout used by the boto3 library when making requests to S3.

22.27.2 Bugfixes

- Fix the failure of `barman recover` in cases where `backup_compression` is set in the Barman configuration but the PostgreSQL server is unavailable.

22.28 3.1.0 (2022-09-14)

22.28.1 Notable changes

- Backups taken with `backup_method = postgres` can now be compressed using lz4 and zstd compression by setting `backup_compression = lz4` or `backup_compression = zstd` respectively. These options are only supported with PostgreSQL 15 (beta) or later.
- A new option `backup_compression_workers` is available which sets the number of threads used for parallel compression. This is currently only available with `backup_method = postgres` and `backup_compression = zstd`.
- A new option `primary_conninfo` can be set to avoid the need for backups of standbys to wait for a WAL switch to occur on the primary when finalizing the backup. Barman will use the connection string in `primary_conninfo` to perform WAL switches on the primary when stopping the backup.
- Support for certain Rsync versions patched for CVE-2022-29154 which require a trailing newline in the `--files-from` argument.
- Allow `barman receive-wal` maintenance options (`--stop`, `--reset`, `--drop-slot` and `--create-slot`) to run against inactive servers.
- Add `--port` option to `barman-wal-archive` and `barman-wal-restore` commands so that a custom SSH port can be used without requiring any SSH configuration.
- Various documentation improvements.
- Python 3.5 is no longer supported.

22.28.2 Bugfixes

- Ensure PostgreSQL connections are closed cleanly during the execution of `barman cron`.
- `barman generate-manifest` now treats pre-existing backup_manifest files as an error condition.
- backup_manifest files are renamed by appending the backup ID during recovery operations to prevent future backups including an old backup_manifest file.
- Fix epoch timestamps in json output which were not timezone-aware.
- The output of `pg_basebackup` is now written to the Barman log file while the backup is in progress.
- We thank barthisrael, elhananjair, kraynopp, lucianobotti, and mxey for their contributions to this release.

22.29 3.0.1 (2022-06-27)

22.29.1 Bugfixes

- Fix package signing issue in PyPI (same sources as 3.0.0)

22.30 3.0.0 (2022-06-23)

22.30.1 Breaking changes

- PostgreSQL versions 9.6 and earlier are no longer supported. If you are using one of these versions you will need to use an earlier version of Barman.
- The default backup mode for Rsync backups is now concurrent rather than exclusive. Exclusive backups have been deprecated since PostgreSQL 9.6 and have been removed in PostgreSQL

  15. If you are running Barman against PostgreSQL versions earlier than 15 and want to use exclusive backups you will now need to set `exclusive_backup` in `backup_options`.

- The backup metadata stored in the `backup.info` file for each backup has an extra field. This means that earlier versions of Barman will not work in the presence of any backups taken with 3.0.0. Additionally, users of pg-backup-api will need to upgrade it to version 0.2.0 so that pg-backup-api can work with the updated metadata.

22.30.2 Notable changes

- Backups taken with `backup_method = postgres` can now be compressed by pg_basebackup by setting the `backup_compression` config option. Additional options are provided to control the compression level, the backup format and whether the pg_basebackup client or the PostgreSQL server applies the compression. NOTE: Recovery of these backups requires Barman to stage the compressed files on the recovery server in a location specified by the `recovery_staging_path` option.
- Add support for PostgreSQL 15. Exclusive backups are not supported by PostgreSQL 15 therefore Barman configurations for PostgreSQL 15 servers are not allowed to specify `exclusive_backup` in `backup_options`.
- Use custom_compression_magic, if set, when identifying compressed WAL files. This allows Barman to correctly identify uncompressed WALs (such as `*.partial` files in the `streaming` directory) and return them instead of attempting to decompress them.

22.30.3 Minor changes

- Various documentation improvements.

22.30.4 Bugfixes

- Fix an ordering bug which caused Barman to log the message "Backup failed issuing start backup command." while handling a failure in the stop backup command.
- Fix a bug which prevented recovery using `--target-tli` when timelines greater than 9 were present, due to hexadecimal values from WAL segment names being parsed as base 10 integers.
- Fix an import error which occurs when using barman cloud with certain python2 installations due to issues with the enum34 dependency.
- Fix a bug where Barman would not read more than three bytes from a compressed WAL when attempting to identify the magic bytes. This means that any custom compressed WALs using magic longer than three bytes are now decompressed correctly.
- Fix a bug which caused the `--immediate-checkpoint` flag to be ignored during backups with `backup_method = rsync`.

22.31 2.19 (2022-03-09)

22.31.1 Notable changes

- Change `barman diagnose` output date format to ISO8601.
- Add Google Cloud Storage (GCS) support to barman cloud.
- Support `current` and `latest` recovery targets for the `--target-tli` option of `barman recover`.
- Add documentation for installation on SLES.

22.31.2 Bugfixes

- `barman-wal-archive --test` now returns a non-zero exit code when an error occurs.
- Fix `barman-cloud-check-wal-archive` behaviour when `-t` option is used so that it exits after connectivity test.
- `barman recover` now continues when `--no-get-wal` is used and `"get-wal"` is not set in `recovery_options`.
- Fix `barman show-servers --format=json ${server}` output for inactive server.
- Check for presence of `barman_home` in configuration file.
- Passive barman servers will no longer store two copies of the tablespace data when syncing backups taken with `backup_method = postgres`.
- We thank richyen for his contributions to this release.

22.32 2.18 (2022-01-21)

22.32.1 Notable changes

- Add snappy compression algorithm support in barman cloud (requires the optional python-snappy dependency).
- Allow Azure client concurrency parameters to be set when uploading WALs with barman-cloud-wal-archive.

- Add `--tags` option in barman cloud so that backup files and archived WALs can be tagged in cloud storage (aws and azure).
- Update the barman cloud exit status codes so that there is a dedicated code (2) for connectivity errors.
- Add the commands `barman verify-backup` and `barman generate-manifest` to check if a backup is valid.
- Add support for Azure Managed Identity auth in barman cloud which can be enabled with the `--credential` option.

22.32.2 Bugfixes

- Change `barman-cloud-check-wal-archive` behavior when bucket does not exist.
- Ensure `list-files` output is always sorted regardless of the underlying filesystem.
- Man pages for barman-cloud-backup-keep, barman-cloud-backup-delete and barman-cloud-check-wal-archive added to Python packaging.
- We thank richyen and stratakis for their contributions to this release.

22.33 2.17 (2021-12-01)

22.33.1 Notable changes

- Resolves a performance regression introduced in version 2.14 which increased copy times for `barman backup` or `barman recover` commands when using the `--jobs` flag.
- Ignore rsync partial transfer errors for `sender` processes so that such errors do not cause the backup to fail (thanks to barthisrael).

22.34 2.16 (2021-11-17)

22.34.1 Notable changes

- Add the commands `barman-check-wal-archive` and `barman-cloud-check-wal-archive` to validate if a proposed archive location is safe to use for a new PostgreSQL server.
- Allow Barman to identify WAL that' s already compressed using a custom compression scheme to avoid compressing it again.
- Add `last_backup_minimum_size` and `last_wal_maximum_age` options to `barman check`.

22.34.2 Bugfixes

- Use argparse for command line parsing instead of the unmaintained argh module.
- Make timezones consistent for `begin_time` and `end_time`.
- We thank chtitux, George Hansper, stratakis, Thoro, and vrms for their contributions to this release.

22.35 2.15 (2021-10-12)

22.35.1 Notable changes

- Add plural forms for the `list-backup`, `list-server` and `show-server` commands which are now `list-backups`, `list-servers` and `show-servers`. The singular forms are retained for backward compatibility.

- Add the `last-failed` backup shortcut which references the newest failed backup in the catalog so that you can do:
  - `barman delete <SERVER> last-failed`

22.35.2 Bugfixes

- Tablespaces will no longer be omitted from backups of EPAS versions 9.6 and 10 due to an issue detecting the correct version string on older versions of EPAS.

22.36 2.14 (2021-09-22)

22.36.1 Notable changes

- Add the `barman-cloud-backup-delete` command which allows backups in cloud storage to be deleted by specifying either a backup ID or a retention policy.
- Allow backups to be retained beyond any retention policies in force by introducing the ability to tag existing backups as archival backups using `barman keep` and `barman-cloud-backup-keep`.
- Allow the use of SAS authentication tokens created at the restricted blob container level (instead of the wider storage account level) for Azure blob storage
- Significantly speed up `barman restore` into an empty directory for backups that contain hundreds of thousands of files.

22.36.2 Bugfixes

- The backup privileges check will no longer fail if the user lacks "userepl" permissions and will return better error messages if any required permissions are missing (#318 and #319).

22.37 2.13 (2021-07-26)

22.37.1 Notable changes

- Add Azure blob storage support to barman-cloud
- Support tablespace remapping in barman-cloud-restore via `--tablespace name:location`
- Allow barman-cloud-backup and barman-cloud-wal-archive to run as Barman hook scripts, to allow data to be relayed to cloud storage from the Barman server

22.37.2 Bugfixes

- Stop backups failing due to idle_in_transaction_session_timeout https://github.com/EnterpriseDB/barman/issues/333
- Fix a race condition between backup and archive-wal in updating xlog.db entries (#328)
- Handle PGDATA being a symlink in barman-cloud-backup, which led to "seeking backwards is not allowed" errors on restore (#351)
- Recreate pg_wal on restore if the original was a symlink (#327)
- Recreate pg_tblspc symlinks for tablespaces on restore (#343)
- Make barman-cloud-backup-list skip backups it cannot read, e.g., because they are in Glacier storage (#332)

- Add `-d database` option to barman-cloud-backup to specify which database to connect to initially (#307)
- Fix "Backup failed uploading data" errors from barman-cloud-backup on Python 3.8 and above, caused by attempting to pickle the boto3 client (#361)
- Correctly enable server-side encryption in S3 for buckets that do not have encryption enabled by default.

  In Barman 2.12, barman-cloud-backup's `--encryption` option did not correctly enable encryption for the contents of the backup if the backup was stored in an S3 bucket that did not have encryption enabled. If this is the case for you, please consider deleting your old backups and taking new backups with Barman 2.13.

  If your S3 buckets already have encryption enabled by default (which we recommend), this does not affect you.

22.38 2.12.1 (2021-06-30)

22.38.1 Bugfixes

- Allow specifying target-tli with other `target-*` recovery options.
- Fix incorrect NAME in barman-cloud-backup-list manpage.
- Don't raise an error if SIGALRM is ignored.
- Fetch wal_keep_size, not wal_keep_segments, from Postgres 13.

22.39 2.12 (2020-11-05)

22.39.1 Notable changes

- Introduce a new backup_method option called local-rsync which targets those cases where Barman is installed on the same server where PostgreSQL is and directly uses rsync to take base backups, bypassing the SSH layer.

22.39.2 Bugfixes

- Avoid corrupting boto connection in worker processes.
- Avoid connection attempts to PostgreSQL during tests.

22.40 2.11 (2020-07-09)

22.40.1 Notable changes

- Introduction of the barman-cli-cloud package that contains all cloud related utilities.
- Add barman-cloud-wal-restore to restore a WAL file previously archived with barman-cloud-wal-archive from an object store.
- Add barman-cloud-restore to restore a backup previously taken with barman-cloud-backup from an object store.
- Add barman-cloud-backup-list to list backups taken with barman-cloud-backup in an object store.
- Add support for arbitrary archive size for barman-cloud-backup.
- Add support for –endpoint-url option to cloud utilities.
- Remove strict superuser requirement for PG 10+ (by Kaarel Moppel).
- Add –log-level runtime option for barman to override default log level for a specific command.
- Support for PostgreSQL 13

22.40.2 Bugfixes

- Suppress messages and warning with SSH connections in barman-cli (GH-257).
- Fix a race condition when retrieving uploaded parts in barman-cloud-backup (GH-259).
- Close the PostgreSQL connection after a backup (GH-258).
- Check for uninitialized replication slots in receive-wal –reset (GH-260).
- Ensure that begin_wal is valorised before acting on it (GH-262).
- Fix bug in XLOG/WAL arithmetic with custom segment size (GH-287).
- Fix rsync compatibility error with recent rsync.
- Fix PostgreSQLClient version parsing.
- Fix PostgreSQL exception handling with non ASCII messages.
- Ensure each postgres connection has an empty search_path.
- Avoid connecting to PostgreSQL while reading a backup.info file.

If you are using already `barman-cloud-wal-archive` or `barman-cloud-backup` installed via RPM/Apt package and you are upgrading your system, you must install the barman-cli-cloud package. All cloud related tools are now part of the barman-cli-cloud package, including `barman-cloud-wal-archive` and `barman-cloud-backup` that were previously shipped with `barman-cli`. The reason is complex dependency management of the boto3 library, which is a requirement for the cloud utilities.

22.41 2.10 (2019-12-05)

22.41.1 Notable changes

- Pull .partial WAL files with get-wal and barman-wal-restore, allowing restore_command in a recovery scenario to fetch a partial WAL file's content from the Barman server. This feature simplifies and enhances RPO=0 recovery operations.
- Store the PostgreSQL system identifier in the server directory and inside the backup information file. Improve check command to verify the consistency of the system identifier with active connections (standard and replication) and data on disk.
- A new script called barman-cloud-wal-archive has been added to the barman-cli package to directly ship WAL files from PostgreSQL (using archive_command) to cloud object storage services that are compatible with AWS S3. It supports encryption and compression.
- A new script called barman-cloud-backup has been added to the barman-cli package to directly ship base backups from a local PostgreSQL server to cloud object storage services that are compatible with AWS S3. It supports encryption, parallel upload, compression.
- Automated creation of replication slots through the server/global option create_slot. When set to auto, Barman creates the replication slot, in case streaming_archiver is enabled and slot_name is defined. The default value is manual for back-compatibility.
- Add '-w/–wait' option to backup command, making Barman wait for all required WAL files to be archived before considering the backup completed. Add also the –wait-timeout option (default 0, no timeout).
- Redact passwords from Barman output, in particular from barman diagnose (InfoSec)
- Improve robustness of receive-wal –reset command, by verifying that the last partial file is aligned with the current location or, if present, with replication slot's.
- Documentation improvements

22.41.2 Bugfixes

- Wrong string matching operation when excluding tablespaces inside PGDATA (GH-245).
- Minor fixes in WAL delete hook scripts (GH-240).
- Fix PostgreSQL connection aliveness check (GH-239).

22.42 2.9 (2019-08-01)

22.42.1 Notable changes

- Transparently support PostgreSQL 12, by supporting the new way of managing recovery and standby settings through GUC options and signal files (recovery.signal and standby.signal)
- Add –bwlimit command line option to set bandwidth limitation for backup and recover commands
- Ignore WAL archive failure for check command in case the latest backup is WAITING_FOR_WALS
- Add –target-lsn option to set recovery target Log Sequence Number for recover command with PostgreSQL 10 or higher
- Add –spool-dir option to barman-wal-restore so that users can change the spool directory location from the default, avoiding conflicts in case of multiple PostgreSQL instances on the same server (thanks to Drazen Kacar).
- Rename barman_xlog directory to barman_wal
- JSON output writer to export command output as JSON objects and facilitate integration with external tools and systems (thanks to Marcin Onufry Hlybin). Experimental in this release.

22.42.2 Bugfixes

- `replication-status` doesn't show streamers with no slot (GH-222)
- When checking that a connection is alive ("SELECT 1" query), preserve the status of the PostgreSQL connection (GH-149). This fixes those cases of connections that were terminated due to idle-in-transaction timeout, causing concurrent backups to fail.

22.43 2.8 (2019-05-17)

22.43.1 Notable changes

- Add support for reuse_backup in geo-redundancy for incremental backup copy in passive nodes
- Improve performance of rsync based copy by using strptime instead of the more generic dateutil.parser (#210)
- Add '–test' option to barman-wal-archive and barman-wal-restore to verify the connection with the Barman server
- Complain if backup_options is not explicitly set, as the future default value will change from exclusive_backup to concurrent_backup when PostgreSQL 9.5 will be declared EOL by the PGDG
- Display additional settings in the show-server and diagnose commands: archive_timeout, data_checksums, hot_standby, max_wal_senders, max_replication_slots and wal_compression.
- Merge the barman-cli project in Barman

22.43.2 Minor changes

- Improve messaging of check –nagios for inactive servers.
- Log remote SSH command with recover command.
- Hide logical decoding connections in replication-status command.

This release officially supports Python 3 and deprecates Python 2 (which might be discontinued in future releases).

PostgreSQL 9.3 and older is deprecated from this release of Barman. Support for backup from standby is now limited to PostgreSQL 9.4 or higher and to WAL shipping from the standby (please refer to the documentation for details).

22.43.3 Bugfixes

- Fix encoding error in get-wal on Python 3 (Jeff Janes, #221).
- Fix exclude_and_protect_filter (Jeff Janes, #217).
- Remove spurious message when resetting WAL (Jeff Janes, #215).
- Fix sync-wals error if primary has WALs older than the first backup.
- Support for double quotes in synchronous_standby_names setting.

22.44 2.7 (2019-03-12)

22.44.1 Notable changes

- Fix error handling during the parallel backup. Previously an unrecoverable error during the copy could have corrupted the barman internal state, requiring a manual kill of barman process with SIGTERM and a manual cleanup of the running backup in PostgreSQL. (GH#199).
- Fix support of UTF-8 characters in input and output (GH#194 and GH#196).
- Ignore history/backup/partial files for first sync of geo-redundancy (GH#198).
- Fix network failure with geo-redundancy causing cron to break (GH#202).
- Fix backup validation in PostgreSQL older than 9.2.
- Various documentation fixes.

22.45 2.6 (2019-02-04)

22.45.1 Notable changes

- Add support for Geographical redundancy, introducing 3 new commands: sync-info, sync-backup and sync-wals. Geo-redundancy allows a Barman server to use another Barman server as data source instead of a PostgreSQL server.
- Add put-wal command that allows Barman to safely receive WAL files via PostgreSQL’s archive_command using the barman-wal-archive script included in barman-cli.
- Add ANSI colour support to check command.

22.45.2 Bugfixes

- Fix switch-wal on standby with an empty WAL directory.
- Honour archiver locking in wait_for_wal method.
- Fix WAL compression detection algorithm.
- Fix current_action in concurrent stop backup errors.
- Do not treat lock file busy as an error when validating a backup.

22.46 2.5 (2018-10-23)

22.46.1 Notable changes

- Add support for PostgreSQL 11
- Add check-backup command to verify that WAL files required for consistency of a base backup are present in the archive. Barman now adds a new state (WAITING_FOR_WALS) after completing a base backup, and sets it to DONE once it has verified that all WAL files from start to the end of the backup exist. This command is included in the regular cron maintenance job. Barman now notifies users attempting to recover a backup that is in WAITING_FOR_WALS state.
- Allow switch-xlog –archive to work on a standby (just for the archive part)

22.46.2 Bugfixes

- Fix decoding errors reading external commands output (issue #174).
- Fix documentation regarding WAL streaming and backup from standby.

22.47 2.4 (2018-05-25)

22.47.1 Notable changes

- Add standard and retry hook scripts for backup deletion (pre/post).
- Add standard and retry hook scripts for recovery (pre/post).
- Add standard and retry hook scripts for WAL deletion (pre/post).
- Add –standby-mode option to barman recover to add standby_mode = on in pre-generated recovery.conf.
- Add –target-action option to barman recover, allowing users to add shutdown, pause or promote to the pre-generated recovery.conf file.
- Improve usability of point-in-time recovery with consistency checks (e.g. recovery time is after end time of backup).
- Minor documentation improvements.
- Drop support for Python 3.3.

22.47.2 Bugfixes

- Fix remote get_file_content method (GitHub #151), preventing incremental recovery from happening.
- Unicode issues with command (GitHub #143 and #150).
- Add –wal-method=none when pg_basebackup >= 10 (GitHub #133).

- Stop process manager module from overwriting lock files content
- Relax the rules for rsync output parsing
- Ignore vanished files in streaming directory
- Case insensitive slot names (GitHub #170)
- Make DataTransferFailure.from_command_error() more resilient (GitHub #86)
- Rename command() to barman_command() (GitHub #118)
- Initialise synchronous standby names list if not set (GitHub #111)
- Correct placeholders ordering (GitHub #138)
- Force datestyle to iso for replication connections
- Returns error if delete command does not remove the backup
- Fix exception when calling is_power_of_two(None)
- Downgraded sync standby names messages to debug (GitHub #89)

22.48 2.3 (2017-09-05)

22.48.1 Notable changes

- Add support to PostgreSQL 10
- Follow naming changes in PostgreSQL 10:
  - The switch-xlog command has been renamed to switch-wal.
  - In commands output, the xlog word has been changed to WAL and location has been changed to LSN when appropriate.
- Add the –network-compression/–no-network-compression options to barman recover to enable or disable network compression at run-time
- Add –target-immediate option to recover command, in order to exit recovery when a consistent state is reached (end of the backup, available from PostgreSQL 9.4)
- Show cluster state (master or standby) with barman status command
- Documentation improvements

22.48.2 Bugfixes

- Fix high memory usage with parallel_jobs > 1 (#116)
- Better handling of errors using parallel copy (#114)
- Make barman diagnose more robust with system exceptions
- Let archive-wal ignore files with .tmp extension

22.49 2.2 (2017-07-17)

22.49.1 Notable changes

- Implement parallel copy for backup/recovery through the parallel_jobs global/server option to be overridden by the –jobs or -j runtime option for the backup and recover command. Parallel backup is available only for the rsync

copy method. By default, it is set to 1 (for behaviour compatibility with previous versions).

- Support custom WAL size for PostgreSQL 8.4 and newer. At backup time, Barman retrieves from PostgreSQL wal_segment_size and wal_block_size values and computes the necessary calculations.
- Improve check command to ensure that incoming directory is empty when archiver=off, and streaming directory is empty when streaming_archiver=off (#80).
- Add external_configuration to backup_options so that users can instruct Barman to ignore backup of configuration files when they are not inside PGDATA (default for Debian/Ubuntu installations). In this case, Barman does not display a warning anymore.
- Add –get-wal and –no-get-wal options to barman recover
- Add max_incoming_wals_queue global/server option for the check command so that a non blocking error is returned in case incoming WAL directories for both archiver and the streaming_archiver contain more files than the specified value.
- Documentation improvements
- File format changes:
  - The format of backup.info file has changed. For this reason a backup taken with Barman 2.2 cannot be read by a previous version of Barman. But, backups taken by previous versions can be read by Barman 2.2.

22.49.2 Bugfixes

- Allow replication-status to work against a standby
- Close any PostgreSQL connection before starting pg_basebackup (#104, #108)
- Safely handle paths containing special characters
- Archive .partial files after promotion of streaming source
- Recursively create directories during recovery (SF#44)
- Improve xlog.db locking (#99)
- Remove tablespace_map file during recover (#95)
- Reconnect to PostgreSQL if connection drops (SF#82)

22.50 2.1 (2017-01-05)

22.50.1 Notable changes

- Add –archive and –archive-timeout options to switch-xlog command.
- Preliminary support for PostgreSQL 10 (#73).
- Minor additions:
  - Add last archived WAL info to diagnose output.
  - Add start time and execution time to the output of delete command.

22.50.2 Bugfixes

- Return failure for get-wal command on inactive server
- Make streaming_archiver_names and streaming_backup_name options global (#57)
- Fix rsync failures due to files truncated during transfer (#64)

- Correctly handle compressed history files (#66)
- Avoid de-referencing symlinks in pg_tblspc when preparing recovery (#55)
- Fix comparison of last archiving failure (#40, #58)
- Avoid failing recovery if postgresql.conf is not writable (#68)
- Fix output of replication-status command (#56)
- Exclude files from backups like pg_basebackup (#65, #72)
- Exclude directories from other Postgres versions while copying tablespaces (#74)
- Make retry hook script options global

22.51 2.0 (2016-09-27)

22.51.1 Notable changes

- Support for pg_basebackup and base backups over the PostgreSQL streaming replication protocol with backup_method=postgres (PostgreSQL 9.1 or higher required)
- Support for physical replication slots through the slot_name configuration option as well as the –create-slot and –drop-slot options for the receive-wal command (PostgreSQL 9.4 or higher required). When slot_name is specified and streaming_archiver is enabled, receive-wal transparently integrates with pg_receivexlog, and check makes sure that slots exist and are actively used
- Support for the new backup API introduced in PostgreSQL 9.6, which transparently enables concurrent backups and backups from standby servers using the standard rsync method of backup. Concurrent backup was only possible for PostgreSQL 9.2 to 9.5 versions through the pgespresso extension. The new backup API will make pgespresso redundant in the future
- If properly configured, Barman can function as a synchronous standby in terms of WAL streaming. By properly setting the streaming_archiver_name in the synchronous_standby_names priority list on the master, and enabling replication slot support, the receive-wal command can now be part of a PostgreSQL synchronous replication cluster, bringing RPO=0 (PostgreSQL 9.5.5 or higher required)
- Introduce barman-wal-restore, a standard and robust script written in Python that can be used as restore_command in recovery.conf files of any standby server of a cluster. It supports remote parallel fetching of WAL files by efficiently invoking get-wal through SSH. Currently available as a separate project called barman-cli. The barman-cli package is required for remote recovery when get-wal is listed in recovery_options
- Control the maximum execution time of the check command through the check_timeout global/server configuration option (30 seconds by default)
- Limit the number of WAL segments that are processed by an archive-wal run, through the archiver_batch_size and streaming_archiver_batch_size global/server options which control archiving of WAL segments coming from, respectively, the standard archiver and receive-wal
- Removed locking of the XLOG database during check operations
- The show-backup command is now aware of timelines and properly displays which timelines can be used as recovery targets for a given base backup. Internally, Barman is now capable of parsing .history files
- Improved the logic behind the retry mechanism when copy operations experience problems. This involves backup (rsync and postgres) as well as remote recovery (rsync)
- Code refactoring involving remote command and physical copy interfaces

22.51.2 Bugfixes

- Correctly handle .history files from streaming
- Fix replication-status on PostgreSQL 9.1
- Fix replication-status when sent and write locations are not available
- Fix misleading message on pg_receivexlog termination

22.52 1.6.1 (2016-05-23)

22.52.1 Minor changes

- Add –peek option to get-wal command to discover existing WAL files from the Barman's archive
- Add replication-status command for monitoring the status of any streaming replication clients connected to the PostgreSQL server. The –target option allows users to limit the request to only hot standby servers or WAL streaming clients
- Add the switch-xlog command to request a switch of a WAL file to the PostgreSQL server. Through the '–force' it issues a CHECKPOINT beforehand
- Add streaming_archiver_name option, which sets a proper application_name to pg_receivexlog when streaming_archiver is enabled (only for PostgreSQL 9.3 and above)
- Check for *superuser* privileges with PostgreSQL's standard connections (#30)
- Check the WAL archive is never empty
- Check for 'backup_label' on the master when server is down
- Improve barman-wal-restore contrib script

22.52.2 Bugfixes

- Treat the "failed backups" check as non-fatal
- Rename '-x' option for get-wal as '-z'
- Add archive_mode=always support for PostgreSQL 9.5 (#32)
- Properly close PostgreSQL connections when necessary
- Fix receive-wal for pg_receive_xlog version 9.2

22.53 1.6.0 (2016-02-29)

22.53.1 Notable changes

- Support for streaming replication connection through the streaming_conninfo server option
- Support for the streaming_archiver option that allows Barman to receive WAL files through PostgreSQL's native streaming protocol. When set to 'on', it relies on pg_receivexlog to receive WAL data, reducing Recovery Point Objective. Currently, WAL streaming is an additional feature (standard log archiving is still required)
- Implement the receive-wal command that, when streaming_archiver is on, wraps pg_receivexlog for WAL streaming. Add –stop option to stop receiving WAL files via streaming protocol. Add –reset option to reset the streaming status and restart from the current xlog in Postgres.
- Automatic management (startup and stop) of receive-wal command via cron command

- Support for the path_prefix configuration option
- Introduction of the archiver option (currently fixed to on) which enables continuous WAL archiving for a specific server, through log shipping via PostgreSQL's archive_command
- Support for streaming_wals_directory and errors_directory options
- Management of WAL duplicates in archive-wal command and integration with check command
- Verify if pg_receivexlog is running in check command when streaming_archiver is enabled
- Verify if failed backups are present in check command
- Accept compressed WAL files in incoming directory
- Add support for the pigz compressor (thanks to Stefano Zacchiroli zack@upsilon.cc)
- Implement pygzip and pybzip2 compressors (based on an initial idea of Christoph Moench-Tegeder christoph@2ndquadrant.de)
- Creation of an implicit restore point at the end of a backup
- Current size of the PostgreSQL data files in barman status
- Permit archive_mode=always for PostgreSQL 9.5 servers (thanks to Christoph Moench-Tegeder christoph@2ndquadrant.de)
- Complete refactoring of the code responsible for connecting to PostgreSQL
- Improve messaging of cron command regarding sub-processes
- Native support for Python >= 3.3
- Changes of behaviour:
  - Stop trashing WAL files during archive-wal (commit:e3a1d16)

22.53.2 Bugfixes

- Atomic WAL file archiving (#9 and #12)
- Propagate "-c" option to any Barman subprocess (#19)
- Fix management of backup ID during backup deletion (#22)
- Improve archive-wal robustness and log messages (#24)
- Improve error handling in case of missing parameters

22.54 1.5.1 (2015-11-16)

22.54.1 Minor changes

- Add support for the 'archive-wal' command which performs WAL maintenance operations on a given server
- Add support for "per-server" concurrency of the 'cron' command
- Improved management of xlog.db errors
- Add support for mixed compression types in WAL files (SF.net#61)

22.54.2 Bugfixes

- Avoid retention policy checks during the recovery
- Avoid 'wal_level' check on PostgreSQL version < 9.0 (#3)
- Fix backup size calculation (#5)

22.55 1.5.0 (2015-09-28)

22.55.1 Notable changes

- Add support for the get-wal command which allows users to fetch any WAL file from the archive of a specific server
- Add support for retry hook scripts, a special kind of hook scripts that Barman tries to run until they succeed
- Add active configuration option for a server to temporarily disable the server by setting it to False
- Add barman_lock_directory global option to change the location of lock files (by default: 'barman_home')
- Execute the full suite of checks before starting a backup, and skip it in case one or more checks fail
- Forbid to delete a running backup
- Analyse include directives of a PostgreSQL server during backup and recover operations
- Add check for conflicting paths in the configuration of Barman, both intra (by temporarily disabling a server) and inter-server (by refusing any command, to any server).
- Add check for wal_level
- Add barman-wal-restore script to be used as restore_command on a standby server, in conjunction with barman get-wal
- Implement a standard and consistent policy for error management
- Improved cache management of backups
- Improved management of configuration in unit tests
- Tutorial and man page sources have been converted to Markdown format
- Add code documentation through Sphinx
- Complete refactor of the code responsible for managing the backup and the recover commands
- Changed internal directory structure of a backup
- Introduce copy_method option (currently fixed to rsync)

22.55.2 Bugfixes

- Manage options without '=' in PostgreSQL configuration files
- Preserve Timeline history files (Fixes: #70)
- Workaround for rsync on SUSE Linux (Closes: #13 and #26)
- Disables dangerous settings in postgresql.auto.conf (Closes: #68)

22.56 1.4.1 (2015-05-05)

22.56.1 Minor changes

- Improved management of xlogdb file, which is now correctly fsynced when updated. Also, the rebuild-xlogdb command now operates on a temporary new file, which overwrites the main one when finished.
- Add unit tests for dateutil module compatibility
- Modified Barman version following PEP 440 rules and added support of tests in Python 3.4

22.56.2 Bugfixes

- Fix for WAL archival stop working if first backup is EMPTY (Closes: #64)
- Fix exception during error handling in Barman recovery (Closes: #65)
- After a backup, limit cron activity to WAL archiving only (Closes: #62)
- Improved robustness and error reporting of the backup delete command (Closes: #63)
- Fix computation of WAL production ratio as reported in the show-backup command

22.57 1.4.0 (2015-01-26)

22.57.1 Notable changes

- Incremental base backup implementation through the reuse_backup global/server option. Possible values are off (disabled, default), copy (preventing unmodified files from being transferred) and link (allowing for deduplication through hard links).
- Store and show deduplication effects when using reuse_backup= link.
- Added transparent support of pg_stat_archiver (PostgreSQL 9.4) in check, show-server and status commands.
- Improved administration by invoking WAL maintenance at the end of a successful backup.
- Changed the way unused WAL files are trashed, by differentiating between concurrent and exclusive backup cases.
- Improved performance of WAL statistics calculation.
- Treat a missing pg_ident.conf as a WARNING rather than an error.
- Refactored output layer by removing remaining yield calls.
- Check that rsync is in the system path.
- Include history files in WAL management.
- Improved robustness through more unit tests.

22.57.2 Bugfixes

- Fixed bug #55: Ignore fsync EINVAL errors on directories.
- Fixed bug #58: retention policies delete.

22.58 1.3.3 (2014-08-21)

22.58.1 Notable changes

- Added "last_backup_max_age", a new global/server option that allows administrators to set the max age of the last backup in a catalogue, making it easier to detect any issues with periodical backup execution
- Improved robustness of "barman backup" by introducing two global/ server options: "basebackup_retry_times" and "basebackup_retry_sleep". These options allow an administrator to specify, respectively, the number of attempts for a copy operation after a failure, and the number of seconds of wait before retrying
- Improved the recovery process via rsync on an existing directory (incremental recovery), by splitting the previous rsync call into several ones - invoking checksum control only when necessary
- Added support for PostgreSQL 8.3

22.58.2 Minor changes

- Support for comma separated list values configuration options
- Improved backup durability by calling fsync() on backup and WAL files during "barman backup" and "barman cron"
- Improved Nagios output for "barman check –nagios"
- Display compression ratio for WALs in "barman show-backup"
- Correctly handled keyboard interruption (CTRL-C) while performing barman backup
- Improved error messages of failures regarding the stop of a backup
- Wider coverage of unit tests

22.58.3 Bugfixes

- Copies "recovery.conf" on the remote server during "barman recover" (#45)
- Correctly detect pre/post archive hook scripts (#41)

22.59 1.3.2 (2014-04-15)

22.59.1 Bugfixes

- Fixed incompatibility with PostgreSQL 8.4 (Closes #40, bug introduced in version 1.3.1)

22.60 1.3.1 (2014-04-14)

22.60.1 Minor changes

- Added support for concurrent backup of PostgreSQL 9.2 and 9.3 servers that use the "pgespresso" extension. This feature is controlled by the "backup_options" configuration option (global/ server) and activated when set to "concurrent_backup". Concurrent backup allows DBAs to perform full backup operations from a streaming replicated standby.
- Added the "barman diagnose" command which prints important information about the Barman system (extremely useful for support and problem solving)
- Improved error messages and exception handling interface

22.60.2 Bugfixes

- Fixed bug in recovery of tablespaces that are created inside the PGDATA directory (bug introduced in version 1.3.0)
- Fixed minor bug of unhandled -q option, for quiet mode of commands to be used in cron jobs (bug introduced in version 1.3.0)
- Minor bug fixes and code refactoring

22.61 1.3.0 (2014-02-03)

22.61.1 Notable changes

- Refactored BackupInfo class for backup metadata to use the new FieldListFile class (infofile module)
- Refactored output layer to use a dedicated module, in order to facilitate integration with Nagios (NagiosOutputWriter class)
- Refactored subprocess handling in order to isolate stdin/stderr/ stdout channels (command_wrappers module)
- Refactored hook scripts management
- Extracted logging configuration and userid enforcement from the configuration class.
- Support for hook scripts to be executed before and after a WAL file is archived, through the 'pre_archive_script' and 'post_archive_script' configuration options.
- Implemented immediate checkpoint capability with –immediate-checkpoint command option and 'immediate_checkpoint' configuration option
- Implemented network compression for remote backup and recovery through the 'network_compression' configuration option (#19)
- Implemented the 'rebuild-xlogdb' command (Closes #27 and #28)
- Added deduplication of tablespaces located inside the PGDATA directory
- Refactored remote recovery code to work the same way local recovery does, by performing remote directory preparation (assuming the remote user has the right permissions on the remote server)
- 'barman backup' now tries and create server directories before attempting to execute a full backup (#14)

22.61.2 Bugfixes

- Fixed bug #22: improved documentation for tablespaces relocation
- Fixed bug #31: 'barman cron' checks directory permissions for lock file
- Fixed bug #32: xlog.db read access during cron activities

22.62 1.2.3 (2013-09-05)

22.62.1 Minor changes

- Added support for PostgreSQL 9.3
- Added support for the "–target-name" recovery option, which allows to restore to a named point previously specified with pg_create_restore_point (only for PostgreSQL 9.1 and above users)
- Introduced Python 3 compatibility

22.62.2 Bugfixes

- Fixed bug #27 about flock() usage with barman.lockfile (many thanks to Damon Snyder damonsnyder@users.sf.net)

22.63 1.2.2 (2013-06-24)

22.63.1 Bugfixes

- Fix python 2.6 compatibility

22.64 1.2.1 (2013-06-17)

22.64.1 Minor changes

- Added the "bandwidth_limit" global/server option which allows to limit the I/O bandwidth (in KBPS) for backup and recovery operations
- Added the "tablespace_bandwidth_limit" global/server option which allows to limit the I/O bandwidth (in KBPS) for backup and recovery operations on a per tablespace basis
- Added /etc/barman/barman.conf as default location

22.64.2 Bugfixes

- Avoid triggering the minimum_redundancy check on FAILED backups (thanks to Jérôme Vanandruel)

22.65 1.2.0 (2013-01-31)

22.65.1 Notable changes

- Added the "retention_policy_mode" global/server option which defines the method for enforcing retention policies (currently only "auto")
- Added the "minimum_redundancy" global/server option which defines the minimum number of backups to be kept for a server
- Added the "retention_policy" global/server option which defines retention policies management based on redundancy (e.g. REDUNDANCY 4) or recovery window (e.g. RECOVERY WINDOW OF 3 MONTHS)
- Added retention policy support to the logging infrastructure, the "check" and the "status" commands
- The "check" command now integrates minimum redundancy control
- Added retention policy states (valid, obsolete and potentially obsolete) to "show-backup" and "list-backup" commands
- The 'all' keyword is now forbidden as server name
- Added basic support for Nagios plugin output to the 'check' command through the –nagios option
- Barman now requires argh => 0.21.2 and argcomplete-
- Minor bug fixes

22.66 1.1.2 (2012-11-29)

22.66.1 Minor changes

- Added "configuration_files_directory" option that allows to include multiple server configuration files from a directory
- Support for special backup IDs: latest, last, oldest, first
- Management of multiple servers to the 'list-backup' command. 'barman list-backup all' now list backups for all the configured servers.
- Added "application_name" management for PostgreSQL >= 9.0

22.66.2 Bugfixes

- Fixed bug #18: ignore missing WAL files if not found during delete

22.67 1.1.1 (2012-10-16)

22.67.1 Bugfixes

- Fix regressions in recover command.

22.68 1.1.0 (2012-10-12)

22.68.1 Notable changes

- Support for hook scripts to be executed before and after a 'backup' command through the 'pre_backup_script' and 'post_backup_script' configuration options.
- Management of multiple servers to the 'backup' command. 'barman backup all' now iteratively backs up all the configured servers.
- Add warning in recovery when file location options have been defined in the postgresql.conf file (issue #10)
- Fail fast on recover command if the destination directory contains the ':' character (Closes: #4) or if an invalid tablespace relocation rule is passed
- Report an informative message when pg_start_backup() invocation fails because an exclusive backup is already running (Closes: #8)

22.68.2 Bugfixes

- Fixed bug #9: "9.2 issue with pg_tablespace_location()"

22.69 1.0.0 (2012-07-06)

22.69.1 Notable changes

- Backup of multiple PostgreSQL servers, with different versions. Versions from PostgreSQL 8.4+ are supported.
- Support for secure remote backup (through SSH)
- Management of a catalog of backups for every server, allowing users to easily create new backups, delete old ones or restore them

- Compression of WAL files that can be configured on a per server basis using compression/decompression filters, both predefined (gzip and bzip2) or custom
- Support for INI configuration file with global and per-server directives. Default location for configuration files are /etc/barman.conf or ~/.barman.conf. The '-c' option allows users to specify a different one
- Simple indexing of base backups and WAL segments that does not require a local database
- Maintenance mode (invoked through the 'cron' command) which performs ordinary operations such as WAL archival and compression, catalog updates, etc.
- Added the 'backup' command which takes a full physical base backup of the given PostgreSQL server configured in Barman
- Added the 'recover' command which performs local recovery of a given backup, allowing DBAs to specify a point in time. The 'recover' command supports relocation of both the PGDATA directory and, where applicable, the tablespaces
- Added the '–remote-ssh-command' option to the 'recover' command for remote recovery of a backup. Remote recovery does not currently support relocation of tablespaces
- Added the 'list-server' command that lists all the active servers that have been configured in barman
- Added the 'show-server' command that shows the relevant information for a given server, including all configuration options
- Added the 'status' command which shows information about the current state of a server, including Postgres version, current transaction ID, archive command, etc.
- Added the 'check' command which returns 0 if everything Barman needs is functioning correctly
- Added the 'list-backup' command that lists all the available backups for a given server, including size of the base backup and total size of the related WAL segments
- Added the 'show-backup' command that shows the relevant information for a given backup, including time of start, size, number of related WAL segments and their size, etc.
- Added the 'delete' command which removes a backup from the catalog
- Added the 'list-files' command which lists all the files for a single backup
- RPM Package for RHEL 5/6

CHAPTER 23

Glossary

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

Index

A

AWS, 259

B

Barman, 259

D

DBA, 259
DEB, 259

E

External Configuration Files, 259

G

GCP, 259
GPG, 259

I

IAC, 259
ICT, 259

L

libpq, 259

P

PGDATA, 259
PGDG, 260
PITR, 260

R

RHEL, 260
RPM, 260
RPO, 260

S

SLES, 260
SPOF, 260

V

VLDB, 260

W

WORM, 260