Barman

Backup and Recovery Manager for PostgreSQL

version 1.6.0

29 February 2016

Contents

Barman (backup and recovery manager) is an administration tool for disaster recovery of PostgreSQL servers written in Python. Barman can perform remote backups of multiple servers in business critical environments, and helps DBAs during the recovery phase.

Barman's most wanted features include: backup catalogues, incremental backup, retention policies, remote recovery, archiving and compression of WAL files and of backups. Barman is written and maintained by PostgreSQL professionals 2ndQuadrant.

1 Introduction

In a perfect world, there would be no need for a backup. However it is important, especially in 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 repair the incident and recover the database in the shortest possible time. We normally refer to this discipline as **Disaster recovery**.

This guide assumes that you are familiar with theoretical disaster recovery concepts, and you have a grasp of PostgreSQL fundamentals in terms of physical backup and disaster recovery. If not, we encourage you to read the PostgreSQL documentation or any of the recommended books on PostgreSQL.

Professional training on this topic is another effective way of learning these concepts. At any time of the year you can find many courses available all over the world, delivered by PostgreSQL companies such as 2ndQuadrant.

For now, you should be aware that any PostgreSQL physical/binary backup (not to be confused with the logical backups produced by the `pg_dump` utility) is composed of:

- a base backup;
- one or more WAL files (usually collected through continuous archiving).

PostgreSQL offers the core primitives that allow DBAs to setup a really robust Disaster Recovery environment. However, it becomes complicated to manage multiple backups, from one or more PostgreSQL servers. Restoring a given backup is another task that any PostgreSQL DBA would love to see more automated and user friendly.

With these goals in mind, 2ndQuadrant started the development of Barman for PostgreSQL. Barman is an acronym for "Backup and Recovery Manager". Currently Barman works only on Linux and Unix operating systems.

2 Before you start

The first step is to decide the architecture of your backup. In a simple scenario, you have one **PostgreSQL instance** (server) running on a host. You want your data continuously backed up to another server, called the **backup server**.

Barman allows you to launch PostgreSQL backups directly from the backup server, using SSH connections. Furthermore, it allows you to centralise your backups in case you have more than one PostgreSQL server to manage.

During this guide, we will assume that:

- there is one PostgreSQL instance on a host (called `pg` for simplicity)

- there is one backup server on another host (called `backup`)
- communication via SSH between the two servers is enabled
- the PostgreSQL server can be reached from the backup server as the `postgres` operating system user (or another user with PostgreSQL database *superuser* privileges, typically configured via ident authentication)

It is important to note that, for disaster recovery, these two servers must not share any physical resource except for the network. You can use Barman in geographical redundancy scenarios for better disaster recovery outcomes.

2.1 System requirements

- Linux/Unix
- Python 2.6 or 2.7
- Python modules:
  - argcomplete
  - argh >= 0.21.2
  - psycopg2
  - python-dateutil < 2.0 (since version 2.0 requires python3)
  - distribute (optional)
- PostgreSQL >= 8.3
- rsync >= 3.0.4

Important: PostgreSQL's Point-In-Time-Recovery requires the same major version of the source-PostgreSQL server to be installed on the backup server.

Important: Users of RedHat Enterprise Linux, CentOS and Scientific Linux are required to install the Extra Packages Enterprise Linux (EPEL) repository.

Note: Version 1.2.3 of Barman has been refactored for Python 3 support. Please consider it as experimental now and report any bug through the ticketing system on SourceForge or mailing list.

3 Installation

Important: The recommended way to install Barman is by using the available packages for your GNU/Linux distribution.

3.1 On RedHat/CentOS using RPM packages

Barman can be installed on RHEL7, RHEL6 and RHEL5 Linux systems using RPM packages. It is required to install the Extra Packages Enterprise Linux (EPEL) repository beforehand.

RPM packages for Barman are available via Yum through the PostgreSQL Global Development Group RPM repository. You need to follow the instructions for your distribution (RedHat, CentOS, Fedora, etc.) and architecture as detailed at yum.postgresql.org.

Then, as `root` simply type:

```
yum install barman
```

2ndQuadrant also maintains RPM packages for Barman and distributes them through Sourceforge.net.

3.2 On Debian/Ubuntu using packages

Barman can be installed on Debian and Ubuntu Linux systems using packages.

It is directly available in the official repository for Debian Sid (unstable) and Ubuntu 14.04 (Trusty Tahr).

However, the recommended method for installing Barman on Debian and Ubuntu is through the PostgreSQL Community APT repository. Instructions can be found in the APT section of the PostgreSQL Wiki.

> **Note:** Thanks to the direct involvement of Barman developers in the PostgreSQL Community APT repository project, you will have access to the most updated versions of Barman.

Installing Barman is as simple as typing, as `root` user:

```
apt-get install barman
```

3.3 From sources

> **WARNING:** Manual installation of Barman from sources should only be performed by expert GNU/Linux users. Installing Barman this way requires system administration activities such as dependencies management, `barman` user creation, configuration of the `barman.conf` file, cron setup for the `barman cron` command, log management, etc.

Create a system user called `barman` on the `backup` server. As `barman` user, download the sources and uncompress them.

For a system-wide installation, type:

```
barman@backup$ ./setup.py build
# run this command with root privileges or through sudo
barman@backup# ./setup.py install
```

For a local installation, type:

```
barman@backup$ ./setup.py install --user
```

> **Important:** The `--user` option works only with `python-distribute`

`barman` will be installed in your user directory (make sure that your `PATH` environment variable is set properly).

4 Getting started

4.1 Prerequisites

4.1.1 SSH connection

Barman needs a bidirectional SSH connection between the `barman` user on the `backup` server and the `postgres` user. SSH must be configured such that there is no password prompt presented when connecting.

As the `barman` user on the `backup` server, generate an SSH key with an empty password, and append the public key to the `authorized_keys` file of the `postgres` user on the `pg` server.

The `barman` user on the `backup` server should then be able to perform the following operation without typing a password:

```
barman@backup$ ssh postgres@pg
```

The procedure must be repeated with sides swapped in order to allow the `postgres` user on the `pg` server to connect to the `backup` server as the `barman` user without typing a password:

```
postgres@pg$ ssh barman@backup
```

For further information, refer to OpenSSH documentation.

4.1.2 PostgreSQL connection

You need to make sure that the `backup` server allows connection to the PostgreSQL server on `pg` as superuser (`postgres`).

You can choose your favourite client authentication method among those offered by PostgreSQL. More information can be found in the PostgreSQL Documentation.

```
barman@backup$ psql -c 'SELECT version()' -U postgres -h pg
```

> **Note:** As of version 1.1.2, Barman honours the `application_name` connection option for PostgreSQL servers 9.0 or higher.

Streaming connection As of version 1.6.0 Barman enables the connection to a PostgreSQL server using its native streaming replication protocol. In order to set up a streaming connection, you need to:

- Properly configure PostgreSQL to accept streaming replication connections from the Barman server. We encourage users to read the PostgreSQL documentation, in particular:
  - Role attributes
  - The pg_hba.conf file
  - Setting up standby servers using streaming replication
- Set the `streaming_conninfo` parameter in the Barman server configuration accordingly.

  IMPORTANT: Setting up streaming replication is not a task that is strictly related to Barman configuration. Please refer to PostgreSQL documentation, mailing lists, and books for this activity.

4.1.3 Backup directory

Barman needs a main backup directory to store all the backups. Even though you can define a separate folder for each server you want to back up and for each type of resource (backup or WAL segments, for instance), we suggest that you adhere to the default rules and stick with the conventions that Barman chooses for you.

You will see that the configuration file (as explained below) defines a `barman_home` variable, which is the directory where Barman will store all your backups by default. We choose `/var/lib/barman` as home directory for Barman:

```
barman@backup$ sudo mkdir /var/lib/barman
barman@backup$ sudo chown barman:barman /var/lib/barman
```

> **Important:** We assume that you have enough space, and that you have already thought about redundancy and safety of your disks.

4.2 Basic configuration

In the `docs` directory you will find a minimal configuration file. Use it as a template, and copy it to `/etc/barman.conf`, or to `~/.barman.conf`. In general, the former applies to all the users on the `backup` server, while the latter applies only to the `barman` user; for the purpose of this tutorial there is no difference in using one or the other.

From version 1.2.1, you can use `/etc/barman/barman.conf` as default system configuration file.

The configuration file follows the standard INI format, and is split in:

- a section for general configuration (identified by the `barman` label)
- a section for each PostgreSQL server to be backed up (identified by the server label, e.g. `main` or `pg`)[1]

4.2.1 Global/server options

Every option in the configuration file has a *scope*:

- global
- server
- global/server

Global options can be present in the *general section* (identified by `barman`). Server options can only be specified in a *server section*.

Some options can be defined at global level and overridden at server level, allowing users to specify a generic behaviour and refine it for one or more servers. For a list of all the available configurations and their scope, please refer to section 5 of the man page.

```
man 5 barman
```

4.2.2 Configuration files directory

As of version 1.1.2, you can now specify a directory for configuration files similarly to other Linux applications, using the `configuration_files_directory` option (empty by default). If the value of `configuration_files_directory` is a directory, Barman will read all the files with `.conf` extension that exist in that folder. For example, if you set it to `/etc/barman.d`, you can specify your PostgreSQL servers placing each section in a separate `.conf` file inside the `/etc/barman.d` folder.

Otherwise, you can use Barman's standard way of specifying sections within the main configuration file.

4.2.3 Lock files

Since version 1.5.0, Barman allows DBAs to specify a directory for lock files through the `barman_lock_directory` global option.

Lock files are used to coordinate concurrent work at global and server level (for example, cron operations, backup operations, access to the WAL archive, etc.).

By default (for backward compatibility reasons), `barman_lock_directory` is set to `barman_home`.

> **Important:** This change won't affect users upgrading from a version of Barman older than 1.5.0, unless you have written applications that depend on the names of the lock files. However, this is not a typical and common case for Barman and most of users do not fall into this category.

[1] `all` and `barman` are reserved words and cannot be used as server labels.

> **Tip:** Users are encouraged to use a directory in a volatile partition, such as the one dedicated to run-time variable data (e.g. `/var/run/barman`).

4.2.4 Customisation of binary paths

As of version 1.6.0, Barman allows users to specify one or more directories where Barman looks for executable files, using the global/server option `path_prefix`.

If a `path_prefix` is provided, it must contain a list of one or more directories separated by colon. Barman will search inside these directories first, then in those specified by the `PATH` environment variable.

By default the `path_prefix` option is empty.

4.2.5 Example of configuration

Here follows a basic example of PostgreSQL configuration:

```
[barman]
barman_home = /var/lib/barman
barman_user = barman
log_file = /var/log/barman/barman.log
compression = gzip
reuse_backup = link
minimum_redundancy = 1

[main]
description = "Main PostgreSQL Database"
ssh_command = ssh postgres@pg
conninfo = host=pg user=postgres
```

For more detailed information, please refer to the distributed `barman.conf` file.

4.2.6 Initial checks

Once you have created your configuration file (or files), you can now test Barman's configuration by executing:

```
barman@backup$ barman show-server main
barman@backup$ barman check main
```

Write down the `incoming_wals_directory`, as printed by the `barman show-server main` command, because you will need it to setup continuous WAL archiving.

> **Important:** The `barman check main` command automatically creates all the directories for the continuous backup of the `main` server.

4.2.7 Continuous WAL archiving

Barman requires that continuous WAL archiving via PostgreSQL's `archive_command` is properly configured on the master. Edit the `postgresql.conf` file of the PostgreSQL instance on the `pg` database and activate the archive mode:

```
wal_level = 'archive' # For PostgreSQL >= 9.0
archive_mode = on
archive_command = 'rsync -a %p barman@backup:INCOMING_WALS_DIRECTORY/%f'
```

Make sure you change the `INCOMING_WALS_DIRECTORY` placeholder with the value returned by the `barman show-server main` command above.

In case you use Hot Standby, `wal_level` must be set to `hot_standby`.

Restart the PostgreSQL server.

In order to test that continuous archiving is on and properly working, you need to check both the PostgreSQL server and the `backup` server (in particular, that WAL files are correctly collected in the destination directory).

> **Warning:** It is currently a requirement that WAL files from PostgreSQL are shipped to the Barman server. Without `archive_command` being properly set in PostgreSQL to send WAL files to Barman, **full backups cannot be taken**.

> **Important:** PostgreSQL 9.5 introduces support for WAL file archiving using `archive_command` from a standby. This feature is not yet implemented in Barman.

Reducing RPO with WAL streaming From version 1.6.0, Barman improves its Recovery Point Objective (RPO) performance by allowing users to add, on top of the standard `archive_command` strategy, continuous WAL streaming from a PostgreSQL server.

> **Important:** Currently, WAL streaming is only supported in Barman as an additional method for continuous archiving of transaction logs. Future versions of Barman will allow users of PostgreSQL 9.4 and above to rely exclusively on WAL streaming (and abandon the standard `archive_command` method), thanks to *replication slots*. Following our incremental development approach, version 1.6.0 aims to gradually introduce support of this feature in PostgreSQL Disaster Recovery solutions based on Barman. At the same time, users' feedback and production adoption will allow us to improve future versions and implement a "streaming-only" method of WAL archiving that is based on transparent management of replication slots in Barman.

Barman relies on `pg_receivexlog`, a utility that is available from PostgreSQL 9.2 which exploits the native streaming replication protocol and continuously receives transaction logs from a PostgreSQL server (be it a master or a standby).

> **Important:** Barman requires that `pg_receivexlog` is installed in the same server. For PostgreSQL 9.2 servers, you need `pg_receivexlog` of version 9.2 installed alongside with Barman. For PostgreSQL 9.3 and above, it is recommended to install the latest available version of `pg_receivexlog`, as it is back compatible. Otherwise, users can install multiple versions of `pg_receivexlog` in the Barman server and properly point to the specific version for a server, using the `path` option in the configuration file.

In order to enable streaming of transaction logs, you need to:

1. setup a streaming connection, as previously described;
2. set the `streaming_archiver` option to `on`.

The `cron` command, if the aforementioned requirements are met, transparently manages log streaming through the execution of the `receive-wal` command. This is the recommended scenario.

However, users can manually execute the `receive-wal` command:

```
barman receive-wal <server_name>
```

Note: The `receive-wal` command is a foreground process.

Transaction logs are streamed directly in the directory specified by the `streaming_wals_directory` configuration option and are then archived by the `archive-wal` command.

Stopping a receive-wal process for a server If a `receive-wal` process is running in background (e.g. started by the cron command), it is possible to ask barman to stop it by invoking the `receive-wal` command with the `--stop` option:

```
barman receive-wal --stop <server_name>
```

Reset location of receive-wal In some cases, mainly due to the current lack of support for replication slots in Barman, it may be necessary to reset the location of the streaming WAL archiver (e.g.: a prolonged interruption of the `receive-wal` process might cause Barman to go out of sync with the master).

You can reset the location using `--reset` option of the `receive-wal` command, as follows:

```
barman receive-wal --reset <server_name>
```

Note: The `--reset` option requires that no `receive-wal` is running.

4.3 Listing the servers

The following command displays the list of all the available servers:

```
barman@backup$ barman list-server
```

4.4 Executing a full backup

To take a backup for the `main` server, issue the following command:

```
barman@backup$ barman backup main
```

As of version 1.1.0, you can serialise the backup of your managed servers by using the `all` target for the server:

```
barman@backup$ barman backup all
```

This will iterate through your available servers and sequentially take a backup for each of them.

4.4.1 Implicit restore point

As of version 1.5.1, at the end of a successful backup Barman automatically creates a restore point that can be used jointly with `--target-name` during recovery.

By default, the restore point name uses the following convention: `barman_<backup_id>`.

Barman internally uses the PostgreSQL function called `pg_create_restore_point`: for further information, please refer to the PostgreSQL documentation on system administration functions.

Important: This feature is only available for PostgreSQL 9.1 or above.

4.4.2 Immediate Checkpoint

As of version 1.3.0, it is possible to use the `immediate_checkpoint` configuration global/server option (set to `false` by default).

Before starting a backup, Barman requests a checkpoint, which generates additional workload. Normally that checkpoint is throttled according to the settings for workload control on the PostgreSQL server, which means that the backup could be delayed.

If `immediate_checkpoint` is set to `true`, PostgreSQL will not try to limit the workload, and the checkpoint will happen at maximum speed, starting the backup as soon as possible.

At any time, you can override the configuration option behaviour, by issuing `barman backup` with any of these two options:

- `--immediate-checkpoint`, which forces an immediate checkpoint;
- `--no-immediate-checkpoint`, which forces to wait for the checkpoint to happen.

4.5 Viewing the list of backups for a server

To list all the available backups for a given server, issue:

```
barman@backup$ barman list-backup main
```

the format of the output is as in:

```
main - 20120529T092136 - Wed May 30 15:20:25 2012 - Size: 5.0 TiB
 - WAL Size: 845.0 GiB (tablespaces: a:/disk1/a, t:/disk2/t)
```

where `20120529T092136` is the ID of the backup and `Wed May 30 15:20:25 2012` is the start time of the operation, `Size` is the size of the base backup and `WAL Size` is the size of the archived WAL files.

As of version 1.1.2, you can get a listing of the available backups for all your servers, using the `all` target for the server:

```
barman@backup$ barman list-backup all
```

4.6 Restoring a whole server

To restore a whole server issue the following command:

```
barman@backup$ barman recover main 20110920T185953 /path/to/recover/dir
```

where `20110920T185953` is the ID of the backup to be restored. When this command completes successfully, `/path/to/recover/dir` contains a complete data directory ready to be started as a PostgreSQL database server.

Here is an example of a command that starts the server:

```
barman@backup$ pg_ctl -D /path/to/recover/dir start
```

> **Important:** If you run this command as user `barman`, it will become the database superuser.

You can retrieve a list of backup IDs for a specific server with:

```
barman@backup$ barman list-backup srvpgsql
```

> **Important:** Barman does not currently keep track of symbolic links inside PGDATA (except for tablespaces inside `pg_tblspc`). We encourage system administrators to keep track of symbolic links and to add them to the disaster recovery plans/procedures in case they need to be restored in their original location.

4.6.1 Remote recovery

Barman is able to recover a backup on a remote server through the `--remote-ssh-command COMMAND` option for the `recover` command.

If this option is specified, barman uses `COMMAND` to connect to a remote host.

Here is an example of a command that starts recovery on a remote server:

```
barman@backup$ barman recover --remote-ssh-command="ssh user@remotehost" \
main 20110920T185953 /path/to/recover/dir
```

> **Note:** The `postgres` user is normally used to recover on a remote host.

There are some limitations when using remote recovery. It is important to be aware that:

- unless `get-wal` is specified in the `recovery_options` (available from version 1.5.0), Barman requires at least 4GB of free space in the system's temporary directory (usually `/tmp`);
- the SSH connection between Barman and the remote host **must** use public key exchange authentication method;
- the remote user must be able to create the required destination directories for `PGDATA` and, where applicable, tablespaces;
- there must be enough free space on the remote server to contain the base backup and the WAL files needed for recovery.

4.6.2 Relocating one or more tablespaces

> **Important:** As of version 1.3.0, it is possible to relocate a tablespace both with local and remote recovery.

Barman is able to automatically relocate one or more tablespaces using the `recover` command with the `--tablespace` option. The option accepts a pair of values as arguments using the `NAME:DIRECTORY` format:

- name/identifier of the tablespace (`NAME`);
- destination directory (`DIRECTORY`).

If the destination directory does not exists, Barman will try to create it (assuming you have enough privileges).

4.6.3 Restoring to a given point in time

Barman employs PostgreSQL's Point-in-Time Recovery (PITR) by allowing DBAs to specify a recovery target, either as a timestamp or as a transaction ID; you can also specify whether the recovery target should be included or not in the recovery.

The recovery target can be specified using one of three mutually exclusive options:

- `--target-time TARGET_TIME`: to specify a timestamp
- `--target-xid TARGET_XID`: to specify a transaction ID
- `--target-name TARGET_NAME`: to specify a named restore point - previously created with the `pg_create_restore_point(n` function[2]

You can use the `--exclusive` option to specify whether to stop immediately before or immediately after the recovery target.

Barman allows you to specify a target timeline for recovery, using the `target-tli` option. The notion of timeline goes beyond the scope of this document; you can find more details in the PostgreSQL documentation, or in one of 2ndQuadrant's Recovery training courses.

4.6.4 Limitations of partial WAL files with recovery

Version 1.6.0 introduces support for WAL streaming, by integrating PostgreSQL's `pg_receivexlog` utility with Barman. The standard behaviour of `pg_receivexlog` is to write transactional information in a file with `.partial` suffix after the WAL segment name.

Barman expects a partial file to be in the `streaming_wals_directory` of a server. When completed, `pg_receivexlog` removes the `.partial` suffix and opens the following one, delivering the file to the `archive-wal` command of Barman for permanent storage and compression.

In case of a sudden and unrecoverable failure of the master PostgreSQL server, the `.partial` file that has been streamed to Barman contains very important information that the standard archiver (through PostgreSQL's `archive_command`) has not been able to deliver to Barman.

> **Important:** A current limitation of Barman is that the `recover` command is not yet able to transparently manage `.partial` files. In such situations, users will need to manually copy the latest partial file from the server's `streaming_wals_directory` of their Barman installation to the destination for recovery, making sure that the `.partial` suffix is removed. Restoring a server using the last partial file, reduces data loss, by bringing down *recovery point objective* to values around 0.

4.7 Retry of copy in backup/recovery operations

As of version 1.3.3, it is possible to take advantage of two new options in Barman:

- `basebackup_retry_times` (set to 0 by default)
- `basebackup_retry_sleep` (set to 30 by default)

When issuing a backup or a recovery, Barman normally tries to copy the base backup once. If the copy fails (e.g. due to network problems), Barman terminates the operation with a failure.

By setting `basebackup_retry_times`, Barman will try to re-execute a copy operation as many times as requested by the user. The `basebackup_retry_sleep` option specifies the number of seconds that Barman will wait between each attempt.

At any time you can override the configuration option behaviour from the command line, when issuing `barman backup` or `barman recover`, using:

- `--retry-times <retry_number>` (same logic as `basebackup_retry_times`)
- `--no-retry` (same as `--retry-times 0`)
- `--retry-sleep <number_of_seconds>` (same logic as `basebackup_retry_sleep`)

[2] Only available on PostgreSQL 9.1 and above.

5 Available commands

Barman commands are applied to three different levels:

- **general** commands, which apply to the backup catalogue
- **server** commands, which apply to a specific server (list available backups, execute a backup, etc.)
- **backup** commands, which apply to a specific backup in the catalogue (display information, issue a recovery, delete the backup, etc.)

In the following sections the available commands will be described in detail.

5.1 General commands

5.1.1 List available servers

You can display the list of active servers that have been configured for your backup system with:

```
barman list-server
```

5.1.2 Maintenance mode

Cron command You can perform maintenance operations, on both WAL files and backups, using the command:

```
barman cron
```

As of version 1.5.1 `barman cron` executes WAL archiving operations concurrently on a server basis.

This also enforces retention policies on those servers that have:

- `retention_policy` not empty and valid;
- `retention_policy_mode` set to `auto`.

  Note: This command should be executed in a *cron script*. Our recommendation is to schedule `barman cron` to run every minute.

Archive-wal command As of version 1.5.1, Barman introduces the `archive-wal` command:

```
barman archive-wal <server_name>
```

This is the command responsible for WAL maintenance operations, like compressing WAL files and moving them from the *incoming directory* (if `archiver` is enabled) or the *streaming directory* (if `streaming_archiver` is enabled) into the archive.

Although it can be manually executed, the majority of users will not need to do it, given that it is transparently invoked as a subprocess by the `cron` command, as part of the standard maintenance operations for every server.

5.2 Server commands

5.2.1 Show the configuration for a given server

You can show the configuration parameters for a given server with:

```
barman show-server <server_name>
```

5.2.2 Take a base backup

You can perform a full backup (base backup) for a given server with:

```
barman backup [--immediate-checkpoint] <server_name>
```

> **Tip:** You can use `barman backup all` to sequentially backup all your configured servers.

5.2.3 Show available backups for a server

You can list the catalogue of available backups for a given server with:

```
barman list-backup <server_name>
```

5.2.4 Check a server is properly working

You can check if the connection to a given server is properly working with:

```
barman check <server_name>
```

> **Tip:** You can use `barman check all` to check all your configured servers.

From version 1.3.3, you can automatically be notified if the latest backup of a given server is older than, for example, *7 days*.[3]
Barman introduces the option named `last_backup_maximum_age` having the following syntax:

```
last_backup_maximum_age = {value {DAYS | WEEKS | MONTHS}}
```

where `value` is a positive integer representing the number of days, weeks or months of the time frame.

5.2.5 Diagnose a Barman installation

You can gather important information about all the configured server using:

```
barman diagnose
```

The `diagnose` command also provides other useful information, such as global configuration, SSH version, Python version, `rsync` version, as well as current configuration and status of all servers.

> **Tip:** You can use `barman diagnose` when you want to ask questions or report errors to Barman developers, providing them with all the information about your issue.

5.2.6 Rebuild the WAL archive

At any time, you can regenerate the content of the WAL archive for a specific server (or every server, using the `all` shortcut). The WAL archive is contained in the `xlog.db` file, and every Barman server has its own copy. From version 1.2.4 you can now rebuild the `xlog.db` file with the `rebuild-xlogdb` command. This will scan all the archived WAL files and regenerate the metadata for the archive.

[3] This feature is commonly known among the development team members as *smelly backup check*.

> **Important:** Users of Barman < 1.2.3 might have suffered from a bug due to bad locking in highly concurrent environments. You can now regenerate the WAL archive using the `rebuild-xlogdb` command.

```
barman rebuild-xlogdb <server_name>
```

5.2.7 Access WAL archive using `get-wal`

From version 1.5.0, Barman allows users to request any *xlog* file from its WAL archive through the `get-wal` command:

```
barman get-wal [-o OUTPUT_DIRECTORY] [-j|-x] <server_name> <wal_id>
```

If the requested WAL file is found in the server archive, the uncompressed content will be returned to `STDOUT`, unless otherwise specified.

The following options are available for the `get-wal` command:

- `-o` allows users to specify a destination directory where Barman will deposit the requested WAL file
- `-j` will compress the output using `bzip2` algorithm
- `-x` will compress the output using `gzip` algorithm

It is possible to use `get-wal` during a recovery operation, transforming the Barman server in a *WAL hub* for your servers. This can be automatically achieved by adding the `get-wal` value to the `recovery_options` global/server configuration option:

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

`recovery_options` is a global/server option that accepts a list of comma separated values. If the keyword `get-wal` is present, during a recovery operation Barman will prepare the `recovery.conf` file by setting the `restore_command` so that `barman get-wal` is used to fetch the required WAL files.

This is an example of a `restore_command` for a remote recovery:

```
restore_command = 'ssh barman@pgbackup barman get-wal SERVER %f > %p'
```

This is an example of a `restore_command` for a local recovery:

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

> **Important:** Even though `recovery_options` aims to automate the process, using the `get-wal` facility requires manual intervention and proper testing.

5.3 Backup commands

> **Note:** Remember: a backup ID can be retrieved with `barman list-backup <server_name>`

5.3.1 Show backup information

You can show all the available information for a particular backup of a given server with:

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

From version 1.1.2, in order to show the latest backup, you can issue:

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

5.3.2 Delete a backup

You can delete a given backup with:

```
barman delete <server_name> <backup_id>
```

From version 1.1.2, in order to delete the oldest backup, you can issue:

```
barman delete <server_name> oldest
```

5.3.3 List backup files

You can list the files (base backup and required WAL files) for a given backup with:

```
barman list-files [--target TARGET_TYPE] <server_name> <backup_id>
```

With the `--target TARGET_TYPE` option, it is possible to choose the content of the list for a given backup.

Possible values for `TARGET_TYPE` are:

- `data`: lists just the data files;
- `standalone`: lists the base backup files, including required WAL files;
- `wal`: lists all WAL files from the beginning of the base backup to the start of the following one (or until the end of the log);
- `full`: same as `data` + `wal`.

The default value for `TARGET_TYPE` is `standalone`.

> **Important:** The `list-files` command facilitates interaction with external tools, and therefore can be extremely useful to integrate > Barman into your archiving procedures.

6 Main features

6.1 Incremental backup

From version 1.4.0, Barman implements **file-level incremental backup**. Incremental backup is a kind of full periodic backup which saves only data changes from the latest full backup available in the catalogue for a specific PostgreSQL server. It must not be confused with differential backup, which is implemented by *WAL continuous archiving*.

The main goals of incremental backup in Barman are:

- Reduce the time taken for the full backup process
- Reduce the disk space occupied by several periodic backups (**data deduplication**)

This feature heavily relies on `rsync` and hard links, which must be therefore supported by both the underlying operating system and the file system where the backup data resides.

The main concept is that a subsequent base backup will share those files that have not changed since the previous backup, leading to relevant savings in disk usage. This is particularly true of VLDB contexts and, more in general, of those databases containing a high percentage of *read-only historical tables*.

Barman implements incremental backup through a global/server option, called `reuse_backup`, that transparently manages the `barman backup` command. It accepts three values:

- `off`: standard full backup (default)
- `link`: 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)
- `copy`: incremental backup, by reusing the last backup for a server and creating a copy of the unchanged files (just for backup time reduction)

The most common scenario is to set `reuse_backup` to `link`, as follows:

```
reuse_backup = link
```

Setting this at global level will automatically enable incremental backup for all your servers.

As a final note, users can override the setting of the `reuse_backup` option through the `--reuse-backup` runtime option for the `barman backup` command. Similarly, the runtime option accepts three values: `off`, `link` and `copy`. For example, you can run a one-off incremental backup as follows:

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

6.2 WAL compression

The `barman cron` command (see below) will compress WAL files if the `compression` option is set in the configuration file. This option allows five values:

- `bzip2`: for Bzip2 compression (requires the `bzip2` utility)
- `gzip`: for Gzip compression (requires the `gzip` utility)
- `pybzip2`: for Bzip2 compression (uses Python's internal compression module)
- `pygzip`: for Gzip compression (uses Python's internal compression module)
- `pigz`: for Pigz compression (requires the `pigz` utility)
- `custom`: 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

  NOTE: The `pybzip2`, `pygzip` and `pigz` options for standard compression have been introduced in Barman 1.6.0. All methods but `pybzip2` and `pygzip` require `barman archive-wal` to fork a new process.

6.3 Limiting bandwidth usage

From version 1.2.1, it is possible to limit the usage of I/O bandwidth through the `bandwidth_limit` option (global/per server), by specifying the maximum number of kilobytes per second. By default it is set to 0, meaning no limit.

In case you have several tablespaces and you prefer to limit the I/O workload of your backup procedures on one or more tablespaces, you can use the `tablespace_bandwidth_limit` option (global/per server):

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

The option accepts a comma separated list of pairs made up of the tablespace name and the bandwidth limit (in kilobytes per second).

When backing up a server, Barman will try and locate any existing tablespace in the above option. If found, the specified bandwidth limit will be enforced. If not, the default bandwidth limit for that server will be applied.

6.4 Network Compression

From version 1.3.0 it is possible to reduce the size of transferred data using compression. It can be enabled using the `network_compression` option (global/per server):

```
network_compression = true|false
```

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

6.5 Backup ID shortcuts

As of version 1.1.2, you can use any of the following **shortcuts** to identify a particular backup for a given server:

- `latest`: the latest available backup for that server, in chronological order. You can also use the `last` synonym.
- `oldest`: the oldest available backup for that server, in chronological order. You can also use the `first` synonym.

These aliases can be used with any of the following commands: `show-backup`, `delete`, `list-files` and `recover`.

6.6 Minimum redundancy safety

From version 1.2.0, you can define the minimum number of periodic backups for a PostgreSQL server.

You can use the global/per server configuration option called `minimum_redundancy` for this purpose, by default set to 0.

By setting this value to any number greater than 0, Barman makes sure that at any time you will have at least that number of backups in a server catalogue.

This will protect you from accidental `barman delete` operations.

> **Important:** Make sure that your policy retention settings do not collide with minimum redundancy requirements. Regularly check Barman's log for messages on this topic.

6.7 Retention policies

From version 1.2.0, Barman supports **retention policies** for backups.

A backup retention policy is an user-defined policy that determines how long backups and related archive logs (Write Ahead Log segments) need to be retained for recovery procedures.

Based on the user's request, Barman retains the periodic backups required to satisfy the current retention policy, and any archived WAL files required for the complete recovery of those backups.

Barman users can define a retention policy in terms of **backup redundancy** (how many periodic backups) or a **recovery window** (how long).

Retention policy based on redundancy In a redundancy based retention policy, the user determines how many periodic backups to keep. A redundancy-based retention policy is contrasted with retention policies that use a recovery window.

Retention policy based on recovery window A recovery window is one type of Barman backup retention policy, in which the DBA specifies a period of time and Barman ensures retention of backups and/or archived WAL files required for point-in-time recovery to any time during the recovery window. The interval always ends with the current time and extends back in time for the number of days specified by the user. For example, if the retention policy is set for a recovery window of seven days, and the current time is 9:30 AM on Friday, Barman retains the backups required to allow point-in-time recovery back to 9:30 AM on the previous Friday.

6.7.1 Scope

Retention policies can be defined for:

- **PostgreSQL periodic base backups**: through the `retention_policy` configuration option;
- **Archive logs**, for Point-In-Time-Recovery: through the `wal_retention_policy` configuration option.

  Important: In a temporal dimension, archive logs must be included in the time window of periodic backups.

There are two typical use cases here: full or partial point-in-time recovery.

Full point in time recovery scenario Base backups and archive logs share the same retention policy, allowing DBAs to recover at any point in time from the first available backup.

Partial point in time recovery scenario Base backup retention policy is wider than that of archive logs, allowing users for example to keep full weekly backups of the last 6 months, but archive logs for the last 4 weeks (granting to recover at any point in time starting from the last 4 periodic weekly backups).

Important: Currently, Barman implements only the **full point in time recovery** scenario, by constraining the `wal_retention_policy` option to `main`.

6.7.2 How they work

Retention policies in Barman can be:

- **automated**: enforced by `barman cron`;
- **manual**: Barman simply reports obsolete backups and allows DBAs to delete them.

  Important: Currently Barman does not implement manual enforcement. This feature will be available in future versions.

6.7.3 Configuration and syntax

Retention policies can be defined through the following configuration options:

- `retention_policy`: for base backup retention;
- `wal_retention_policy`: for archive logs retention;
- `retention_policy_mode`: can only be set to `auto` (retention policies are automatically enforced by the `barman cron` command).

These configuration options can be defined both at a global level and a server level, allowing users maximum flexibility on a multi-server environment.

Syntax for `retention_policy` The general syntax for a base backup retention policy through `retention_policy` is the following:

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

Where:

- syntax is case insensitive;
- `value` is an integer and is > 0;
- in case of **redundancy retention policy**:
  - `value` must be greater than or equal to the server minimum redundancy level (if not is is assigned to that value and a warning is generated);
  - the first valid backup is the value-th backup in a reverse ordered time series;
- in case of **recovery window policy**:
  - the point of recoverability is: current time - window;
  - the first valid backup is the first available backup before the point of recoverability; its value in a reverse ordered time series must be greater than or equal to the server minimum redundancy level (if not is is assigned to that value and a warning is generated).

By default, `retention_policy` is empty (no retention enforced).

Syntax for `wal_retention_policy` Currently, the only allowed value for `wal_retention_policy` is the special value `main`, that maps the retention policy of archive logs to that of base backups.

6.8 Concurrent Backup and backup from a standby

Normally, during backup operations, Barman uses PostgreSQL native functions `pg_start_backup` and `pg_stop_backup` for *exclusive backup*. These operations are not allowed on a read-only standby server.

As of version 1.3.1, Barman is also capable of performing backups of PostgreSQL 9.2/9.3 database servers in a **concurrent way**, primarily through the `backup_options` configuration parameter.[4]

This introduces a new architecture scenario with Barman: **backup from a standby server**, using `rsync`.

> **Important: Concurrent backup** requires users of PostgreSQL 9.2 and 9.3 to install the `pgespresso` open source extension on the PostgreSQL server. For more detailed information and the source code, please visit the pgespresso extension website.

By default, `backup_options` is transparently set to `exclusive_backup` (the only supported method by any Barman version prior to 1.3.1).

When `backup_options` is set to `concurrent_backup`, Barman activates the *concurrent backup mode* for a server and follows these two simple rules:

- `ssh_command` must point to the destination Postgres server;
- `conninfo` must point to a database on the destination Postgres 9.2 or 9.3 server where `pgespresso` is correctly installed through `CREATE EXTENSION`.

[4] Concurrent backup is a technology that has been available in PostgreSQL since version 9.1, through the *streaming replication protocol* (using, for example, a tool like `pg_basebackup`).

The destination Postgres server can be either the master or a streaming replicated standby server.

> **Note:** When backing up from a standby server, continuous archiving of WAL files must be configured on the master to ship files to the Barman server (as outlined in the "Continuous WAL archiving" section above)[5].

6.9 Hook scripts

Barman allows a database administrator to run *hook scripts* on these two events:

- before and after a backup
- before and after a WAL file is archived

There are two types of hook scripts that Barman can manage:

- standard hook scripts (already present in Barman since version 1.1.0)
- retry hook scripts, introduced in version 1.5.0

The only difference between these two types of hook scripts is that Barman executes a standard hook script only once, without checking its return code, whereas a retry hook script may be executed more than once depending on its return code.

Precisely, when executing a retry hook script, Barman checks the return code and retries indefinitely until the script returns either `SUCCESS` (with standard return code `0`), or `ABORT_CONTINUE` (return code `62`), or `ABORT_STOP` (return code `63`). Barman treats any other return code as a transient failure to be retried. Users are given more power: a hook script can control its workflow by specifying whether a failure is transient. Also, in case of a 'pre' hook script, by returning `ABORT_STOP`, users can request Barman to interrupt the main operation with a failure.

Hook scripts are executed in the following order:

1. The standard 'pre' hook script (if present)
2. The retry 'pre' hook script (if present)
3. The actual event (i.e. backup operation, or WAL archiving), if retry 'pre' hook script was not aborted with `ABORT_STOP`
4. The retry 'post' hook script (if present)
5. The standard 'post' hook script (if present)

The output generated by any hook script is written in the log file of Barman.

> **Note:** Currently, `ABORT_STOP` is ignored by retry 'post' hook scripts. In these cases, apart from lodging an additional warning, `ABORT_STOP` will behave like `ABORT_CONTINUE`.

6.9.1 Backup scripts

Version 1.1.0 introduced backup scripts.

These scripts can be configured with the following global configuration options (which can be overridden on a per server basis):

[5] In case of concurrent backup, currently Barman does not have a way to determine that the closing WAL file of a full backup has actually been shipped - opposite to the case of an exclusive backup where it is Postgres itself that makes sure that the WAL file is correctly archived. Be aware that the full backup cannot be considered consistent until that WAL file has been received and archived by Barman. We encourage Barman users to wait to delete the previous backup - at least until that moment.

- `pre_backup_script`: *hook script* executed *before* a base backup, only once, with no check on the exit code
- `pre_backup_retry_script`: *retry hook script* executed *before* a base backup, repeatedly until success or abort
- `post_backup_retry_script`: *retry hook script* executed *after* a base backup, repeatedly until success or abort
- `post_backup_script`: *hook script* executed *after* a base backup, only once, with no check on the exit code

The script definition is passed to a shell and can return any exit code. Only in case of a *retry* script, Barman checks the return code (see the upper section).

The shell environment will contain the following variables:

- `BARMAN_BACKUP_DIR`: backup destination directory
- `BARMAN_BACKUP_ID`: ID of the backup
- `BARMAN_CONFIGURATION`: configuration file used by barman
- `BARMAN_ERROR`: error message, if any (only for the `post` phase)
- `BARMAN_PHASE`: phase of the script, either `pre` or `post`
- `BARMAN_PREVIOUS_ID`: ID of the previous backup (if present)
- `BARMAN_RETRY`: 1 if it is a retry script (from 1.5.0), `0` if not
- `BARMAN_SERVER`: name of the server
- `BARMAN_STATUS`: status of the backup
- `BARMAN_VERSION`: version of Barman (from 1.2.1)

6.9.2 WAL archive scripts

Version 1.3.0 introduced WAL archive hook scripts.

Similarly to backup scripts, archive scripts can be configured with global configuration options (which can be overridden on a per server basis):

- `pre_archive_script`: *hook script* executed *before* a WAL file is archived by maintenance (usually `barman cron`), only once, with no check on the exit code
- `pre_archive_retry_script`: *retry hook script* executed *before* a WAL file is archived by maintenance (usually `barman cron`), repeatedly until success or abort
- `post_archive_retry_script`: *retry hook script* executed *after* a WAL file is archived by maintenance, repeatedly until success or abort
- `post_archive_script`: *hook script* executed *after* a WAL file is archived by maintenance, only once, with no check on the exit code

The script is executed through a shell and can return any exit code. Only in case of a *retry* script, Barman checks the return code (see the upper section).

Archive scripts share with backup scripts some environmental variables:

- `BARMAN_CONFIGURATION`: configuration file used by barman
- `BARMAN_ERROR`: error message, if any (only for the `post` phase)
- `BARMAN_PHASE`: phase of the script, either `pre` or `post`
- `BARMAN_SERVER`: name of the server

Following variables are specific to archive scripts:

- `BARMAN_SEGMENT`: name of the WAL file

- `BARMAN_FILE`: full path of the WAL file
- `BARMAN_SIZE`: size of the WAL file
- `BARMAN_TIMESTAMP`: WAL file timestamp
- `BARMAN_COMPRESSION`: type of compression used for the WAL file

6.10 Integration with standby servers

Barman has been designed for integration with standby servers (with streaming replication or traditional file based log shipping) and high availability tools like repmgr.

From an architectural point of view, PostgreSQL must be configured to archive WAL files directly to the Barman server.

7 Support and sponsor opportunities

Barman is free software, written and maintained by 2ndQuadrant. If you require support on using Barman, or if you need new features, please get in touch with 2ndQuadrant. You can sponsor the development of new features of Barman and PostgreSQL which will be made publicly available as open source.

For further information, please visit:

- Barman website
- Support section
- 2ndQuadrant website
- FAQs
- 2ndQuadrant blog

> **Important:** When submitting requests on the mailing list, please always report the output of the `barman diagnose` command.

8 Submitting a bug

Barman has been extensively tested, and is currently being used in several production environments. However, as any software, Barman is not bug free.

If you discover a bug, please follow this procedure:

- execute the `barman diagnose` command;
- file a bug through the Sourceforge bug tracker, by attaching the output obtained by the diagnostics command above (`barman diagnose`).

9 Contributing to Barman

2ndQuadrant has a team of software engineers, architects, database administrators, system administrators, QA engineers, developers and managers that dedicate their time and expertise to improve Barman's code. We adopt lean and agile methodologies for software development, and we believe in the *devops* culture that allowed us to implement rigorous testing procedures through cross-functional collaboration. Every Barman commit is the contribution of multiple individuals, at different stages of the production pipeline.

Even though this is our preferred way of developing Barman, we gladly accept patches from external developers, as long as:

- user documentation (tutorial and man pages) is provided;
- source code is properly documented and contains relevant comments;
- code supplied is covered by unit tests;
- no unrelated feature is compromised or broken;
- source code is rebased on the current master branch;
- commits and pull requests are limited to a single feature (multi-feature patches are hard to test and review);
- changes to the user interface are discussed beforehand with 2ndQuadrant.

We also require that any contributions provide a copyright assignment and a disclaimer of any work-for-hire ownership claims from the employer of the developer.

You can use Github's pull requests system for this purpose.

10 Authors

In alphabetical order:

- Gabriele Bartolini gabriele.bartolini@2ndquadrant.it (project leader)
- Stefano Bianucci stefano.bianucci@2ndquadrant.it (developer)
- Giuseppe Broccolo giuseppe.broccolo@2ndquadrant.it (QA/testing)
- Giulio Calacoci giulio.calacoci@2ndquadrant.it (developer)
- Francesco Canovai francesco.canovai@2ndquadrant.it (QA/testing)
- Leonardo Cecchi leonardo.cecchi@2ndquadrant.it (developer)
- Gianni Ciolli gianni.ciolli@2ndquadrant.it (QA/testing)
- Marco Nenciarini marco.nenciarini@2ndquadrant.it (lead developer)

Past contributors:

- Carlo Ascani

11 Links

- check-barman: a Nagios plugin for Barman, written by Holger Hamann (MIT license)
- puppet-barman: Barman module for Puppet (GPL)
- Tutorial on "How To Back Up, Restore, and Migrate PostgreSQL Databases with Barman on CentOS 7", by Sadequl Hussain (available on DigitalOcean Community)

12 License and Contributions

Barman is the exclusive property of 2ndQuadrant Italia and its code is distributed under GNU General Public License 3.

Barman has been partially funded through 4CaaSt, a research project funded by the European Commission's Seventh Framework programme.

Contributions to Barman are welcome, and will be listed in the `AUTHORS` file. 2ndQuadrant Italia requires that any contributions provide a copyright assignment and a disclaimer of any work-for-hire ownership claims from the employer of the developer. This lets us make sure that all of the Barman distribution remains free code. Please contact info@2ndQuadrant.it for a copy of the relevant Copyright Assignment Form.