2ndQuadrant
Professional PostgreSQL

Barman

Backup and Recovery Manager for PostgreSQL Tutorial

21 August 2014

Gabriele Bartolini
Marco Nenciarini
Giulio Calacoci
Gianni Ciolli
Giuseppe Broccolo

Revision History		
Revision 1.3.3	21 August 2014	gb
Version 1.3.3		
Revision 1.3.3-alpha1	27 June 2014	gb, mn
Version 1.3.3-alpha1		
Revision 1.3.2	15 April 2014	gb, mn
Version 1.3.2		
Revision 1.3.1	14 April 2014	gb
Version 1.3.1		
Revision 1.3.0	3 February 2014	gb
Version 1.3.0		
Revision 1.2.3	5 September 2013	gib
Version 1.2.3		
Revision 1.2.2	24 June 2013	mn
Version 1.2.2		
Revision 1.2.1	17 June 2013	gb
Version 1.2.1		
Revision 1.2.0	31 January 2013	gb
Version 1.2.0		
Revision 1.1.2	29 November 2012	gb
Version 1.1.2		
Revision 1.1.0	12 October 2012	gb, mn
Version 1.1.0		
Revision 1.0.0	2 July 2012	gb, ca, mn, gc
First public release		

Table of 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, retention policies, remote recovery, archiving and compression of WAL files and of backups. Barman is written and maintained by PostgreSQL professionals 2ndQuadrant.

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.

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.

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

The same major version of PostgreSQL should be installed on both servers.

Tip

Users of RedHat Enterprise Linux, CentOS and Scientific Linux are advised to install the **Extra Packages Enterprise Linux** (EPEL) repository. [1]

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.

[1] Further information at http://fedoraproject.org/wiki/EPEL

Installation

On RedHat/CentOS using RPM packages

Barman can be installed on RHEL5 and RHEL6 Linux systems using RPM packages.

Barman is available through the PostgreSQL Global Development Group RPM repository with Yum. You need to follow the instructions for your distribution (RedHat, CentOS, Fedora, etc.) and architecture that you can find at http://yum.postgresql.org/.

Then, as `root` simply type:

```
yum install barman
```

On Debian/Ubuntu using packages

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

It is available in the official repository for Debian **Sid** (unstable) and Ubuntu 12.10 (Quantal Quetzal).

> ### Note
>
> You can install an up-to-date version of Barman on many Debian and Ubuntu releases using the PostgreSQL Community APT repository at http://apt.postgresql.org/. Instructions can be found at https://wiki.postgresql.org/wiki/Apt.

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

```
apt-get install barman
```

From sources

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

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).

Getting started

Prerequisites

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. on the `pg` server. 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.

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 here: http://www.postgresql.org/docs/current/static/client-authentication.html

```
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.

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.

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`) [2]

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.

```
; Barman, Backup and Recovery Manager for PostgreSQL
; http://www.pgbarman.org/ - http://www.2ndQuadrant.com/
;
; Main configuration file

[barman]
; Main directory
barman_home = /var/lib/barman

; System user
barman_user = barman

; Log location
log_file = /var/log/barman/barman.log

; Default compression level: possible values are None (default), bzip2, gzip or custom
;compression = gzip

; Pre/post backup hook scripts
;pre_backup_script = env | grep ^BARMAN
;post_backup_script = env | grep ^BARMAN

; Pre/post archive hook scripts
;pre_archive_script = env | grep ^BARMAN
;post_archive_script = env | grep ^BARMAN
```

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

```
; Directory of configuration files. Place your sections in separate files with .conf
 extension
; For example place the 'main' server section in /etc/barman.d/main.conf
;configuration_files_directory = /etc/barman.d

; Minimum number of required backups (redundancy) - default 0
;minimum_redundancy = 0

; Global retention policy (REDUNDANCY or RECOVERY WINDOW) - default empty
;retention_policy =

; Global bandwidth limit in KBPS - default 0 (meaning no limit)
;bandwidth_limit = 4000

; Immediate checkpoint for backup command - default false
;immediate_checkpoint = false

; Enable network compression for data transfers - default false
;network_compression = false

; Identify the standard behavior for backup operations: possible values are
; exclusive_backup (default), concurrent_backup
;backup_options = exclusive_backup

; Number of retries of data copy during base backup after an error - default 0
;basebackup_retry_times = 0

; Number of seconds of wait after a failed copy, before retrying - default 30
;basebackup_retry_sleep = 30

; Time frame that must contain the latest backup date.
; If the latest backup is older than the time frame, barman check
; command will report an error to the user.
; If empty, the latest backup is always considered valid.
; Syntax for this option is: "i (DAYS | WEEKS | MONTHS)" where i is an
; integer > 0 which identifies the number of days | weeks | months of
; validity of the latest backup for this check. Also known as 'smelly backup'.
;last_backup_maximum_age =

;; ; 'main' PostgreSQL Server configuration
;; [main]
;; ; Human readable description
;; description =  "Main PostgreSQL Database"
;;
;; ; SSH options
;; ssh_command = ssh postgres@pg
;;
;; ; PostgreSQL connection string
;; conninfo = host=pg user=postgres
;;
;; ; Minimum number of required backups (redundancy)
;; ; minimum_redundancy = 1
;;
;; ; Examples of retention policies
;;
;; ; Retention policy (disabled)
;; ; retention_policy =
;; ; Retention policy (based on redundancy)
;; ; retention_policy = REDUNDANCY 2
;; ; Retention policy (based on recovery window)
;; ; retention_policy = RECOVERY WINDOW OF 4 WEEKS
```

You can now test the 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.

Continuous WAL archiving

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 [3] and the `backup` server (in particular, that the WAL files are collected in the destination directory).

Listing the servers

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

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

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.

[3] For more information, refer to the PostgreSQL documentation

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).

When issuing a backup, Barman normally waits for the checkpoint to happen on the PostgreSQL server (depending on the configuration settings for workload or time checkpoint control). This might take longer for a backup to start.

By setting `immediate_checkpoint` to `true`, you can force the checkpoint on the Postgres server to happen immediately and start your backup copy process 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.

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: tb_name:/home/tblspace/name, tb_temp:/home/tblspace/temp)
```

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 WAL files archived.

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
```

Restoring a whole server

To restore a whole server issue the following command:

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

where `20110920T185953` is the ID of the backup to be restored. When this command completes succesfully, `/path/to/recover/directory` 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/directory 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.

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.

> **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:

- Barman needs at least 4GB of free space in the system 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.

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).

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(name)` function [4]

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.

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`)

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 three values:

- `gzip`: for Gzip compression (requires gzip)
- `bzip2`: for Bzip2 compression (requires bzip2)
- `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

[4] Only available for PostgreSQL 9.1 and above users

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.

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`.

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`.

Minimum redundancy safety

From version 1.2.0, you can define the minimum number of periodical 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.

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 periodical 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 periodical backups) or a **recovery window** (how long).

Retention policy based on redundancy
: In a retention policy, the setting that determines how many periodical backups to keep. A redundancy-based retention policy is contrasted with retention policy that uses 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.

Scope

Retention policies can be defined for:

- **PostgreSQL periodical 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 periodical 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 periodical weekly backups).

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

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.

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.

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. [5] 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. Detailed information as well as the source code of `pgespresso` can be found at https://github.com/2ndquadrant-it/pgespresso.

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`.

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).[6]

[5] 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`).

[6] 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

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.

General commands

List available servers

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

```
barman list-server
```

Maintenance mode

You can perform maintenance operations, like compressing WAL files and moving them from the *incoming directory* to the archived one, with:

```
barman cron
```

This command 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*.

Server commands

Show the configuration for a given server

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

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

Take a base backup

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

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.

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

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

Show available backups for a server

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

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

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*. [7] 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.

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.

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

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

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.

> **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>
```

Backup commands

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

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
```

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
```

> **Warning**
>
> Until retention policies are natively supported, you must use the `oldest` shortcut with extreme care and caution. Iteratively executing this command can easily wipe out your backup archive.

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.

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

> **Important**
>
> No check is performed on the exit code of a script. The result will be simply written in the log file.

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):

- `pre_backup_script`: hook script launched **before** a base backup
- `post_backup_script`: hook script launched **after** a base backup

The script definition is passed to a shell and can return any exit code.

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_SERVER: name of the server
- BARMAN_STATUS: status of the backup
- BARMAN_VERSION: version of Barman (from 1.2.1)

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 launched **before** a WAL file is archived by maintenance (usually barman cron)
- post_archive_script: hook script launched **after** a WAL file is archived by maintenance

The script is executed through a shell and can return any exit code.

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

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 our websites:

- Barman website: http://www.pgbarman.org/
- Support section on the website: http://www.pgbarman.org/support/
- 2ndQuadrant website: http://www.2ndquadrant.com/

Useful information can be found in:

- the FAQ section of the website: http://www.pgbarman.org/faq/
- the "Barman" category of 2ndQuadrant's blog: http://blog.2ndquadrant.com/tag/barman/

Important

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

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`).

Authors

In alphabetical order:

- Gabriele Bartolini <gabriele.bartolini@2ndquadrant.it> (core team, project leader)
- Giuseppe Broccolo <giuseppe.broccolo@2ndquadrant.it> (core team, QA/testing)
- Giulio Calacoci <giulio.calacoci@2ndquadrant.it> (core team, developer)
- Francesco Canovai <francesco.canovai@2ndquadrant.it> (core team, QA/testing)
- Marco Nenciarini <marco.nenciarini@2ndquadrant.it> (core team, team leader)

Past contributors:

- Carlo Ascani

Links

- `check-barman`: a Nagios plugin for Barman, written by Holger Hamann (https://github.com/hamann/check-barman, MIT license)
- `puppet-barman`: Barman module for Puppet (https://github.com/2ndquadrant-it/puppet-barman, GPL)

License and Contributions

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 file `AUTHORS`. 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.