2ndQuadrant®
P o s t g r e S Q L

Barman
Backup and recovery manager for PostgreSQL

Barman Manual

August 1, 2019 (2.9)

2ndQuadrant Limited

Contents

2ndQuadrant®
P o s t g r e S Q L

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

Barman is distributed under GNU GPL 3 and maintained by 2ndQuadrant, a platinum sponsor of the PostgreSQL project.

> **IMPORTANT:**
> This manual assumes that you are familiar with theoretical disaster recovery concepts, and that you have a grasp of PostgreSQL fundamentals in terms of physical backup and disaster recovery. See section *"Before you start"* below for details.

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 fix the incident and recover the database in the shortest time possible. We normally refer to this discipline as **disaster recovery**, and more broadly *business continuity*.

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

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

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

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

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

Based on our experience at 2ndQuadrant, we can confirm that PostgreSQL open source clusters with Barman and repmgr can easily achieve more than 99.99% uptime over a year, if properly configured and monitored.

In any case, it is important for us to emphasise more on cultural aspects related to disaster recovery, rather than the actual tools. Tools without human beings are useless.

Our mission with Barman is to promote a culture of disaster recovery that:

- focuses on backup procedures

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

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

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

Be prepared, don't be scared.

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

Currently, Barman works only on Linux and Unix operating systems.

Before you start

Before you start using Barman, it is fundamental that you get familiar with PostgreSQL and the concepts around physical backups, Point-In-Time-Recovery and replication, such as base backups, WAL archiving, etc.

Below you can find a non exhaustive list of resources that we recommend for you to read:

- *PostgreSQL documentation*:

    - SQL Dump[1]
    - File System Level Backup
    - Continuous Archiving and Point-in-Time Recovery (PITR)
    - Recovery Configuration
    - Reliability and the Write-Ahead Log

- *Book*: PostgreSQL 10 Administration Cookbook

Professional training on these topics 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.

---

[1]It is important that you know the difference between logical and physical backup, therefore between `pg_dump` and a tool like Barman.

Design and architecture

Where to install Barman

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

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

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

- Install Barman on a dedicated server
- Do not share the same storage with your PostgreSQL server
- Integrate Barman with your monitoring infrastructure [2]
- Test everything before you deploy it to production

A reasonable way to start modelling your disaster recovery architecture is to:

- design a couple of possibile architectures in respect to PostgreSQL and Barman, such as:

  1. same data centre
  2. different data centre in the same metropolitan area
  3. different data centre

- elaborate the pros and the cons of each hypothesis
- evaluate the single points of failure (SPOF) of your system, with cost-benefit analysis
- make your decision and implement the initial solution

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

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

---

[2]Integration with Nagios/Icinga is straightforward thanks to the `barman check --nagios` command, one of the most important features of Barman and a true lifesaver.

Backup architecture for PostgreSQL

Example of geographical redundancy
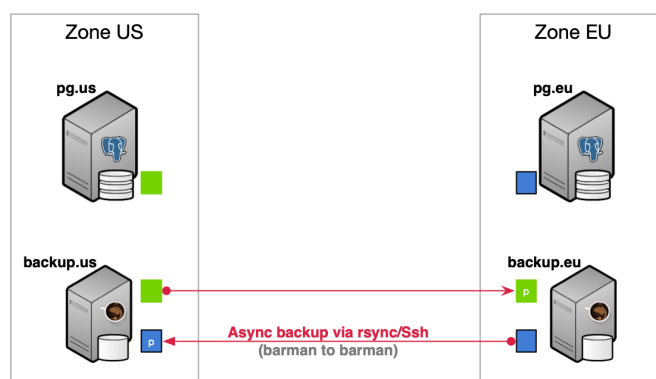
Figure 1: An example of architecture with geo-redundancy

With *geographical redundancy*, you can rely on a Barman instance that is located in a different data centre/availability zone to synchronise the entire content of the source Barman server. There's more: given that geo-redundancy can be configured in Barman not only at global level, but also at server level, you can create *hybrid installations* of Barman where some servers are directly connected to the local PostgreSQL servers, and others are backing up subsets of different Barman installations (*cross-site backup*). Figure 1 below shows two availability zones (one in Europe and one in the US), each with a primary PostgreSQL server that is backed up in a local Barman installation, and relayed on the other Barman server (defined as *passive*) for multi-tier backup via rsync/SSH. Further information on geo-redundancy is available in the specific section.

Thanks to *hook scripts* instead, backups of Barman can be exported on different media, such as *tape* via `tar`, or locations, like an *S3 bucket* in the Amazon cloud.

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

One Barman, many PostgreSQL servers

Another relevant feature that was first introduced by Barman is support for multiple servers. Barman can store backup data coming from multiple PostgreSQL instances, even with different versions, in a centralised way. [3]

---

[3]The same requirements for PostgreSQL's PITR apply for recovery, as detailed in the section *"Requirements for recovery"*.

As a result, you can model complex disaster recovery architectures, forming a "star schema", where PostgreSQL servers rotate around a central Barman server.

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

From this point forward, for the sake of simplicity, this guide will assume a basic architecture:

- one PostgreSQL instance (with host name `pg`)
- one backup server with Barman (with host name `backup`)

Streaming backup vs rsync/SSH

Traditionally, Barman has always operated remotely via SSH, taking advantage of `rsync` for physical backup operations. Version 2.0 introduces native support for PostgreSQL's streaming replication protocol for backup operations, via `pg_basebackup`. [4]

Choosing one of these two methods is a decision you will need to make.

On a general basis, starting from Barman 2.0, backup over streaming replication is the recommended setup for PostgreSQL 9.4 or higher. Moreover, if you do not make use of tablespaces, backup over streaming can be used starting from PostgreSQL 9.2.

> **IMPORTANT:**
> Because Barman transparently makes use of `pg_basebackup`, features such as incremental backup, parallel backup, deduplication, and network compression are currently not available. In this case, bandwidth limitation has some restrictions - compared to the traditional method via `rsync`.

Traditional backup via `rsync`/SSH is available for all versions of PostgreSQL starting from 8.3, and it is recommended in all cases where `pg_basebackup` limitations occur (for example, a very large database that can benefit from incremental backup and deduplication).

The reason why we recommend streaming backup is that, based on our experience, it is easier to setup than the traditional one. Also, streaming backup allows you to backup a PostgreSQL server on Windows[5], and makes life easier when working with Docker.

Standard archiving, WAL streaming . . . or both

PostgreSQL's Point-In-Time-Recovery requires that transactional logs, also known as *xlog* or WAL files, are stored alongside of base backups.

---

[4]Check in the "Feature matrix" which PostgreSQL versions support streaming replication backups with Barman.

[5]Backup of a PostgreSQL server on Windows is possible, but it is still experimental because it is not yet part of our continuous integration system. See section *"How to setup a Windows based server"* for details.

Traditionally, Barman has supported standard WAL file shipping through PostgreSQL's `archive_command` (usually via `rsync`/SSH, now via `barman-wal-archive` from the `barman-cli` package). With this method, WAL files are archived only when PostgreSQL *switches* to a new WAL file. To keep it simple, this normally happens every 16MB worth of data changes.

Barman 1.6.0 introduces streaming of WAL files for PostgreSQL servers 9.2 or higher, as an additional method for transactional log archiving, through `pg_receivewal` (also known as `pg_receivexlog` before PostgreSQL 10). WAL streaming is able to reduce the risk of data loss, bringing RPO down to *near zero* values.

Barman 2.0 introduces support for replication slots with PostgreSQL servers 9.4 or above, therefore allowing WAL streaming-only configurations. Moreover, you can now add Barman as a synchronous WAL receiver in your PostgreSQL 9.5 (or higher) cluster, and achieve **zero data loss** (RPO=0).

In some cases you have no choice and you are forced to use traditional archiving. In others, you can choose whether to use both or just WAL streaming. Unless you have strong reasons not to do it, we recommend to use both channels, for maximum reliability and robustness.

Two typical scenarios for backups

In order to make life easier for you, below we summarise the two most typical scenarios for a given PostgreSQL server in Barman.

Bear in mind that this is a decision that you must make for every single server that you decide to back up with Barman. This means that you can have heterogeneous setups within the same installation.

As mentioned before, we will only worry about the PostgreSQL server (pg) and the Barman server (`backup`). However, in real life, your architecture will most likely contain other technologies such as repmgr, pgBouncer, Nagios/Icinga, and so on.

Scenario 1: Backup via streaming protocol

If you are using PostgreSQL 9.4 or higher, and your database falls under a general use case scenario, you will likely end up deciding on a streaming backup installation - see figure 2 below.

In this scenario, you will need to configure:

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

This setup, in Barman's terminology, is known as **streaming-only** setup, as it does not require any SSH connection for backup and archiving operations. This is particularly suitable and extremely practical for Docker environments.

However, as mentioned before, you can configure standard archiving as well and implement a more robust architecture - see figure 3 below.
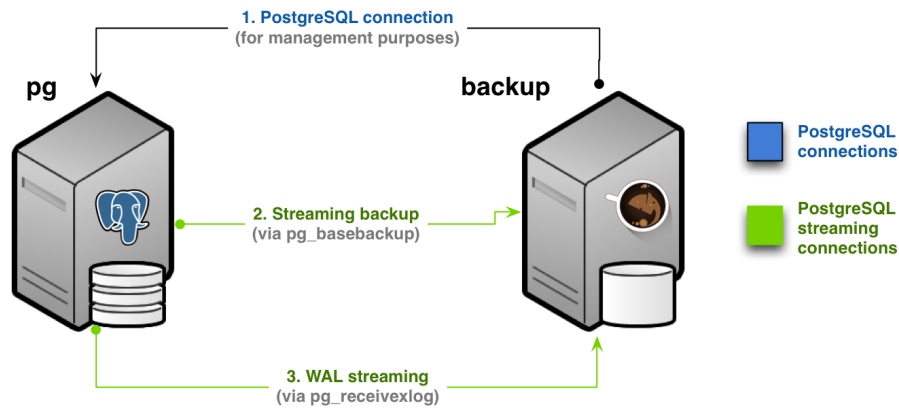
Figure 2: Streaming-only backup (Scenario 1)

Backup architecture for PostgreSQL

Scenario 1b - Streaming backup with fallback WAL archiving
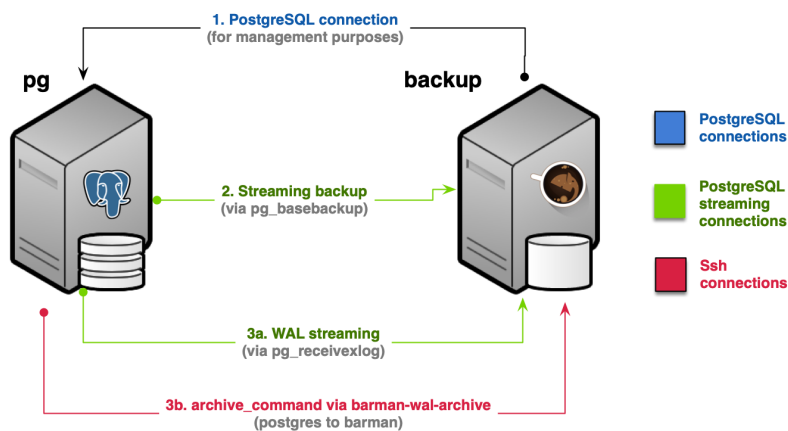
Figure 3: Streaming backup with WAL archiving (Scenario 1b)

Backup architecture for PostgreSQL

Scenario 2 - Traditional Barman setup
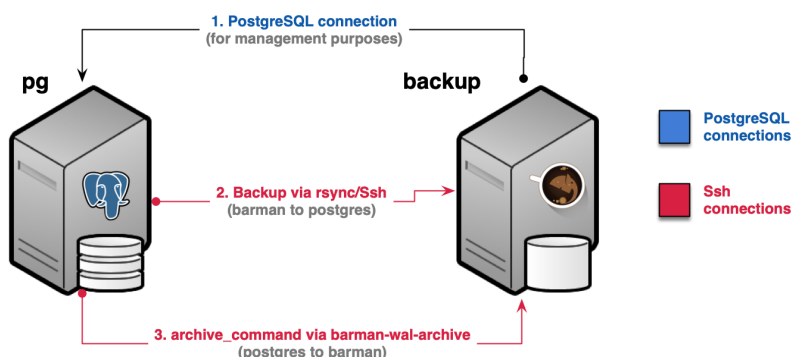
Figure 4: Scenario 2 - Backup via rsync/SSH

This alternate approach requires:

- an additional SSH connection that allows the `postgres` user on the PostgreSQL server to connect as `barman` user on the Barman server
- the `archive_command` in PostgreSQL be configured to ship WAL files to Barman

This architecture is available also to PostgreSQL 9.2/9.3 users that do not use tablespaces.

Scenario 2: Backup via `rsync`/SSH

The *traditional* setup of `rsync` over SSH is the only available option for:

- PostgreSQL servers version 8.3, 8.4, 9.0 or 9.1
- PostgreSQL servers version 9.2 or 9.3 that are using tablespaces
- incremental backup, parallel backup and deduplication
- network compression during backups
- finer control of bandwidth usage, including on a tablespace basis

In this scenario, you will need to configure:

1. a standard connection to PostgreSQL for management, coordination, and monitoring purposes

Backup architecture for PostgreSQL

Scenario 2b - Traditional Barman setup with WAL streaming
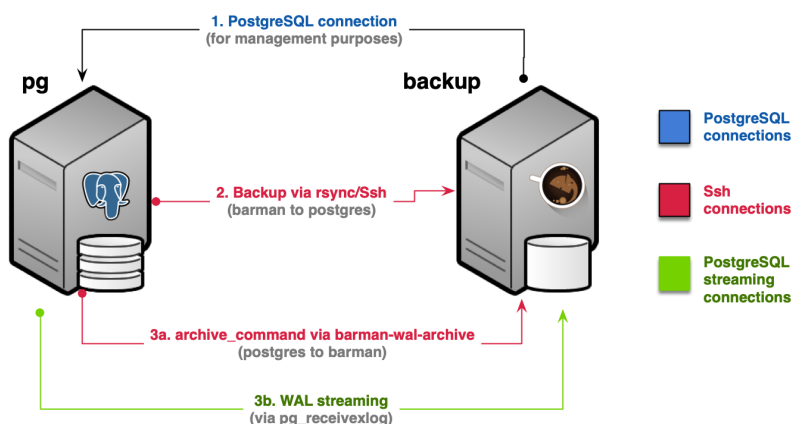
Figure 5: Backup via rsync/SSH with WAL streaming (Scenario 2b)

2. an SSH connection for base backup operations to be used by `rsync` that allows the `barman` user on the Barman server to connect as `postgres` user on the PostgreSQL server
3. an SSH connection for WAL archiving to be used by the `archive_command` in PostgreSQL and that allows the `postgres` user on the PostgreSQL server to connect as `barman` user on the Barman server

Starting from PostgreSQL 9.2, you can add a streaming replication connection that is used for WAL streaming and significantly reduce RPO. This more robust implementation is depicted in figure 5.

System requirements

- Linux/Unix
- Python >= 3.4
- Python modules:

    - argcomplete
    - argh >= 0.21.2
    - psycopg2 >= 2.4.2
    - python-dateutil
    - setuptools

- PostgreSQL >= 8.3
- rsync >= 3.0.4 (optional for PostgreSQL >= 9.2)

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

NOTE: Support for Python 2.6 and 2.7 is deprecated and will be discontinued in future releases. Support for PostgreSQL < 9.4 is deprecated and will be discontinued in future releases.

Requirements for backup

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

Although the only file systems that we officially support are XFS and Ext4, we are aware of users that deploy Barman on different file systems including ZFS and NFS.

Requirements for recovery

Barman allows you to recover a PostgreSQL instance either locally (where Barman resides) or remotely (on a separate server).

Remote recovery is definitely the most common way to restore a PostgreSQL server with Barman.

Either way, the same requirements for PostgreSQL's Log shipping and Point-In-Time-Recovery apply:

- identical hardware architecture
- identical major version of PostgreSQL

In general, it is **highly recommended** to create recovery environments that are as similar as possible, if not identical, to the original server, because they are easier to maintain. For example, we suggest that you use the same operating system, the same PostgreSQL version, the same disk layouts, and so on.

Additionally, dedicated recovery environments for each PostgreSQL server, even on demand, allows you to nurture the disaster recovery culture in your team. You can be prepared for when something unexpected happens by practising recovery operations and becoming familiar with them.

Based on our experience, designated recovery environments reduce the impact of stress in real failure situations, and therefore increase the effectiveness of recovery operations.

Finally, it is important that time is synchronised between the servers, using NTP for example.

Installation

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

Installation on RedHat/CentOS using RPM packages

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

Official RPM packages for Barman are distributed by 2ndQuadrant via Yum through 2ndQuadrant Public RPM repository, by following the instructions you find on that website.

Then, as `root` simply type:

```
yum install barman
```

> **NOTE:** We suggest that you exclude any Barman related packages from getting updated via the PGDG repository. This can be done by adding the following line to any PGDG repository definition that is included in the Barman server inside any `/etc/yum.repos.d/pgdg-*.repo` file:
>
> `exclude=barman*`
>
> By doing this, you solely rely on 2ndQuadrant repositories for package management of Barman software.

For historical reasons, 2ndQuadrant keeps maintaining package distribution of Barman through Sourceforge.net.

Installation 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 and Ubuntu, however, these repositories might not contain the latest available version. If you want to have the latest version of Barman, the recommended method is to install both these repositories:

- 2ndQuadrant Public APT repository, directly maintained by Barman developers
- the PostgreSQL Community APT repository, by following instructions 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 always have access to the most updated versions of Barman.

Installing Barman is as easy. As `root` user simply type:

```
apt-get install barman
```

Installation 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, and so on.

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
```

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

Barman is also available on the Python Package Index (PyPI) and can be installed through `pip`.

Upgrading from Barman 1.X

Version 2.0 requires that users explicitly configure their archiving strategy. Before, the file based archiver, controlled by `archiver`, was enabled by default.

When you upgrade your Barman installation to 2.0, make sure you add the following line either globally or for any server that requires it:

```
archiver = on
```

Additionally, for a few releases, Barman will transparently set `archiver = on` with any server that has not explicitly set an archiving strategy and emit a warning.

Besides that, version 2.0 is fully compatible with older ones.

Configuration

There are two types of configuration files in Barman:

- **global/general configuration**
- **server configuration**

The main configuration file (set to `/etc/barman.conf` by default) contains general options such as main directory, system user, log file, and so on.

Server configuration files, one for each server to be backed up by Barman, are located in the `/etc/barman.d` directory and must have a `.conf` suffix.

> **IMPORTANT**: For historical reasons, you can still have one single configuration file containing both global and server options. However, for maintenance reasons, this approach is deprecated.

Configuration files in Barman follow the *INI* format.

Configuration files accept distinct types of parameters:

- string
- enum
- integer
- boolean, `on`/`true`/`1` are accepted as well are `off`/`false`/`0`.

None of them requires to be quoted.

> *NOTE*: some `enum` allows `off` but not `false`.

Options scope

Every configuration option has a *scope*:

- global
- server
- global/server: server options that can be generally set at global level

Global options are allowed in the *general section*, which is identified in the INI file by the `[barman]` label:

```
[barman]
; ... global and global/server options go here
```

Server options can only be specified in a *server section*, which is identified by a line in the configuration file, in square brackets (`[` and `]`). The server section represents the ID of that server in Barman. The following example specifies a section for the server named `pg`:

```
[pg]
; Configuration options for the
; server named 'pg' go here
```

There are two reserved words that cannot be used as server names in Barman:

- `barman`: identifier of the global section
- `all`: a handy shortcut that allows you to execute some commands on every server managed by Barman in sequence

Barman implements the **convention over configuration** design paradigm, which attempts to reduce the number of options that you are required to configure without losing flexibility. Therefore, some server options can be defined at global level and overridden at server level, allowing users to specify a generic behavior and refine it for one or more servers. These options have a global/server scope.

For a list of all the available configurations and their scope, please refer to section 5 of the 'man' page.

```
man 5 barman
```

Examples of configuration

The following is a basic example of main configuration file:

```
[barman]
barman_user = barman
configuration_files_directory = /etc/barman.d
barman_home = /var/lib/barman
log_file = /var/log/barman/barman.log
log_level = INFO
compression = gzip
```

The example below, on the other hand, is a server configuration file that uses streaming backup:

```
[streaming-pg]
description =  "Example of PostgreSQL Database (Streaming-Only)"
conninfo = host=pg user=barman dbname=postgres
streaming_conninfo = host=pg user=streaming_barman
backup_method = postgres
streaming_archiver = on
slot_name = barman
```

The following code shows a basic example of traditional backup using `rsync`/SSH:

```
[ssh-pg]
description =  "Example of PostgreSQL Database (via Ssh)"
ssh_command = ssh postgres@pg
conninfo = host=pg user=barman dbname=postgres
backup_method = rsync
parallel_jobs = 1
reuse_backup = link
archiver = on
```

For more detailed information, please refer to the distributed `barman.conf` file, as well as the `ssh-server.conf-template` and `streaming-server.conf-template` template files.

Setup of a new server in Barman

As mentioned in the *"Design and architecture"* section, we will use the following conventions:

- `pg` as server ID and host name where PostgreSQL is installed
- `backup` as host name where Barman is located
- `barman` as the user running Barman on the `backup` server (identified by the parameter `barman_user` in the configuration)
- `postgres` as the user running PostgreSQL on the `pg` server

Preliminary steps

This section contains some preliminary steps that you need to undertake before setting up your PostgreSQL server in Barman.

> **IMPORTANT:** Before you proceed, it is important that you have made your decision in terms of WAL archiving and backup strategies, as outlined in the *"Design and architecture"* section. In particular, you should decide which WAL archiving methods to use, as well as the backup method.

PostgreSQL connection

You need to make sure that the `backup` server can connect to the PostgreSQL server on `pg` as superuser. This operation is mandatory.

We recommend creating a specific user in PostgreSQL, named `barman`, as follows:

```
postgres@pg$ createuser -s -P barman
```

> **IMPORTANT:** The above command will prompt for a password, which you are then advised to add to the `~barman/.pgpass` file on the `backup` server. For further information, please refer to "The Password File" section in the PostgreSQL Documentation.

This connection is required by Barman in order to coordinate its activities with the server, as well as for monitoring purposes.

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

Make sure you test the following command before proceeding:

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

Write down the above information (user name, host name and database name) and keep it for later. You will need it with in the `conninfo` option for your server configuration, like in this example:

```
[pg]
; ...
conninfo = host=pg user=barman dbname=postgres
```

> **NOTE:** Barman honours the `application_name` connection option for PostgreSQL servers 9.0 or higher.

PostgreSQL WAL archiving and replication

Before you proceed, you need to properly configure PostgreSQL on `pg` to accept streaming replication connections from the Barman server. Please read the following sections in the PostgreSQL documentation:

- Role attributes
- The pg_hba.conf file
- Setting up standby servers using streaming replication

One configuration parameter that is crucially important is the `wal_level` parameter. This parameter must be configured to ensure that all the useful information necessary for a backup to be coherent are included in the transaction log file.

```
wal_level = 'replica'
```

For PostgreSQL versions older than 9.6, `wal_level` must be set to `hot_standby`.

Restart the PostgreSQL server for the configuration to be refreshed.

PostgreSQL streaming connection

If you plan to use WAL streaming or streaming backup, you need to setup a streaming connection. We recommend creating a specific user in PostgreSQL, named `streaming_barman`, as follows:

```
postgres@pg$ createuser -P --replication streaming_barman
```

> **IMPORTANT:** The above command will prompt for a password, which you are then advised to add to the `~barman/.pgpass` file on the `backup` server. For further information, please refer to "The Password File" section in the PostgreSQL Documentation.

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

```
barman@backup$ psql -U streaming_barman -h pg \
  -c "IDENTIFY_SYSTEM" \
  replication=1
```

> **IMPORTANT:** Please make sure you are able to connect via streaming replication before going any further.

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

```
max_wal_senders = 2
```

This option represents the maximum number of concurrent streaming connections that the server will be allowed to manage.

Another important parameter is `max_replication_slots`, which represents the maximum number of replication slots [6] that the server will be allowed to manage. This parameter is needed if you are planning to use the streaming connection to receive WAL files over the streaming connection:

```
max_replication_slots = 2
```

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

SSH connections

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

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

> **NOTE:** This procedure is not needed if you plan to use the streaming connection only to archive transaction logs and backup your PostgreSQL server.

---

[6]Replication slots have been introduced in PostgreSQL 9.4. See section *"WAL Streaming / Replication slots"* for details.

SSH configuration of postgres user

Unless you have done it before, you need to create an SSH key for the PostgreSQL user. Log in as `postgres`, in the `pg` host and type:

```
postgres@pg$ ssh-keygen -t rsa
```

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

SSH configuration of barman user

As in the previous paragraph, you need to create an SSH key for the Barman user. Log in as `barman` in the `backup` host and type:

```
barman@backup$ ssh-keygen -t rsa
```

For the same reason, no passphrase should be entered.

From PostgreSQL to Barman

The SSH connection from the PostgreSQL server to the backup server is needed to correctly archive WAL files using the `archive_command` setting.

To successfully connect from the PostgreSQL server to the backup server, the PostgreSQL public key has to be configured into the authorized keys of the backup server for the `barman` user.

The public key to be authorized is stored inside the `postgres` user home directory in a file named `.ssh/id_rsa.pub`, and its content should be included in a file named `.ssh/authorized_keys` inside the home directory of the `barman` user in the backup server. If the `authorized_keys` file doesn't exist, create it using `600` as permissions.

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

```
postgres@pg$ ssh barman@backup -C true
```

The value of the `archive_command` configuration parameter will be discussed in the *"WAL archiving via archive_command section"*.

From Barman to PostgreSQL

The SSH connection between the backup server and the PostgreSQL server is used for the traditional backup over rsync. Just as with the connection from the PostgreSQL server to the backup server, we should authorize the public key of the backup server in the PostgreSQL server for the `postgres` user.

The content of the file `.ssh/id_rsa.pub` in the `barman` server should be put in the file named `.ssh/authorized_keys` in the PostgreSQL server. The permissions of that file should be `600`.

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

```
barman@backup$ ssh postgres@pg -C true
```

The server configuration file

Create a new file, called `pg.conf`, in `/etc/barman.d` directory, with the following content:

```
[pg]
description =  "Our main PostgreSQL server"
conninfo = host=pg user=barman dbname=postgres
backup_method = postgres
# backup_method = rsync
```

The `conninfo` option is set accordingly to the section *"Preliminary steps: PostgreSQL connection"*.

The meaning of the `backup_method` option will be covered in the backup section of this guide.

If you plan to use the streaming connection for WAL archiving or to create a backup of your server, you also need a `streaming_conninfo` parameter in your server configuration file:

```
streaming_conninfo = host=pg user=streaming_barman dbname=postgres
```

This value must be choosen accordingly as described in the section *"Preliminary steps: PostgreSQL connection"*.

WAL streaming

Barman can reduce the Recovery Point Objective (RPO) by allowing users to add continuous WAL streaming from a PostgreSQL server, on top of the standard `archive_command` strategy.

Barman relies on `pg_receivewal`, a utility that has been available from PostgreSQL 9.2 which exploits the native streaming replication protocol and continuously receives transaction logs from a PostgreSQL server (master or standby). Prior to PostgreSQL 10, `pg_receivewal` was named `pg_receivexlog`.

IMPORTANT: Barman requires that `pg_receivewal` is installed on the same server. For PostgreSQL 9.2 servers, you need `pg_receivexlog` of version 9.2 installed alongside Barman. For PostgreSQL 9.3 and above, it is recommended to install the latest available version of `pg_receivewal`, as it is back compatible. Otherwise, users can install multiple versions of `pg_receivewal`/`pg_receivexlog` on the Barman server and properly point to the specific version for a server, using the `path_prefix` 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.

Unless otherwise specified in the `streaming_archiver_name` parameter, and only for PostgreSQL 9.3 or above, Barman will set `application_name` of the WAL streamer process to `barman_receive_wal`, allowing you to monitor its status in the `pg_stat_replication` system view of the PostgreSQL server.

Replication slots

IMPORTANT: replication slots are available since PostgreSQL 9.4

Replication slots are an automated way to ensure that the PostgreSQL server will not remove WAL files until they were received by all archivers. Barman uses this mechanism to receive the transaction logs from PostgreSQL.

You can find more information about replication slots in the PostgreSQL manual.

You can even base your backup architecture on streaming connection only. This scenario is useful to configure Docker-based PostgreSQL servers and even to work with PostgreSQL servers running on Windows.

IMPORTANT: In this moment, the Windows support is still experimental, as it is not yet part of our continuous integration system.

How to configure the WAL streaming

First, the PostgreSQL server must be configured to stream the transaction log files to the Barman server.

To configure the streaming connection from Barman to the PostgreSQL server you need to enable the `streaming_archiver`, as already said, including this line in the server configuration file:

```
streaming_archiver = on
```

If you plan to use replication slots (recommended), another essential option for the setup of the streaming-based transaction log archiving is the `slot_name` option:

```
slot_name = barman
```

This option defines the name of the replication slot that will be used by Barman. It is mandatory if you want to use replication slots.

When you configure the replication slot name, you can create a replication slot for Barman with this command:

```
barman@backup$ barman receive-wal --create-slot pg
Creating physical replication slot 'barman' on server 'pg'
Replication slot 'barman' created
```

Limitations of partial WAL files with recovery

The standard behaviour of `pg_receivewal` 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_receivewal` 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, or exactly 0 in case of synchronous replication.

WAL archiving via `archive_command`

The `archive_command` is the traditional method to archive WAL files.

The value of this PostgreSQL configuration parameter must be a shell command to be executed by the PostgreSQL server to copy the WAL files to the Barman incoming directory.

This can be done in two ways, both requiring a SSH connection:

- via `barman-wal-archive` utility (from Barman 2.6)
- via rsync/SSH (common approach before Barman 2.6)

See sections below for more details.

> **IMPORTANT:** PostgreSQL 9.5 introduced support for WAL file archiving using `archive_command` from a standby. Read the "Concurrent Backup and backup from a standby" section for more detailed information on how Barman supports this feature.

WAL archiving via `barman-wal-archive`

From Barman 2.6, the **recommended way** to safely and reliably archive WAL files to Barman via `archive_command` is to use the `barman-wal-archive` command contained in the `barman-cli` package, distributed via 2ndQuadrant public repositories and available under GNU GPL 3 licence. `barman-cli` must be installed on each PostgreSQL server that is part of the Barman cluster.

Using `barman-wal-archive` instead of rsync/SSH reduces the risk of data corruption of the shipped WAL file on the Barman server. When using rsync/SSH as `archive_command` a WAL file, there is no mechanism that guarantees that the content of the file is flushed and fsync-ed to disk on destination.

For this reason, we have developed the `barman-wal-archive` utility that natively communicates with Barman's `put-wal` command (introduced in 2.6), which is responsible to receive the file, fsync its content and place it in the proper `incoming` directory for that server. Therefore, `barman-wal-archive` reduces the risk of copying a WAL file in the wrong location/directory in Barman, as the only parameter to be used in the `archive_command` is the server's ID.

For more information on the `barman-wal-archive` command, type `man barman-wal-archive` on the PostgreSQL server.

You can check that `barman-wal-archive` can connect to the Barman server, and that the required PostgreSQL server is configured in Barman to accept incoming WAL files with the following command:

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

Where `backup` is the host where Barman is installed, `pg` is the name of the PostgreSQL server as configured in Barman and DUMMY is a placeholder (`barman-wal-archive` requires an argument for the WAL file name, which is ignored).

Edit the `postgresql.conf` file of the PostgreSQL instance on the `pg` database, activate the archive mode and set `archive_command` to use `barman-wal-archive`:

```
archive_mode = on
wal_level = 'replica'
archive_command = 'barman-wal-archive backup pg %p'
```

Then restart the PostgreSQL server.

WAL archiving via rsync/SSH

You can retrieve the incoming WALs directory using the `show-server` Barman command and looking for the `incoming_wals_directory` value:

```
barman@backup$ barman show-server pg |grep incoming_wals_directory
        incoming_wals_directory: /var/lib/barman/pg/incoming
```

Edit the `postgresql.conf` file of the PostgreSQL instance on the `pg` database and activate the archive mode:

```
archive_mode = on
wal_level = 'replica'
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 pg` command above.

Restart the PostgreSQL server.

In some cases, you might want to add stricter checks to the `archive_command` process. For example, some users have suggested the following one:

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

Where the `HOSTNAME` placeholder should be replaced with the value returned by `hostname --fqdn`. This *trick* is a safeguard in case the server is cloned and avoids receiving WAL files from recovered PostgreSQL instances.

Verification of WAL archiving configuration

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, you need to check that WAL files are correctly collected in the destination directory.

For this purpose and to facilitate the verification of the WAL archiving process, the `switch-wal` command has been developed:

```
barman@backup$ barman switch-wal --force --archive pg
```

The above command will force PostgreSQL to switch WAL file and trigger the archiving process in Barman. Barman will wait for one file to arrive within 30 seconds (you can change the timeout through the `--archive-timeout` option). If no WAL file is received, an error is returned.

You can verify if the WAL archiving has been correctly configured using the `barman check` command.

Streaming backup

Barman can backup a PostgreSQL server using the streaming connection, relying on `pg_basebackup`, a utility that has been available from PostgreSQL 9.1.

> **IMPORTANT:** Barman requires that `pg_basebackup` is installed in the same server. For PostgreSQL 9.2 servers, you need the `pg_basebackup` of version 9.2 installed alongside with Barman. For PostgreSQL 9.3 and above, it is recommented to install the last available version of `pg_basebackup`, as it is back compatible. You can even install multiple versions of `pg_basebackup` on the Barman server and properly point to the specific version for a server, using the `path_prefix` option in the configuration file.

To successfully backup your server with the streaming connection, you need to use `postgres` as your backup method:

```
backup_method = postgres
```

> **IMPORTANT:** keep in mind that if the WAL archiving is not currently configured, you will not be able to start a backup.

To check if the server configuration is valid you can use the `barman check` command:

```
barman@backup$ barman check pg
```

To start a backup you can use the `barman backup` command:

```
barman@backup$ barman backup pg
```

> **IMPORTANT:** `pg_basebackup` 9.4 or higher is required for tablespace support if you use the `postgres` backup method.

Backup with `rsync`/SSH

The backup over `rsync` was the only available method before 2.0, and is currently the only backup method that supports the incremental backup feature. Please consult the *"Features in detail"* section for more information.

To take a backup using `rsync` you need to put these parameters inside the Barman server configuration file:

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

The `backup_method` option activates the `rsync` backup method, and the `ssh_command` option is needed to correctly create an SSH connection from the Barman server to the PostgreSQL server.

> **IMPORTANT:** Keep in mind that if the WAL archiving is not currently configured, you will not be able to start a backup.

To check if the server configuration is valid you can use the `barman check` command:

```
barman@backup$ barman check pg
```

To take a backup use the `barman backup` command:

```
barman@backup$ barman backup pg
```

How to setup a Windows based server

You can backup a PostgreSQL server running on Windows using the streaming connection for both WAL archiving and for backups.

> **IMPORTANT:** This feature is still experimental because it is not yet part of our continuous integration system.

Follow every step discussed previously for a streaming connection setup.

> **WARNING::** At this moment, `pg_basebackup` interoperability from Windows to Linux is still experimental. If you are having issues taking a backup from a Windows server and your PostgreSQL locale is not in English, a possible workaround for the issue is instructing your PostgreSQL to emit messages in English. You can do this by putting the following parameter in your `postgresql.conf` file:

```
lc_messages = 'English'
```

This has been reported to fix the issue.

You can backup your server as usual.

Remote recovery is not supported for Windows servers, so you must recover your cluster locally in the Barman server and then copy all the files on a Windows server or use a folder shared between the PostgreSQL server and the Barman server.

Additionally, make sure that the system user chosen to run PostgreSQL has the permission needed to access the restored data. Basically, it must have full control over the PostgreSQL data directory.

General commands

Barman has many commands and, for the sake of exposition, we can organize them by scope.

The scope of the **general commands** is the entire Barman server, that can backup many PostgreSQL servers. **Server commands**, instead, act only on a specified server. **Backup commands** work on a backup, which is taken from a certain server.

The following list includes the general commands.

`cron`

barman doesn't include a long-running daemon or service file (there's nothing to `systemctl start`, `service start`, etc.). Instead, the `barman cron` subcommand is provided to perform `barman`'s background "steady-state" backup operations.

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

```
barman cron
```

> **NOTE:** This command should be executed in a *cron script*. Our recommendation is to schedule `barman cron` to run every minute. If you installed Barman using the rpm or debian package, a cron entry running on every minute will be created for you.

`barman cron` executes WAL archiving operations concurrently on a server basis, and this also enforces retention policies on those servers that have:

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

The `cron` command ensures that WAL streaming is started for those servers that have requested it, by transparently executing the `receive-wal` command.

In order to stop the operations started by the `cron` command, comment out the cron entry and execute:

```
barman receive-wal --stop SERVER_NAME
```

You might want to check `barman list-server` to make sure you get all of your servers.

diagnose

The `diagnose` command creates a JSON report useful for diagnostic and support purposes. This report contains information for all configured servers.

> **IMPORTANT:** Even if the diagnose is written in JSON and that format is thought to be machine readable, its structure is not to be considered part of the interface. Format can change between different Barman versions.

list-server

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

```
barman list-server
```

A machine readble output can be obtained with the `--minimal` option:

```
barman list-server --minimal
```

Server commands

As we said in the previous section, server commands work directly on a PostgreSQL server or on its area in Barman, and are useful to check its status, perform maintainance operations, take backups, and manage the WAL archive.

archive-wal

The `archive-wal` command execute maintainance operations on WAL files for a given server. This operations include processing of the WAL files received from the streaming connection or from the `archive_command` or both.

> **IMPORTANT:** The `archive-wal` command, even if it can be directly invoked, is designed to be started from the `cron` general command.

backup

The `backup` command takes a full backup (*base backup*) of a given server. It has several options that let you override the corresponding configuration parameter for the new backup. For more information, consult the manual page.

You can perform a full backup for a given server with:

`barman backup <server_name>`

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

check

You can check the connection to a given server and the configuration coherence with the `check` command:

`barman check <server_name>`

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

> **IMPORTANT:** The `check` command is probably the most critical feature that Barman implements. We recommend to integrate it with your alerting and monitoring infrastructure. The `--nagios` option allows you to easily create a plugin for Nagios/Icinga.

get-wal

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
- `-p SIZE` peeks from the archive up to WAL files, starting from the requested file

It is possible to use `get-wal` during a recovery operation, transforming the Barman server into 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 configuration by setting the `restore_command` so that `barman get-wal` is used to fetch the required WAL files. Similarly, one can use the `--get-wal` option for the `recover` command at run-time.

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

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

Please note that the `get-wal` command should always be invoked as `barman` user, and that it requires the correct permission to read the WAL files from the catalog. This is the reason why we are using `sudo -u barman` in the example.

Setting `recovery_options` to `get-wal` for a remote recovery will instead generate a `restore_command` using the `barman-wal-restore` script. `barman-wal-restore` is a more resilient shell script which manages SSH connection errors.

This script has many useful options such as the automatic compression and decompression of the WAL files and the *peek* feature, which allows you to retrieve the next WAL files while PostgreSQL is applying one of them. It is an excellent way to optimise the bandwidth usage between PostgreSQL and Barman.

`barman-wal-restore` is available in the `barman-cli` package.

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

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

Since it uses SSH to communicate with the Barman server, SSH key authentication is required for the `postgres` user to login as `barman` on the backup server.

You can check that `barman-wal-restore` can connect to the Barman server, and that the required PostgreSQL server is configured in Barman to send WAL files with the following command:

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

Where `backup` is the host where Barman is installed, `pg` is the name of the PostgreSQL server as configured in Barman and DUMMY is a placeholder (`barman-wal-restore` requires two argument for the WAL file name and destination directory, which are ignored). For more information on the `barman-wal-restore` command, type `man barman-wal-restore` on the PostgreSQL server.

list-backup

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

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

> **TIP:** You can request a full list of the backups of all servers using `all` as the server name.

To have a machine-readable output you can use the `--minimal` option.

rebuild-xlogdb

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 server managed by Barman has its own copy.

The `xlog.db` file can be rebuilt with the `rebuild-xlogdb` command. This will scan all the archived WAL files and regenerate the metadata for the archive.

For example:

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

receive-wal

This command manages the `receive-wal` process, which uses the streaming protocol to receive WAL files from the PostgreSQL streaming connection.

receive-wal process management

If the command is run without options, a `receive-wal` process will be started. This command is based on the `pg_receivewal` PostgreSQL command.

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

If the command is run with the `--stop` option, the currently running `receive-wal` process will be stopped.

The `receive-wal` process uses a status file to track last written record of the transaction log. When the status file needs to be cleaned, the `--reset` option can be used.

> **IMPORTANT:** If you are not using replication slots, you rely on the value of `wal_keep_segments`. Be aware that under high peeks of workload on the database, the `receive-wal` process might fall behind and go out of sync. As a precautionary measure, Barman currently requires that users manually execute the command with the `--reset` option, to avoid making wrong assumptions.

Replication slot management

The `receive-wal` process is also useful to create or drop the replication slot needed by Barman for its WAL archiving procedure.

With the `--create-slot` option, the replication slot named after the `slot_name` configuration option will be created on the PostgreSQL server.

With the `--drop-slot`, the previous replication slot will be deleted.

replication-status

The `replication-status` command reports the status of any streaming client currently attached to the PostgreSQL server, including the `receive-wal` process of your Barman server (if configured).

You can execute the command as follows:

```
barman replication-status <server_name>
```

> **TIP:** You can request a full status report of the replica for all your servers using `all` as the server name.

To have a machine-readable output you can use the `--minimal` option.

show-server

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

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

> **TIP:** you can request a full configuration report using `all` as the server name.

status

The `status` command shows live information and status of a PostgreSQL server or of all servers if you use `all` as server name.

```
barman status <server_name>
```

switch-wal

This command makes the PostgreSQL server switch to another transaction log file (WAL), allowing the current log file to be closed, received and then archived.

```
barman switch-wal <server_name>
```

If there has been no transaction activity since the last transaction log file switch, the switch needs to be forced using the `--force` option.

The `--archive` option requests Barman to trigger WAL archiving after the xlog switch. By default, a 30 seconds timeout is enforced (this can be changed with `--archive-timeout`). If no WAL file is received, an error is returned.

> **NOTE:** In Barman 2.1 and 2.2 this command was called `switch-xlog`. It has been renamed for naming consistency with PostgreSQL 10 and higher.

Backup commands

Backup commands are those that works directly on backups already existing in Barman's backup catalog.

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

Backup ID shortcuts

Barman allows you to use special keywords to identify a specific backup:

- `last/latest`: identifies the newest backup in the catalog
- `first/oldest`: identifies the oldest backup in the catalog

Using those keywords with Barman commands allows you to execute actions without knowing the exact ID of a backup for a server. For example we can issue:

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

to remove the oldest backup available in the catalog and reclaim disk space.

check-backup

Starting with version 2.5, you can check that all required WAL files for the consistency of a full backup have been correctly archived by `barman` with the `check-backup` command:

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

> **IMPORTANT:** This command is automatically invoked by `cron` and at the end of a `backup` operation. This means that, under normal circumstances, you should never need to execute it.

In case one or more WAL files from the start to the end of the backup have not been archived yet, `barman` will label the backup as `WAITING_FOR_WALS`. The `cron` command will continue to check that missing WAL files are archived, then label the backup as `DONE`.

In case the first required WAL file is missing at the end of the backup, such backup will be marked as `FAILED`. It is therefore important that you verify that WAL archiving (whether via streaming or `archive_command`) is properly working before executing a backup operation - especially when backing up from a standby server.

delete

You can delete a given backup with:

`barman delete <server_name> <backup_id>`

The `delete` command accepts any shortcut to identify backups.

list-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 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 can therefore be extremely useful to integrate Barman into your archiving procedures.

recover

The `recover` command is used to recover a whole server after a backup is executed using the `backup` command.

This is achieved issuing a command like the following:

`barman@backup$ barman recover <server_name> <backup_id> /path/to/recover/dir`

> **IMPORTANT:** Do not issue a `recover` command using a target data directory where a PostgreSQL instance is running. In that case, remember to stop it before issuing the recovery. This applies also to tablespace directories.

At the end of the execution of the recovery, the selected backup is recovered locally and the destination path contains a data directory ready to be used to start a PostgreSQL instance.

> **IMPORTANT:** Running this command as user `barman`, it will become the database superuser.

The specific ID of a backup can be retrieved using the list-backup command.

> **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.

The recovery command has several options that modify the command behavior.

Remote recovery

Add the `--remote-ssh-command <COMMAND>` option to the invocation of the recovery command. Doing this will allow Barman to execute the copy on a remote server, using the provided command to connect to the remote host.

> **NOTE:** It is advisable to use the `postgres` user to perform the recovery on the remote host.

> **IMPORTANT:** Do not issue a `recover` command using a target data directory where a PostgreSQL instance is running. In that case, remember to stop it before issuing the recovery. This applies also to tablespace directories.

Known limitations of the remote recovery are:

- Barman requires at least 4GB of free space in the system temporary directory unless the `get-wal` command is specified in the `recovery_option` parameter in the Barman configuration.
- The SSH connection between Barman and the remote host **must** use the public key exchange authentication method
- The remote user **must** be able to create the directory structure of the backup in the destination directory.
- There must be enough free space on the remote server to contain the base backup and the WAL files needed for recovery.

Tablespace remapping

Barman is able to automatically remap 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` is the identifier of the tablespace
- `DIRECTORY` is the new destination path for the tablespace

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

Point in time recovery

Barman wraps PostgreSQL's Point-in-Time Recovery (PITR), allowing you to specify a recovery target, either as a timestamp, as a restore label, or as a transaction ID.

> **IMPORTANT:** The earliest PITR for a given backup is the end of the base backup itself. If you want to recover at any point in time between the start and the end of a backup, you must use the previous backup. From Barman 2.3 you can exit recovery when consistency is reached by using `--target-immediate` option (available only for PostgreSQL 9.4 and newer).

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

- `--target-time TARGET_TIME`: to specify a timestamp
- `--target-xid TARGET_XID`: to specify a transaction ID
- `--target-lsn TARGET_LSN`: to specify a Log Sequence Number (LSN) - requires PostgreSQL 10 or higher
- `--target-name TARGET_NAME`: to specify a named restore point previously created with the pg_create_restore_point(name) function[7]
- `--target-immediate`: recovery ends when a consistent state is reached (that is the end of the base backup process) [8]

> **IMPORTANT:** Recovery target via *time*, *XID* and LSN **must be** subsequent to the end of the backup. If you want to recover to a point in time between the start and the end of a backup, you must recover from the previous backup in the catalogue.

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

---

[7]Only available on PostgreSQL 9.1 and above
[8]Only available on PostgreSQL 9.4 and above

2ndQuadrant®
PostgreSQL

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, as mentioned in the *"Before you start"* section.

Barman 2.4 introduces support for `--target-action` option, accepting the following values:

- `shutdown`: once recovery target is reached, PostgreSQL is shut down [9]
- `pause`: once recovery target is reached, PostgreSQL is started in pause state, allowing users to inspect the instance [10]
- `promote`: once recovery target is reached, PostgreSQL will exit recovery and is promoted as a master [11]

> **IMPORTANT:** By default, no target action is defined (for back compatibility). The `--target-action` option requires a Point In Time Recovery target to be specified.

For more detailed information on the above settings, please consult the PostgreSQL documentation on recovery target settings.

Barman 2.4 also adds the `--standby-mode` option for the `recover` command which, if specified, adds `standby_mode = on` to the generated recovery configuration, causing the recovered PostgreSQL instance to start as a standby. Further information on *standby mode* is available in the PostgreSQL documentation on standby server settings.

show-backup

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

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

The `show-backup` command accepts any shortcut to identify backups.

---

[9]Only available on PostgreSQL 9.5 and above
[10]Only available on PostgreSQL 9.1 and above
[11]Only available on PostgreSQL 9.5 and above

Features in detail

In this section we present several Barman features and discuss their applicability and the configuration required to use them.

This list is not exhaustive, as many scenarios can be created working on the Barman configuration. Nevertheless, it is useful to discuss common patterns.

Backup features

Incremental backup

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

> **NOTE:** Block level incremental backup will be available in future versions.

> **IMPORTANT:** The `reuse_backup` option can't be used with the `postgres` backup method at this time.

The main goals of incremental backups 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 therefore be 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 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>
```

Limiting bandwidth usage

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.

> **IMPORTANT:** the `bandwidth_limit` and the `tablespace_bandwidth_limit` options are not supported with the `postgres` backup method

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

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

> **IMPORTANT:** the `network_compression` option is not available with the `postgres` backup method.

```
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`.

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.

Barman is also capable of performing backups of PostgreSQL from 9.2 or greater database servers in a **concurrent way**, primarily through the `backup_options` configuration parameter.[12]

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

> **IMPORTANT: Concurrent backup** requires users of PostgreSQL 9.2, 9.3, 9.4, and 9.5 to install the `pgespresso` open source extension on every PostgreSQL server of the cluster. For more detailed information and the source code, please visit the pgespresso extension website. Barman supports the new API introduced in PostgreSQL 9.6. This removes the requirement of the `pgespresso` extension to perform concurrent backups from this version of PostgreSQL.

By default, `backup_options` is transparently set to `exclusive_backup` for back compatibility reasons. Users of PostgreSQL 9.6 should set `backup_options` to `concurrent_backup`.

> **IMPORTANT:** When PostgreSQL 9.5 is declared EOL by the Community, Barman will by default set `backup_options` to `concurrent_backup`. Support for `pgespresso` will be ceased then.

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 database. Using PostgreSQL 9.2, 9.3, 9.4, and 9.5, `pgespresso` must be correctly installed through `CREATE EXTENSION`. Using 9.6 or greater, concurrent backups are executed through the Postgres native API (which requires an active connection from the start to the stop of the backup).

> **IMPORTANT:** In case of a concurrent backup, currently Barman cannot determine whether the closing WAL file of a full backup has actually been shipped - opposite of an exclusive backup where PostgreSQL itself 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. Barman 2.5 introduces a new state, called `WAITING_FOR_WALS`, which is managed by the `check-backup` command (part of the ordinary maintenance job performed by the `cron` command).

---

[12]Concurrent backup is a technology that has been available in PostgreSQL since version 9.2, through the *streaming replication protocol* (for example, using a tool like `pg_basebackup`).

Current limitations on backup from standby

Barman currently requires that backup data (base backups and WAL files) come from one server only. Therefore, in case of backup from a standby, you should point to the standby server:

- `conninfo`
- `streaming_conninfo`, if you use `postgres` as `backup_method` and/or rely on WAL streaming
- `ssh_command`, if you use `rsync` as `backup_method`

    IMPORTANT: From Barman 2.8, backup from a standby is supported only for PostgreSQL 9.4 or higher (versions 9.4 and 9.5 require `pgespresso`). Support for 9.2 and 9.3 is deprecated.

The recommended and simplest way is to setup WAL streaming with replication slots directly from the standby, which requires PostgreSQL 9.4. This means:

- configure `streaming_archiver = on`, as described in the "WAL streaming" section, including "Replication slots"
- disable `archiver = on`

Alternatively, from PostgreSQL 9.5 you can decide to archive from the standby only using `archive_command` with `archive_mode = always` and by disabling WAL streaming.

    NOTE: Unfortunately, it is not currently possible to enable both WAL archiving and streaming from the standby due to the way Barman performs WAL duplication checks and an undocumented behaviours in all versions of PostgreSQL.

Archiving features

WAL compression

The `barman cron` command 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: All methods but `pybzip2` and `pygzip` require `barman archive-wal` to fork a new process.

Synchronous WAL streaming

> **IMPORTANT:** This feature is available only from PostgreSQL 9.5 and above.

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

To configure such a scenario, the Barman server must be configured to archive WALs via the streaming connection, and the `receive-wal` process should figure as a synchronous standby of the PostgreSQL server.

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

```
barman@backup$ barman show-server pg|grep streaming_archiver_name
    streaming_archiver_name: barman_receive_wal
```

Then the application name should be added to the `postgresql.conf` file as a synchronous standby:

```
synchronous_standby_names = 'barman_receive_wal'
```

> **IMPORTANT:** this is only an example of configuration, to show you that Barman is eligible to be a synchronous standby node. We are not suggesting to use ONLY Barman. You can read *"Synchronous Replication"* from the PostgreSQL documentation for further information on this topic.

The PostgreSQL server needs to be restarted for the configuration to be reloaded.

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

```
[root@backup ~]# barman replication-status pg
Status of streaming clients for server 'pg':
  Current xlog location on master: 0/9000098
  Number of streaming clients: 1

  1. #1 Sync WAL streamer
     Application name: barman_receive_wal
     Sync stage      : 3/3 Remote write
     Communication   : TCP/IP
     IP Address      : 139.59.135.32 / Port: 58262 / Host: -
     User name       : streaming_barman
     Current state   : streaming (sync)
     Replication slot: barman
```

```
WAL sender PID  : 2501
Started at      : 2016-09-16 10:33:01.725883+00:00
Sent location   : 0/9000098 (diff: 0 B)
Write location  : 0/9000098 (diff: 0 B)
Flush location  : 0/9000098 (diff: 0 B)
```

Catalog management features

Minimum redundancy safety

You can define the minimum number of periodic backups for a PostgreSQL server, using the global/per server configuration option called `minimum_redundancy`, 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 catalog.

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

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

Retention policies

Barman supports **retention policies** for backups.

A backup retention policy is a 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.

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 you 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, for example allowing users 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`.

How they work

Retention policies in Barman can be:

- **automated**: enforced by `barman cron`
- **manual**: Barman simply reports obsolete backups and allows you 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 that value is not assigned, 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 it is not 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.

Hook scripts

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

- before and after a backup
- before and after the deletion of a backup
- before and after a WAL file is archived
- before and after a WAL file is deleted

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

- standard hook scripts
- retry hook scripts

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.

Specifically, 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 logging an additional warning, `ABORT_STOP` will behave like `ABORT_CONTINUE`.

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* 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 hook script 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, `0` if not
- `BARMAN_SERVER`: name of the server
- `BARMAN_STATUS`: status of the backup
- `BARMAN_VERSION`: version of Barman

Backup delete scripts

Version **2.4** introduces pre and post backup delete scripts.

As previous scripts, bakup delete scripts can be configured within global configuration options, and it is possible to override them on a per server basis:

- `pre_delete_script`: *hook script* launched *before* the deletion of a backup, only once, with no check on the exit code
- `pre_delete_retry_script`: *retry hook script* executed *before* the deletion of a backup, repeatedly until success or abort
- `post_delete_retry_script`: *retry hook script* executed *after* the deletion of a backup, repeatedly until success or abort
- `post_delete_script`: *hook script* launched *after* the deletion of a backup, 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).

Delete scripts uses the same environmental variables of a backup script, plus:

- `BARMAN_NEXT_ID`: ID of the next backup (if present)

WAL archive scripts

Similar 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 it is successful or aborted
- `post_archive_retry_script`: *retry hook script* executed *after* a WAL file is archived by maintenance, repeatedly until it is successful or aborted
- `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

WAL delete scripts

Version **2.4** introduces pre and post WAL delete scripts.

Similarly to the other hook scripts, wal delete scripts can be configured with global configuration options, and is possible to override them on a per server basis:

- `pre_wal_delete_script`: *hook script* executed *before* the deletion of a WAL file
- `pre_wal_delete_retry_script`: *retry hook script* executed *before* the deletion of a WAL file, repeatedly until it is successful or aborted
- `post_wal_delete_retry_script`: *retry hook script* executed *after* the deletion of a WAL file, repeatedly until it is successful or aborted
- `post_wal_delete_script`: *hook script* executed *after* the deletion of a WAL file

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).

WAL delete scripts use the same environmental variables as WAL archive scripts.

Recovery scripts

Version **2.4** introduces pre and post recovery scripts.

As previous scripts, recovery scripts can be configured within global configuration options, and is possible to override them on a per server basis:

- `pre_recovery_script`: *hook script* launched *before* the recovery of a backup, only once, with no check on the exit code
- `pre_recovery_retry_script`: *retry hook script* executed *before* the recovery of a backup, repeatedly until success or abort
- `post_recovery_retry_script`: *retry hook script* executed *after* the recovery of a backup, repeatedly until success or abort
- `post_recovery_script`: *hook script* launched *after* the recovery of a backup, 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).

Recovery scripts uses the same environmental variables of a backup script, plus:

- `BARMAN_DESTINATION_DIRECTORY`: the directory where the new instance is recovered

- `BARMAN_TABLESPACES`: tablespace relocation map (JSON, if present)

- `BARMAN_REMOTE_COMMAND`: secure shell command used by the recovery (if present)

- `BARMAN_RECOVER_OPTIONS`: recovery additional options (JSON, if present)

Customization

Lock file directory

Barman allows you 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, and so on.).

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

> **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`).

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.

Integration with cluster management systems

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. Barman, thanks to the `get-wal` framework, can also be used as a WAL hub. For this purpose, you can use the `barman-wal-restore` script, part of the `barman-cli` package, with all your standby servers.

The `replication-status` command allows you to get information about any streaming client attached to the managed server, in particular hot standby servers and WAL streamers.

Parallel jobs

By default, Barman uses only one worker for file copy during both backup and recover operations. Starting from version 2.2, it is possible to customize the number of workers that will perform file copy. In this case, the files to be copied will be equally distributed among all parallel workers.

It can be configured in global and server scopes, adding these in the corresponding configuration file:

```
parallel_jobs = n
```

where n is the desired number of parallel workers to be used in file copy operations. The default value is 1.

In any case, users can override this value at run-time when executing `backup` or `recover` commands. For example, you can use 4 parallel workers as follows:

```
barman backup --jobs 4 server1
```

Or, alternatively:

```
barman backup --j 4 server1
```

Please note that this parallel jobs feature is only available for servers configured through `rsync`/SSH. For servers configured through streaming protocol, Barman will rely on `pg_basebackup` which is currently limited to only one worker.

Geographical redundancy

It is possible to set up **cascading backup architectures** with Barman, where the source of a backup server is a Barman installation rather than a PostgreSQL server.

This feature allows users to transparently keep *geographically distributed* copies of PostgreSQL backups.

In Barman jargon, a backup server that is connected to a Barman installation rather than a PostgreSQL server is defined **passive node**. A passive node is configured through the `primary_ssh_command` option, available both at global (for a full replica of a primary Barman installation) and server level (for mixed scenarios, having both *direct* and *passive* servers).

Sync information

The `barman sync-info` command is used to collect information regarding the current status of a Barman server that is useful for synchronisation purposes. The available syntax is the following:

```
barman sync-info [--primary] <server_name> [<last_wal> [<last_position>]]
```

The command returns a JSON object containing:

- A map with all the backups having status `DONE` for that server
- A list with all the archived WAL files
- The configuration for the server
- The last read position (in the *xlog database file*)
- the name of the last read WAL file

The JSON response contains all the required information for the synchronisation between the `master` and a `passive` node.

If `--primary` is specified, the command is executed on the defined primary node, rather than locally.

Configuration

Configuring a server as `passive node` is a quick operation. Simply add to the server configuration the following option:

```
primary_ssh_command = ssh barman@primary_barman
```

This option specifies the SSH connection parameters to the primary server, identifying the source of the backup data for the passive server.

Node synchronisation

When a node is marked as `passive` it is treated in a special way by Barman:

- it is excluded from standard maintenance operations

- direct operations to PostgreSQL are forbidden, including `barman backup`

Synchronisation between a passive server and its primary is automatically managed by `barman cron` which will transparently invoke:

1. `barman sync-info --primary`, in order to collect synchronisation information
2. `barman sync-backup`, in order to create a local copy of every backup that is available on the primary node
3. `barman sync-wals`, in order to copy locally all the WAL files available on the primary node

Manual synchronisation

Although `barman cron` automatically manages passive/primary node synchronisation, it is possible to manually trigger synchronisation of a backup through:

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

Launching `sync-backup` barman will use the primary_ssh_command to connect to the master server, then if the backup is present on the remote machine, will begin to copy all the files using rsync. Only one single synchronisation process per backup is allowed.

WAL files also can be synchronised, through:

`barman sync-wals <server_name>`

Troubleshooting

Diagnose a Barman installation

You can gather important information about the status of all the configured servers using:

```
barman diagnose
```

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

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

Requesting help

Although Barman is extensively documented, there are a lot of scenarios that are not covered.

For any questions about Barman and disaster recovery scenarios using Barman, you can reach the dev team using the community mailing list:

https://groups.google.com/group/pgbarman

or the IRC channel on freenode: irc://irc.freenode.net/barman

In the event you discover a bug, you can open a ticket using Github: https://github.com/2ndquadrant-it/barman/issues

2ndQuadrant provides professional support for Barman, including 24/7 service.

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 Github issue tracker, by attaching the output obtained by the diagnostics command above (`barman diagnose`)

  WARNING: Be careful when submitting the output of the diagnose command as it might disclose information that are potentially dangerous from a security point of view.

The Barman project

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
- Barman FAQs
- 2ndQuadrant blog: Barman

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.

Authors

In alphabetical order:

- Gabriele Bartolini, gabriele.bartolini@2ndquadrant.it (architect)
- Jonathan Battiato, jonathan.battiato@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)
- Britt Cole, britt.cole@2ndquadrant.com (documentation)
- Marco Nenciarini, marco.nenciarini@2ndquadrant.it (project leader)
- Rubens Souza, rubens.souza@2ndquadrant.it (QA/testing)

Past contributors:

- Carlo Ascani
- Stefano Bianucci
- Giuseppe Broccolo

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)
- BarmanAPI: RESTFul API for Barman, written by Mehmet Emin Karakaş (GPL)

License and Contributions

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

Copyright (C) 2011-2017 2ndQuadrant Limited.

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 Limited 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.com for a copy of the relevant Copyright Assignment Form.

Feature matrix

Below you will find a matrix of PostgreSQL versions and Barman features for backup and archiving:

Version	Backup with rsync/SSH	Backup with pg_basebackup	Standard WAL archiving	WAL Streaming	RPO=0
12	Yes	Yes	Yes	Yes	Yes
11	Yes	Yes	Yes	Yes	Yes
10	Yes	Yes	Yes	Yes	Yes
9.6	Yes	Yes	Yes	Yes	Yes
9.5	Yes	Yes	Yes	Yes	Yes (d)
9.4	Yes	Yes	Yes	Yes	Yes (d)
9.3	Yes	Yes (c)	Yes	Yes (b)	No
9.2	Yes	Yes (a)(c)	Yes	Yes (a)(b)	No
9.1	Yes	No	Yes	No	No
9.0	Yes	No	Yes	No	No
8.4	Yes	No	Yes	No	No
8.3	Yes	No	Yes	No	No

NOTE:

a) `pg_basebackup` and `pg_receivexlog` 9.2 required
b) WAL streaming-only not supported (standard archiving required)
c) Backup of tablespaces not supported
d) When using `pg_receivexlog` 9.5, minor version 9.5.5 or higher required [13]

It is required by Barman that `pg_basebackup` and `pg_receivewal`/`pg_receivexlog` of the same version of the PostgreSQL server (or higher) are installed on the same server where Barman resides. The only exception is that PostgreSQL 9.2 users are required to install version 9.2 of `pg_basebackup` and `pg_receivexlog` alongside with Barman.

> **TIP:** We recommend that the last major, stable version of the PostgreSQL clients (e.g. 11) is installed on the Barman server if you plan to use backup and WAL archiving over streaming replication through `pg_basebackup` and `pg_receivewal`, for PostgreSQL 9.3 or higher servers.

---

[13]The commit "Fix pg_receivexlog –synchronous" is required (included in version 9.5.5)

TIP: For "RPO=0" architectures, it is recommended to have at least one synchronous standby server.