*HPC Data MANAGEMENT Environment*

Backup Strategy

Version *1.0*

*1/11/2017*

**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version Number** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Description of Change** |
| 1.0 |  |  |  |  |  |

TABLE OF CONTENTS

[1. Purpose 4](#_Toc471901132)

[2. Introduction 4](#_Toc471901133)

[2.1. What is HPC DME? 4](#_Toc471901134)

[2.2. Intended Users 4](#_Toc471901135)

[2.3. HPC DME URL 4](#_Toc471901136)

[3. Pre-Requisites 5](#_Toc471901137)

[4. HPC DME Overwiew 5](#_Toc471901138)

[1.1. Data Management 5](#_Toc471901139)

[5. Backup strategy 5](#_Toc471901140)

# Purpose

Purpose of this document

# Introduction

## What is HPC DME?

The HPC DME, High Performance Computing Data Management Environment, is a highly adaptable and an open-ended data storage environment supporting storage and management of large data, produced from high performance computing systems. HPC DME provides capabilities for storing, managing, transferring and sharing large data across different systems securely and efficiently.

Users can store data on HPC DME for a duration defined by data storage policy, share and transfer their data such that they do not have to redistribute or maintain copies of the data on other systems by eliminating the data integrity issues. HPC DME stores and associates user defined metadata to any registered data at different levels of data life cycle, enabling the DME not only to help identify the data but also enhancing the search capabilities and to be able to attach a value factor to each dataset.

## Intended Users

The HPC data management environment is designed to meet the data storage and data management needs of NCI cancer community. Any user with a valid NCI user account can log into the HPC DME using NCI credentials.

Note: HPC DME currently only supports NCI account holders.

## HPC DME URL

Following is the HPC DME Server API URL:

<https://hpcdmeapi.nci.nih.gov/><Resource Name>

# Pre-Requisites

# HPC DME Overwiew

## Data Management

# Backup strategy

**Barman Backup and Recovery Manager for PostgreSQL**

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

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

## Introduction

In a perfect world, there would be no need for a backup. However, it is important, especially in business environments, to be prepared for when the *"unexpected"* happens. In a database scenario, the unexpected could take any of the following forms:

• data corruption;

• system failure, including hardware failure;

• human error;

• natural disaster.

## Barman Features

•Full hot physical backup of a PostgreSQL server

•Point-In-Time-Recovery (PITR)

•Management of multiple PostgreSQL servers

•Remote backup of a PostgreSQL server

•Remote recovery of a backup for a PostgreSQL server

•Management of base backups and WAL files through a catalog

•ssh support for remote operations

•rsync over ssh support for file synchronisation and transfers

•Management of retention policies for backups and WAL files

•Incremental backup

•WAL hub (get-wal)

•Compression of WAL files (bzip2, gzip or custom)

•Backup general and disk usage information

•Integration with standard archiving tools (e.g. tar)

•Local recovery

•Relocation of PGDATA and tablespaces at recovery time

•Server diagnostics for backup

•Server status and information

•Pre/Post backup hook scripts

•Local storage of metadata

•INI configuration file

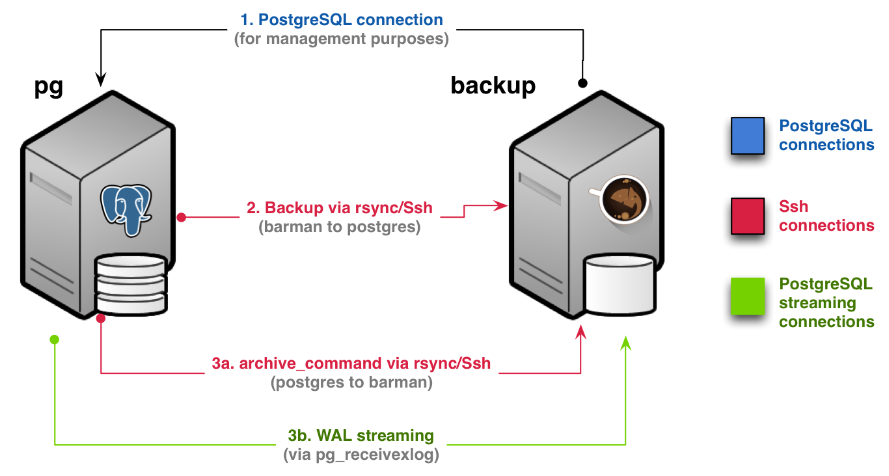
•Written in Python

•Control of bandwidth usage

## Proposed Architecture of NCI HPCDM Backup

**Option 1: Backup Server/ Database Server (pg)**

**PostgreSQL instance** (server) running on a database VM. Database backup continuously backed up to another server, called the **backup server.**



1. **Backup Server Requirements:**
2. Needed a new **Ubuntu VM** with 4 Cup’s and 32 or 64 GB Memory**.**
3. NAS volume: Needed two new 100 GB to 200 GB NAS volume which have to mount one NAS volume on DEV/UAT and other one Production database server.

Advantage of having NAS volume: NAS volume are centralized storage where data is being backed up regularly and stored in a secure location.

1. **On Database VM:**

NAS volume: We have to mount the backup server NAS volume and data is being backed up on NAS regularly and stored in a secure location.

**Option 2: Standalone Database Server (pg)**

**On Database VM:**

**PostgreSQL instance.** (server) running on a database VM. Database backup continuously backed up on same database server.

NAS volume: We have to mount the backup server NAS volume and data is being backed up regularly and stored in a secure location.

## Backup Types

**Backup Types**

Backup of servers will occur every day after regular business hours.

Full backup: Includes all the source files. This method ignores the file's archive bit until after the file is backed up. At the end of the job, all files that have been backed up have their archive bits turned off. Only one **full** backup will be done once a week followed by **differential** and/or **incremental**.

Differential backups: Includes files that have been changed since the last Full (Clear Archive Bit) or Incremental backup. If the archive bit is on, the file is backed up, and archive bit is not turned off. The next time an incremental backup is done, this file is skipped (unless it is modified again).

Incremental backups: Includes only files that have changed since the last Full (Clear Archive Bit) or Incremental backup. The next time an incremental backup is done, this file is skipped (unless it is modified again).

## Backup RotaTION

Strategy of weekly full backups with daily differential backups + WAL is a good option for HPCDM environment.

Daily backups take place on a five-day rotation.

Weekly backups take place on a five-week rotation.

Monthly backups of high availability servers occur the last calendar day of the month and are on a twelve-month rotation.

Special backups may be made for longer retention periods during special situations such as system upgrades and major projects.

**Backups**: Full weekly + WAL and Differential daily

**Media required for recovery**: Most recent full + most recent differential.

**Backup Location**: Local to NAS File System.

**Backup Tools**: Barman

**Backup User id:** barman

# Prerequisites

**System requirements**

• Linux/Unix

• Python 2.6 or 2.7

• Python modules:

• argcomplete

• argh >= 0.21.2

• psycopg2

• python-dateutil < 2.0 (since version 2.0 requires python3)

• distribute (optional)

• PostgreSQL >= 8.3

• rsync >= 3.0.4

## Configuring SSH Connectivity Between ServerS

In this section, we'll establish SSH keys for a secure passwordless connection between the **fr-s-hpcdm-gp-d** and the **fr-s-hpcdm-gp-p**, and vice versa.

Likewise, we'll establish SSH keys between the **standby-db-server** and the **fr-s-hpcdm-gp-p**, and vice versa.

This is to ensure PostgreSQL (on both database servers) and Barman can "talk" to each other during backups and restores.

For this tutorial you will need to make sure:

* User **postgres** can connect remotely from the **fr-s-hpcdm-gp-d** to the **fr-s-hpcdm-gp-p**
* User **postgres** can connect remotely from the **standby-db-server** to the **fr-s-hpcdm-gp-p**

All the commands you'll need are included here, though.

We'll show you how to do this once for setting up the connection for the user **postgres** to connect from the **fr-s-hpcdm-gp-d** to the **fr-s-hpcdm-gp-p**.

From the **fr-s-hpcdm-gp-d**, switch to user **postgres** if it's not already the current user:

* sudo su - postgres

Run the following command to generate an SSH key pair:

* ssh-keygen -t rsa

Accept the default location and name for the key files by pressing ENTER.

Press ENTER twice to create the private key without any passphrase.

Once the keys are generated, there will be a .ssh directory created under the **postgres** user's home directory, with the keys in it.

You will now need to copy the SSH public key to the authorized\_keys file under the **barman** user's .ssh directory on the **fr-s-hpcdm-gp-p**.

**Note:** Unfortunately, you can't use the ssh-copy-id barman@fr-s-hpcdm-gp-p-ip command here. That's because this command will ask for the **barman** user's password, which is not set by default. You will therefore need to copy the public key contents manually.

Run the following command to output the **postgres** user's public key contents:

* cat ~/.ssh/id\_rsa.pub

Copy the contents of the output.

Switch to the console connected to the **fr-s-hpcdm-gp-p** server and switch to the user **barman**:

* sudo su - barman

Run the following commands to create a .ssh directory, set its permissions, copy the public key contents to the authorized\_keys file, and finally make that file readable and writable:

* mkdir -p ~/.ssh

chmod 700 ~/.ssh

* echo "public\_key\_string" >> ~/.ssh/authorized\_keys
* chmod 600 ~/.ssh/authorized\_keys

Make sure you put the long public key string starting with ssh-rsa between the quotation marks, instead of public\_key\_string.

You've copied the key to the remote server.

Now, to test the connection, switch back to the **fr-s-hpcdm-gp-d** and test the connectivity from there:

* ssh barman@fr-s-hpcdm-gp-p-ip

After the initial warning about the authenticity of the remote server not being known and you accepting the prompt, a connection should be established from the **fr-s-hpcdm-gp-d** server to the **fr-s-hpcdm-gp-p**. If successful, log out of the session by executing the exit command.

* exit

**You need to set up SSH key connections three more times.** You can skip making the .ssh directory if it's already made (although this isn't necessary).

* Run the same commands again, this time from the **standby-db-server** to the **fr-s-hpcdm-gp-p**
* Run them a third time, this time originating from the **barman** user on the **fr-s-hpcdm-gp-p**, and going to the **postgres** user on the **fr-s-hpcdm-gp-d**
* Finally, run the commands to copy the key from the **barman** user on the **fr-s-hpcdm-gp-p** to the **postgres** user on the **standby-db-server**

Make sure you test the connection each way so that you can accept the initial warning about the new connection.

From **standby-db-server**:

* ssh barman@fr-s-hpcdm-gp-p-ip

From **fr-s-hpcdm-gp-p**:

* ssh postgres@fr-s-hpcdm-gp-d-ip

From **fr-s-hpcdm-gp-p**:

* ssh postgres@standby-db-server-ip

**Note:** Ensuring SSH connectivity between all three servers is a requirement for backups to work.

## PostgreSQL connection

You need to make sure that the backup server allows connection to the PostgreSQL server on pg

as superuser (postgres).

barman@fr-s-hpcdm-gp-p$ **psql -c 'SELECT version()' -U postgres -h pg**

**Note** As of version 1.1.2, Barman honours the application\_name connection option for PostgreSQL

# Installing Barman

Install Barman on the backup server, which will both control and store our backups

**On Ubuntu using packages**

Barman can be installed on Ubuntu Linux systems using packages.

**Note**

Installing Barman is as simple as typing as root user:

**apt-get install barman**

**Note:** Installing Barman will create an operating system user called **barman**. This account does not have a password; you can switch to this user from your sudo user account.

# Configuring Barman for Backups

You will now configure Barman to back up your main PostgreSQL server.

The main configuration file for BARMAN is /etc/barman.conf. The file contains a section for global parameters, and separate sections for each server that you want to back up. The default file contains a section for a sample PostgreSQL server called **main**, which is commented out. You can use it as a guide to set up other servers you want to back up.

A semicolon (;) at the beginning of a line means that line is commented out. Just like with most Linux-based applications, a commented-out configuration parameter for Barman means the system will use the default value unless you uncomment it and enter a different value.

One such parameter is the configuration\_files\_directory, which has a default value of /etc/barman.d. What this means is, when enabled, Barman will use the .conf files in that directory for different Postgres servers' backup configurations. If you find the main file is getting too lengthy, feel free to make separate files for each server you want to back up.

For the sake of simplicity in this tutorial, we will put everything in the default configuration file.

Open /etc/barman.conf in a text editor as your **sudo user** (user **barman** has only read access to it):

* sudo vi /etc/barman.conf

The global parameters are defined under the [barman] section. Under this section, make the following changes. The finished values are shown below the bullet points:

* Uncomment the line for compression and keep the default value of gzip. This means the PostgreSQL WAL files - when copied under the backup directory - will be saved in gzip compressed form
* Uncomment the line for reuse\_backup and keep the default value of link. When creating full backups of the PostgreSQL server, Barman will try to save space in the backup directory by creating file-level incremental backups. This uses rsync and hard links. Creating incremental full backups has the same benefit of any data de-duplication method: savings in time and disk space
* Uncomment the line for immediate\_checkpoint and set its value to true. This parameter setting ensures that when Barman starts a full backup, it will request PostgreSQL to perform a CHECKPOINT. Checkpointing ensures any modified data in PostgreSQL's memory cache are written to data files. From a backup perspective, this can add some value because BARMAN would be able to back up the latest data changes
* Uncomment the line for basebackup\_retry\_times and set a value of 3. When creating a full backup, Barman will try to connect to the PostgreSQL server three times if the copy operation fails for some reason
* Uncomment the line for basebackup\_retry\_sleep and keep the default value of 30. There will be a 30-second delay between each retry
* Uncomment the line for last\_backup\_maximum\_age and set its value to 1 DAYS

The new settings should look like this exactly:

Excerpts from /etc/barman.conf

[barman]

barman\_home = /var/lib/barman

barman\_user = barman

log\_file = /var/log/barman/barman.log

compression = gzip

reuse\_backup = link

immediate\_checkpoint = true

basebackup\_retry\_times = 3

basebackup\_retry\_sleep = 30

last\_backup\_maximum\_age = 1 DAYS

What we are doing here is this:

* Keeping the default backup location
* Specifying that backup space should be saved. WAL logs will be compressed and base backups will use incremental data copying
* Barman will retry three times if the full backup fails halfway through for some reason
* The age of the last full backup for a PostgreSQL server should not be older than 1 day

At the end of the file, add a new section. Its header should say [fr-s-hpcdm-gp-d] in square brackets. (If you want to back up more database servers with Barman, you can make a block like this for each server and use a unique header name for each.)

This section contains the connection information for the database server, and a few unique backup settings.

Add these parameters in the new block:

Excerpt from /etc/barman.conf

[fr-s-hpcdm-gp-d]

description = "Example of PostgreSQL Database (via Ssh)"

ssh\_command = ssh postgres@fr-s-hpcdm-gp-d

conninfo = host=fr-s-hpcdm-gp-d user=barman dbname=postgres

backup\_method = rsync

reuse\_backup = link

archiver = on

The retention\_policy settings mean that Barman will overwrite older full backup files and WAL logs automatically, while keeping enough backups for a recovery window of 7 days. That means we can restore the entire database server to any point in time in the last seven days. **For a production system, you should probably set this value higher so you have older backups on hand.**

You'll need to use the IP address of the **fr-s-hpcdm-gp-d** in the ssh\_command and conninfo parameters. Otherwise, you can copy the above settings exactly.

The final version of the modified file should look like this, minus all the comments and unmodified settings:

Excerpts from /etc/barman.conf

[barman]

barman\_home = /var/lib/barman

barman\_user = barman

log\_file = /var/log/barman/barman.log

compression = gzip

reuse\_backup = link

immediate\_checkpoint = true

basebackup\_retry\_times = 3

basebackup\_retry\_sleep = 30

last\_backup\_maximum\_age = 1 DAYS

[fr-s-hpcdm-gp-d]

description = "Example of PostgreSQL Database (via Ssh)"

ssh\_command = ssh postgres@fr-s-hpcdm-gp-d

conninfo = host=fr-s-hpcdm-gp-d user=barman dbname=postgres

backup\_method = rsync

reuse\_backup = link

archiver = on

Save and close the file.

Next, we'll make sure our **fr-s-hpcdm-gp-d** is configured to make backups.

**Test the configuration by executing:**

barman@fr-s-hpcdm-gp-p$ **barman show-server main**

**Continuous WAL archiving**

Edit the postgresql.conf file of the PostgreSQL instance on the pg database and activate the

archive mode:

wal\_level = 'archive' # For PostgreSQL >= 9.0

archive\_mode = on

archive\_command = 'rsync -a %p barman@fr-s-hpcdm-gp-p: INCOMING\_WALS\_DIRECTORY/%f'

Make sure you change the INCOMING\_WALS\_DIRECTORY placeholder with the value returned by the

barman show-server main command above.

In case you use Hot Standby, wal\_level must be set to hot\_standby.

Restart the PostgreSQL server.

In order to test that continuous archiving is on and properly working, you need to check both the

PostgreSQL server 3 and the backup server (in particular, that the WAL files are collected in the

destination directory).

**Listing the servers**

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

barman@fr-s-hpcdm-gp-p$ **barman list-server**

# On Database server - Configuring the postgresql.conf File

There is one last configuration to be made on the **fr-s-hpcdm-gp-d**, to switch on backup (or archive) mode.

First, we need to locate the value of the incoming backup directory from the **fr-s-hpcdm-gp-p**. On the Barman server, switch to the user **barman**:

* sudo su - barman

Run this command to locate the incoming backup directory:

* barman show-server fr-s-hpcdm-gp-d | grep incoming\_wals\_directory

This should output something like this:

barman show-server command output

incoming\_wals\_directory: /var/lib/barman/fr-s-hpcdm-gp-d/incoming

Note down the value of incoming\_wals\_directory; in this example, it's /var/lib/barman/fr-s-hpcdm-gp-d/incoming.

Now switch to the **fr-s-hpcdm-gp-d** console.

Switch to the user **postgres** if it's not the current user already.

Open the postgresql.conf file in a text editor:

* vi $PGDATA/postgresql.conf

Make the following changes to the file:

* Uncomment the wal\_level parameter and set its value to archive instead of minimal
* Uncomment the archive\_mode parameter and set its value to on instead of off
* Uncomment the archive\_command parameter and set its value to 'rsync -a %p barman@fr-s-hpcdm-gp-p-ip:/var/lib/barman/fr-s-hpcdm-gp-d/incoming/%f' instead of ''. Use the IP address of the Barman server. If you got a different value for incoming\_wals\_directory, use that one instead

Excerpts from postgresql.conf

wal\_level = archive # minimal, archive, hot\_standby, or logical

archive\_mode = on # allows archiving to be done

archive\_command = 'rsync -a %p barman@fr-s-hpcdm-gp-p-ip:/var/lib/barman/fr-s-hpcdm-gp-d/incoming/%f' # command to use to archive a logfile segment

Switch back to your **sudo user**.

Restart PostgreSQL:

* sudo systemctl restart postgresql

**Note:** If you are configuring an existing production PostgreSQL instance, there's a good chance these three parameters will be set already. You will then have to add/modify only the archive\_command parameter so PostgreSQL sends its WAL files to the backup server.

# Testing Barman

It's now time to check if Barman has all the configurations set correctly and can connect to the **fr-s-hpcdm-gp-d**.

On the **fr-s-hpcdm-gp-p**, switch to the user **barman** if it's not the current user. Run the following command to test the connection to your main database server:

* barman check fr-s-hpcdm-gp-d

Note that if you entered a different name between the square brackets for the server block in the /etc/barman.conf file in Step 5, you should use that name instead.

If everything is okay, the output should look like this:

barman check command output

Server fr-s-hpcdm-gp-d:

PostgreSQL: OK

archive\_mode: OK

wal\_level: OK

archive\_command: OK

continuous archiving: OK

directories: OK

retention policy settings: OK

backup maximum age: FAILED (interval provided: 1 day, latest backup age: No available backups)

compression settings: OK

minimum redundancy requirements: OK (have 0 backups, expected at least 0)

ssh: OK (PostgreSQL server)

not in recovery: OK

Don't worry about the backup maximum age FAILED state. This is happening because we have configured Barman so that the latest backup should not be older than 1 day. There is no backup made yet, so the check fails.

If any of the other parameters are in a FAILED state, you should investigate further and fix the issue before proceeding.

There can be multiple reasons for a check to fail: for example, Barman not being able to log into the Postgres instance, Postgres not being configured for WAL archiving, SSH not working between the servers, etc. Whatever the cause, it needs to be fixed before backups can happen. Run through the previous steps and make sure all the connections work.

To get a list of PostgreSQL servers configured with Barman, run this command:

* barman list-server

Right now it should just show:

Output

fr-s-hpcdm-gp-d - Main DB Server

# Creating the Backup

Create a backup manually.

Run the following command as the **barman** user on the **fr-s-hpcdm-gp-p** to make your first backup:

* barman backup fr-s-hpcdm-gp-d

Again, the fr-s-hpcdm-gp-d value is what you entered as the head of the server block in the /etc/barman.conf file in Step 5.

This will initiate a full backup of the PostgreSQL data directory. Since our instance has only one small database with two tables, it should finish very quickly.

Output

Starting backup for server fr-s-hpcdm-gp-d in /var/lib/barman/fr-s-hpcdm-gp-d/base/20151111T051954

Backup start at xlog location: 0/2000028 (000000010000000000000002, 00000028)

Copying files.

Copy done.

Asking PostgreSQL server to finalize the backup.

Backup size: 26.9 MiB. Actual size on disk: 26.9 MiB (-0.00% deduplication ratio).

Backup end at xlog location: 0/20000B8 (000000010000000000000002, 000000B8)

Backup completed

Processing xlog segments for fr-s-hpcdm-gp-d

Older than first backup. Trashing file 000000010000000000000001 from server fr-s-hpcdm-gp-d

000000010000000000000002

000000010000000000000002.00000028.backup

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

barman@fr-s-hpcdm-gp-p$ **barman backup main**

barman@fr-s-hpcdm-gp-p$ **barman backup all**

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

**Viewing the list of backups for a server**

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

barman@fr-s-hpcdm-gp-p$ **barman list-backup main**

you can get a listing of the available backups for all your servers, using the all

target for the server:

barman@fr-s-hpcdm-gp-p$ **barman list-backup all**

### Listing Backups

There is a specific Barman command to list all the backups for a server. That command is barman list-backup. Run the following command to see what it returns for our fr-s-hpcdm-gp-d:

barman@fr-s-hpcdm-gp-p:~$ **barman list-backup fr-s-hpcdm-gp-d**

fr-s-hpcdm-gp-d 20170113T061316 - Fri Jan 13 06:13:44 2017 - Size: 5.1 GiB - WAL Size: 0 B

* The first part of the output is the name of the server. In this case, fr-s-hpcdm-gp-d
* The second part - a long alphanumeric value - is the backup ID for the backup. A backup ID is used to uniquely identify any backup Barman makes. In this case, it's 20151111T051954. **You will need the backup ID for the next steps**
* The third piece of information tells you when the backup was made
* The fourth part is the size of the base backup (26.9 MB in this case)
* The fifth and final part of the string gives the size of the the WAL archive backed up

To see more details about the backup, execute this command using the name of the server, and the backup ID (20151111T051954 in our example) from the previous command:

**barman show-backup fr-s-hpcdm-gp-d 20170113T061316**

A detailed set of information will be shown:

Output

Backup 20170113T061316:

Server Name : fr-s-hpcdm-gp-d

Status : DONE

PostgreSQL Version: 90503

PGDATA directory : /var/lib/postgresql/9.5/main

Base backup information:

Disk usage : 5.1 GiB

Timeline : 1

Begin WAL : 00000001000000060000003B

End WAL : 00000001000000060000003B

WAL number : 1

Begin time : 2017-01-13 06:13:16.763699

End time : 2017-01-13 06:13:44.677992

Begin Offset : 40

End Offset : 192

Begin XLOG : 6/3B000028

End XLOG : 6/3B0000C0

WAL information:

No of files : 0

Disk usage : 0 B

Last available : 00000001000000060000003B

Catalog information:

Retention Policy: not enforced

Previous Backup : - (this is the oldest base backup)

Next Backup : - (this is the latest base backup)

barman@fr-s-hpcdm-gp-p:~$ barman list-files fr-s-hpcdm-gp-d backup-id

# Scheduling Backups

Ideally your backups should happen automatically on a schedule.

In this step we'll automate our backups, and we'll tell Barman to perform maintenance on the backups so files older than the retention policy are deleted. To enable scheduling, run this command as the **barman** user on the **fr-s-hpcdm-gp-p** (switch to this user if necessary):

* crontab -e

This will open a crontab file for the user **barman**. Edit the file, add these lines, then save and exit:

cron

30 23 \* \* \* /usr/bin/barman backup fr-s-hpcdm-gp-d

\* \* \* \* \* /usr/bin/barman cron

The first command will run a full backup of the **fr-s-hpcdm-gp-d** every night at 11:30 PM. (If you used a different name for the server in the /etc/barman.conf file, use that name instead.)

The second command will run every minute and perform maintenance operations on both WAL files and base backup files.

# Simulating a "Disaster"

You will now see how you can restore from the backup you just created. To test the restoration, let's first simulate a "disaster" scenario where you have lost some data.

**We're dropping a table here. Don't do this on a production database!**

Go back to the **fr-s-hpcdm-gp-d** console and switch to the user **postgres** if it's not already the current user.

Start the psql utility:

* psql

From the psql prompt, execute the following command to switch the database context to ICAT:

* \connect ICAT;

Next, list the tables in the database:

* \dt

The output will show the tables you created at the beginning of this tutorial:

Output

List of relations

List of relations

Schema | Name | Type | Owner

--------+-----------------------------------+-------+-------

public | HPC\_DATA\_OBJECT\_DOWNLOAD\_CLEANUP | table | irods

public | HPC\_EVENT | table | irods

public | HPC\_EVENT\_HISTORY | table | irods

public | HPC\_NOTIFICATION\_DELIVERY\_RECEIPT | table | irods

public | HPC\_NOTIFICATION\_SUBSCRIPTION | table | irods

public | HPC\_SYSTEM\_ACCOUNT | table | irods

public | HPC\_USER | table | irods

public | HPC\_USER\_QUERY | table | irods

You will see that only mytesttable1 remains.

This is the type of data loss situation where you would want to restore from a backup. In this case, you will restore the backup to a separate server: the **standby-db-server**

# Restoring or Migrating to a Remote Server

You can follow this section to restore a backup, or to migrate your latest PostgreSQL backup to a new server.

Go to the **standby-db-server**.

First, stop the PostgreSQL service as the sudo user. (The restart will choke if you try to run the restoration while the service is running.)

* sudo systemctl stop postgresql

To restore a whole server, issue the following command:

barman@fr-s-hpcdm-gp-p$ **barman recover main 20110920T185953 /path/to/recover/directory**

where 20110920T185953 is the ID of the backup to be restored. When this command completes

succesfully, /path/to/recover/directory contains a complete data directory ready to be started

Once the service stops, go to the **fr-s-hpcdm-gp-p**. Switch to the user **barman** if it's not already the current user.

Let's locate the details for the latest backup:

* barman show-backup fr-s-hpcdm-gp-d latest

Output

Backup 20160114T173552:

Server Name : fr-s-hpcdm-gp-d

Status : DONE

PostgreSQL Version : 90405

PGDATA directory : /var/lib/pgsql/9.4/data

Base backup information:

Begin time : <date>

End time : <date>

From the output, note down the backup ID printed on the first line (20160114T173552 above). If the latest backup has the data you want, you can use latest as the backup ID.

Also check when the backup was made, from the Begin time field (<date> above).

Next, run this command to restore the specified backup from the **fr-s-hpcdm-gp-p** to the **standby-db-server**:

* barman recover --target-time "Begin time" --remote-ssh-command "ssh postgres@standby-db-server-ip" fr-s-hpcdm-gp-d backup-id /var/lib/pgsql/9.4/data

There are quite a few options, arguments, and variables here, so let's explain them.

* --target-time "Begin time": Use the begin time from the show-backup command
* --remote-ssh-command "ssh postgres@standby-db-server-ip": Use the IP address of the **standby-db-server**
* fr-s-hpcdm-gp-d: Use the name of the database server from your /etc/barman.conf file
* backup-id: Use the backup ID from the show-backup command, or use latest if that's the one you want
* /var/lib/pgsql/9.4/data: The path where you want the backup to be restored. This path will become the new data directory for Postgres on the standby server. Here, we have chosen the default data directory for Postgres in CentOS. For real-life use cases, choose the appropriate path

For a successful restore operation, you should receive output like this:

Output from Barman Recovery

Starting remote restore for server fr-s-hpcdm-gp-d using backup backup-id

Destination directory: /var/lib/pgsql/9.4/data

Doing PITR. Recovery target time: Begin time

Copying the base backup.

Copying required WAL segments.

Generating recovery.conf

Identify dangerous settings in destination directory.

IMPORTANT

These settings have been modified to prevent data losses

postgresql.conf line 207: archive\_command = false

Your PostgreSQL server has been successfully prepared for recovery!

Now switch to the **standby-db-server** console again. As the **sudo user**, start the PostgreSQL service:

* sudo systemctl start postgresql-9.4.service

That should be it!

Let's verify that our database is up. Switch to user **postgres** and start the psql utility:

* sudo su - postgres
* psql

Switch the database context to ICAT and list the tables in it:

* \connect ICAT;

ICAT=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------------------------------+-------+-------

public | HPC\_DATA\_OBJECT\_DOWNLOAD\_CLEANUP | table | irods

public | HPC\_EVENT | table | irods

public | HPC\_EVENT\_HISTORY | table | irods

public | HPC\_NOTIFICATION\_DELIVERY\_RECEIPT | table | irods

public | HPC\_NOTIFICATION\_SUBSCRIPTION | table | irods

public | HPC\_SYSTEM\_ACCOUNT | table | irods

public | HPC\_USER | table | irods

public | HPC\_USER\_QUERY | table | irods

public | r\_coll\_main | table | irods

public | r\_data\_main | table | irods

public | r\_grid\_configuration | table | irods

public | r\_meta\_main | table | irods

public | r\_microsrvc\_main | table | irods

public | r\_microsrvc\_ver | table | irods

public | r\_objt\_access | table | irods

public | r\_objt\_audit | table | irods

public | r\_objt\_deny\_access | table | irods

public | r\_objt\_metamap | table | irods

public | r\_quota\_main | table | irods

public | r\_quota\_usage | table | irods

public | r\_resc\_group | table | irods

public | r\_resc\_main | table | irods

public | r\_rule\_base\_map | table | irods

public | r\_rule\_dvm | table | irods

public | r\_rule\_dvm\_map | table | irods

public | r\_rule\_exec | table | irods

public | r\_rule\_fnm | table | irods

public | r\_rule\_fnm\_map | table | irods

public | r\_rule\_main | table | irods

public | r\_server\_load | table | irods

public | r\_server\_load\_digest | table | irods

public | r\_specific\_query | table | irods

public | r\_ticket\_allowed\_groups | table | irods

public | r\_ticket\_allowed\_hosts | table | irods

public | r\_ticket\_allowed\_users | table | irods

public | r\_ticket\_main | table | irods

public | r\_tokn\_main | table | irods

public | r\_user\_auth | table | irods

public | r\_user\_group | table | irods

public | r\_user\_main | table | irods

(2 rows)

The list should show two tables in the database. In other words, you have just recovered the dropped table.

Depending on your larger recovery strategy, you may now want to fail over to the **standby-db-server**, or you may want to check that the restored database is working, and then run through this section again to restore to the **fr-s-hpcdm-gp-d**.

To restore to any other server, just make sure you've installed PostgreSQL and made the appropriate connections to the Barman server, and then follow this section using the IP address of your target recovery server