PostgreSQL Plugin

Bacula Systems Documentation

Contents

Contents

This chapter aims at presenting the reader with information about the Bacula Enterprise PostgreSQL Plugin. The document briefly defines the scope of its operations, describes the target technology of the Plugin, and presents its main features and various techniques and strategies to backup PostgreSQL with Bacula Enterprise.

1 Scope

This Plugin is available for 32 and 64-bit Linux platforms, and supports all officially supported PostgreSQL versions since version 8.4.

2 Features

The PostgreSQL Plugin is designed to simplify backup and restore procedure of PostgreSQL clusters, so that the backup administrator does not need to know about internals of Postgres backup techniques or write complex scripts. The Plugin will automatically take care of backing up essential information such as configuration, users definition or tablespace.

The PostgreSQL Plugin supports both Dump and Point In Time Recovery (PITR) backup techniques.

The PostgreSQL Plugin is compatible with Copy/Migration jobs. Read the CopyMigrationJobsReplication for more information.

3 Backup Strategies

The following article presents the comparison of backup strategies for the PostgreSQL Plugin.

3.1 Choosing Between PITR and Dump

The following table helps choosing between backup techniques supported by the Bacula Enterprise PostgreSQL Plugin. Major functionalities such as being able to restore databases to any point in time, or being able to filter objects during backup or restore should be used to guide through the backup design. It is quite common to combine Dump and PITR techniques for the same Cluster. In the table, the Custom format corresponds to the Custom Dump format of pg_dump and the Dump format of our Plugin corresponds to the plain format of pg_dump.

Note: Regardless the backup method used, no temporary local disk space is necessary to save any temporary file.

Table 1: PostgreSQL dump vs PITR backup

	Custom 1	Dump	PITR
Can restore directly a single object (table, schema, . . .)	Yes	No	No
Backup speed	Slow	Slow	Fast
Restore speed	Slow	Very Slow	Fast
Backup size	Small	Small	Big
Can restore at any point in time	No	No	Yes
Incremental/Differential support	No	No	Yes
Can restore in parallel	Yes 2	No	–
Online backup	Yes	Yes	Yes
Consistent	Yes	Yes	Yes
Can restore to previous major version of PostgreSQL	No	Yes 3	No
Can restore to newer major version of PostgreSQL	Yes	Yes	No

1 Custom Dump format is the default.

2 Run the most time-consuming parts of pg_restore - those which load data, create indexes, or create constraints - using multiple concurrent jobs. This option can dramatically reduce time to restore a large database to a server running on a multiprocessor machine. It requires to store the Dump to the disk first.

3 To restore a SQL plain Dump to a previous version of PostgreSQL, you might have to edit the SQL file if you are using some features that are not available in the previous version. Generally, restoring to a previous version of PostgreSQL is not supported or not guaranteed.

4 Installation

4.1 Prerequisites

As with all Bacula plugins, the **Plugin Directory** directive in the **FileDaemon** resource of the bacula-fd.conf file needs to be set:

```
FileDaemon {
  Name = test-fd
  ...
  Plugin Directory = /opt/bacula/plugins
}
```

4.2 PostgreSQL Installation with BIM

In order to install the PostgreSQL Plugin with BIM, install the File Daemon with BIM and choose to install the PostgreSQL Plugin during the FD installation.

Click here for more details on the plugin installation process with BIM.

4.3 PostgreSQL Plugin Installation with Package Manager

The PostgreSQL Plugin is available as a Bacula Enterprise package for all supported platforms.

The Plugin must be installed on the primary node in the PostgreSQL Cluster. The PostgreSQL client software package, usually "postgresql-client", should also be installed as it provides tools such as pg_dump and psql which are used by the plugin, or may be useful for diagnostic purposes.

In order to get information and/or data from the PostgreSQL database, the Bacula Enterprise Plugin uses PostgreSQL standard commands using the unix "postgres" user. This user account needs to be able to connect to the Cluster without interactive password dialog. Such a configuration (which is the default one) can be achieved using the following entry in pg_hba.conf. The following entry should be the first one in the list:

```
local   all   postgres        ident
```

If remote databases need to be accessed, or disabling the interactive password authentication method is not an option, it is possible to define a pgpass file.

5 Configuration

The following chapter aims at presenting how to configure the PostgreSQL Plugin.

5.1 Automatic Object Integration

Since Bacula version 16.0.7, a new solution has been introduced, so that each object can be backed up separately with different Jobs to maximize the throughput and the resiliency. It is highly recommended to use this new solution for that purpose - Automatic Object Integration (Scan Plugin). See an example for PostgreSQL.

5.2 PITR Configuration

PostgreSQL Server Configuration for PITR

In order to use the Point In Time Recovery feature of PostgreSQL, WAL Archiving needs to be enabled. The procedure differs between major PostgreSQL version, so we advise to read the PostgreSQL documentation corresponding to the Cluster version; for PostgreSQL version 9.1, for example, it can be found here: http://www.postgresql.org/docs/9.1/static/continuous-archiving.html

Basically, on 8.4, the `archive_command` and the `archive_mode` settings need to be configured.

```
# on 8.3 - 8.4
archive_mode = on
archive_command = 'test ! -f /mnt/waldir/%f && cp %p /mnt/waldir/%f'
```

For 9.x, it is needed to configure `archive_command`, `wal_level`, and `archive_mode`.

```
# on 9.0 - 9.1
wal_level = archive
archive_mode = on
archive_command = 'test ! -f /mnt/waldir/%f && cp %p /mnt/waldir/%f'
```

The `/mnt/waldir` directory should be purged from time to time when backup jobs are successful. We do not recommend to remove WAL files just after a backup job. If something is going wrong with the backup job and files are no longer on the database server, it will not be possible to easily re-start the backup job.

```
# This example will remove WAL files older than 14 days
find /mnt/waldir -type f -mtime +14 -exec rm -f {} \;
```

As shown below, the PostgreSQL plugin needs to know where WAL files are stored after archiving. In this example, `archive_dir` would point to `/mnt/waldir`.

Note: It may be useful to compress WAL files in the `archive_command` using something like:

```
```

archive_command = ‘test ! -f /mnt/waldir/%f.gz && gzip -c %p > /mnt/waldir/%f.gz’

In this case, the **restore_command** in `recovery.conf` will need to be modified during the restore, as the PostgreSQL Plugin will not be able to reverse the custom `archive_command` automatically. It is good practice to put the needed `restore_command` command as comment into the `postgresql.conf`.

PostgreSQL Plugin Configuration for PITR

With the PITR option, the PostgreSQL Plugin uses Accurate mode information to handle Differential backups, thus the **Accurate** option in the backup Job resource needs to be enabled. If Accurate mode is not used in Differential level backups, orphan data files in the Cluster directory may occur when restoring. Note that using Accurate mode is not mandatory only Full and Incremental backups are planned.

```
Job {
 Name = "Postgresql-PITR"
 Client = laptop1-fd
 Fileset = FS_postgresql
 Accurate = yes
 ...
}

Fileset {
 Name = FS_postgresql
 Include {
   Options {
     Signature = MD5
     Compression = GZIP
   }
   Plugin = "postgresql: mode=pitr"
 }
}
```

In the PITR mode, the PostgreSQL Plugin also accepts the parameters listed here.

If PostgreSQL clusters are planned to be restored using the file relocation feature of Bacula, it is useful to use the **PrefixLinks** directive in the prepared Restore Job.

For example, if symbolic links in the PGDATA cluster directory are used, like for pg_wal[1] or tablespaces, the default Restore Job will re-create those symbolic links pointing to their original locations and not the restored datas directory, which will break PostgreSQLs recovery process.

On the other hand, if restored files are planned to be moved to their original locations later, outside of Bacula's restore process, all files linked with absolute names will be broken.

[1] In old versions of postgresql (<= 9.x), pg_wal was pg_xlog - it was renamed in version 10.

PostgreSQL Plugin Options in PITR Mode

Table 2: PostgreSQL Plugin Options in PITR Mode

Option	Comment	Default	Example
mode=pit	Needed to enable PITR backup.	custom	
abort_on_	Abort the job after a connection error with PostgreSQL (available with Bacula Enterprise 8.2.0 and later).	not set	abort_on_error
archive_d	Should point to where you are archiving WAL with the `archive_command`.	pg_wa	
bin_dir	PostgreSQL binaries location.		bin_dir=/opt/pg9.1/bin
fast_backu	Use fast backup option in pg_backup_start() procedure. It will create a peak of IO if used.	false	fast_backup
pgpass	Path to PostgreSQL password file.		pgpass=/etc/pgpass
user	PostgreSQL Unix super user. A standard user cannot be used here.	postgres	user=dba
service	PostgreSQL connection information.		service=main
timeout	Specify a custom timeout (in secs) for commands sent to PostgreSQL.	300	timeout=600
tmp_dir	Where the plugin will create files and scripts for the database backup (available with Bacula Enterprise 6.6.6 and later).	`/tmp`	tmp_dir=/othertmp
use_sudo	Use sudo to execute Postgresql commands.		use_sudo
unix_user	Use specific unix user to execute Postgresql commands. The unix user owning the cluster is required.	postgres	unix_user=bob

Backup Level in PITR

When using PITR mode, depending on the Job level, the PostgreSQL Plugin will do the following:

- For a **Full** backup, the Plugin will backup the data directory and all WAL files generated during the backup.
- During an **Incremental** backup, the Plugin will force the switch of the current WAL, and will backup WAL files generated since the previous backup.
- During a **Differential** backup, the Plugin will backup data files that changed since the latest Full backup, and it will backup WAL files generated during the backup.

Note: It is not possible to run two concurrent Full or Differential jobs at the same time.

where (1) is period between the latest Incremental and the new Full or Differential backup.

Note: It is recommended to visit ScheduleConsiderationForPITR for further information.

Note that replaying a long list of WAL files may take considerable time on a large system with lot of activities.

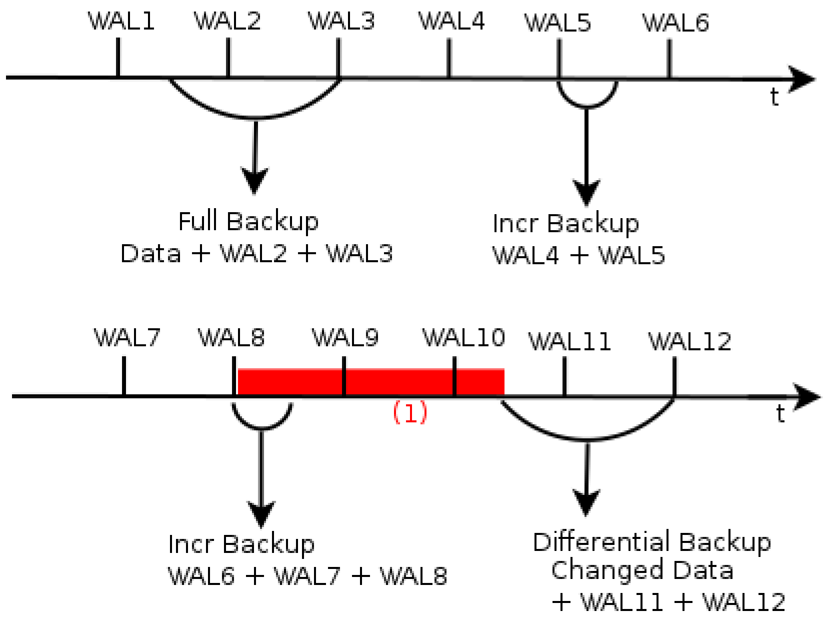


Fig. 1: Backup Level Impact in PITR Mode

Schedule Consideration for PITR

In order to be able to restore to any point in time between the latest Incremental and a previous Full or Differential backup (see the (1) area in figure pg-level), it is good practice to schedule an Incremental to the same time of Full or Differential backups. The **Maximum Concurrent Jobs** setting on the Client and the Job resource should allow to run two jobs concurrently.

```
Schedule {
  Name = SCH_PostgreSQL

  Run = Full 1st sun at 23:05
  Run = Differential 2nd-5th sun at 23:05
  Run = Incremental mon-sun at 23:05
}
```

Without this schedule configuration, it will not be possible to restore to a specific point in time during the period between the latest Incremental backup and the next Full or Differential backup. Note that disabling `Allow Duplicate Jobs` prevents starting two Full backup Job at the same time.

5.3 Dump Configuration

With the Dump option, Bacula cannot perform Incremental or Differential backup.

```
Job {
 Name = "Postgresql-dump"
 Client = laptop1-fd
 Fileset = FS_postgresql_dump
 ...
}
```

(continues on next page)

(continued from previous page)

```
Fileset {
 Name = FS_postgresql_dump
 Include {
   Options {
     Signature = MD5
     Compression = GZIP
   }
   Plugin = postgresql
 }
}
```

With the above example, the Plugin will detect and backup all databases of the Cluster.

```
Fileset {
 Name = FS_postgresql
 Include {
   Options {
     Signature = MD5
     Compression = GZIP
   }
   Plugin = "postgresql: database=bacula"
   Plugin = "postgresql: database=master"
 }
}
```

In this example, the Plugin will backup only the databases `bacula` and `master`.

In Dump mode, the PostgreSQL Plugin also accepts the parameters listed here.

```
Fileset {
 Name = FS_postgresql_dump
 Include {
   Options {
     Signature = MD5
   }
   Plugin = "postgresql: use_sudo user=rob dump_opt=\"-T temp\""
 }
}
```

In this example, the PostgreSQL Plugin will use the Unix account “rob” to perform a Custom Dump backup with the PostgreSQL “rob” account excluding tables named “temp”.

In order to use the `sudo` wrapper, it is needed to comment out the following option in `/etc/sudoers`.

```
Defaults requiretty
```

PostgreSQL Plugin Options in Dump Mode

Table 3: PostgreSQL Plugin Options in Dump Mode

Option	Comment	Default	Example
dump_opt	This string will be passed to the pg_dump command. 1	-c -b -F p	dump_opt="-c"
user	PostgreSQL user to use for PostgreSQL commands.	postgres	user=rob
unix_user	Unix user to use for PostgreSQL commands. 2	set to `user`	unix_user=pg1
service	pg_service to use for PostgreSQL commands.		service=main
pgpass	Path to PostgreSQL password file.		pg-pass=/etc/pgpass
use_sudo	Use sudo instead to run PostgreSQL commands (when not root).		use_sudo
compress	Use pg_dump compression level. 0-9, 0 is off.	0	compress=5
database	Will backup on databases matching this string.		database=prod*
exclude	Will exclude databases matching this string. 5		exclude=test*
bin_dir	PostgreSQL binaries location.		bin_dir=/opt/pg9.1/bin
tmp_dir	Where the plugin will create files and scripts for the database backup. 3	/tmp	tmp_dir=/othertmp
abort_on_err	Abort the job after a connection error with PostgreSQL. 4	not set	abort_on_error
timeout	Specify a custom timeout (in secs) for commands sent to PostgreSQL.	60 seconds	timeout=120

1 The dump_opt option cannot be used to backup remote servers. Please use PGSERVICE instead

2 Available with Bacula Enterprise 8.4.12 and later.

3 Available with Bacula Enterprise 6.6.6 and later.

4 Available with Bacula Enterprise 8.2.0 and later.

5 Available with Bacula Enterprise 14.0.5 and later.

Note: The `database` and `exclude` options support regex.

5.4 Service Connection Information

The connection service file allows PostgreSQL connection parameters to be associated with a single service name. That service name can then be specified by a PostgreSQL connection, and the associated settings will be used.

The connection service file can be a per-user service file at `` /.pg_service.conf``, can have its location specified by the environment variable **PGSERVICEFILE**, or it can be a system-wide file at `/etc/pg_service.conf` or in the directory specified by the environment variable **PGSYSCONFDIR**. If service definitions with the same name exist in the user and the system files, the user file takes precedence.

The file uses an INI file format where the section name represents the service name and the parameters are connection parameters; see http://www.postgresql.org/docs/9.1/static/libpq-connect.html for a list. For example:

```
# comment
[main]
port=5433
```

If a password is needed to connect, the pgpass option in the Plugin command string can be used to make the Plugin define the needed **PGPASSFILE** environment variable.

5.5 Testing Database Access

The estimate command can be used to verify that the PostgreSQL Plugin is well configured.

```
* estimate listing job=pg-test
...
```

If estimate or the job output displays the following error:

```
Error: Can't reach PostgreSQL server to get database config.
```

the next steps should be to verify that the Bacula Enterprise PostgreSQL plugin can retrieve information using the psql command as “postgres” user on the Client.

To verify if the “postgres” user can connect to the PostgreSQL Cluster, the psql -l command can be used, and it should list all databases in the Cluster:

```
postgres%  psql -l
   List of databases
   Name     |  Owner   |
-----------+----------+
postgres   |   xxx
template0  |   xxx
template1  |   xxx
```

If options such as **-h localhost** are needed on the psql command line, a service file as described in **servicecon** will be required.

6 Operations

The following chapter aims at presenting possible operations with the PostgreSQL Plugin.

6.1 Backup

Estimate Information

The estimate command will display all information found by the PostgreSQL Plugin.

Note: In Dump mode, Bacula can not compute the Dump size for databases, so it will display database size instead.

Backup Information in Dump Mode

The PostgreSQL Plugin will generate the following files for a Cluster containing the single database "test":

```
@PG/main/roles.sql
@PG/main/postgresql.conf
@PG/main/pg_hba.conf
@PG/main/pg_ident.conf
@PG/main/tablespaces.sql

@PG/main/test/createdb.sql
@PG/main/test/schema.sql
@PG/main/test/data.sqlc
```

Table 4: Backup Content in Dump Mode

File	Context	Comment
roles.sql	global	List of all users, their password and specific options
postgresql.conf	global	PostgreSQL cluster configuration
pg_hba.conf	global	Client connection configuration
pg_ident.conf	global	Client connection configuration
tablespaces.sql	global	Tablespaces configuration for the PostgreSQL cluster
createdb.sql	database	Database creation script
schema.sql	database	Schema database creation script
data.sqlc	database	Database data in custom format, contains everything needed to restore
data.sql	database	Database data in dump format

6.2 Restore

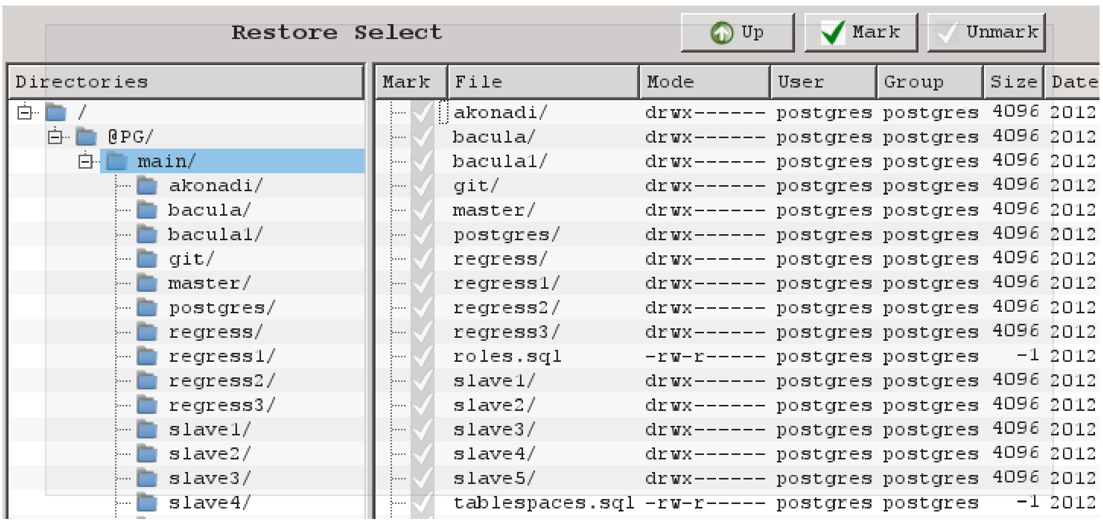
Restoring Using Dumps

Restoring Users and Roles

To restore roles and users to a PostgreSQL Cluster, the `roles.sql` file located in `/@PG/<service>/roles.sql` needs to be selected.

Then, using **where=/** or **where=**, the Plugin will load this SQL file to the database. If some roles already exist, errors will be printed to the Job log.

Note: It is possible to restore the `roles.sql` file to a local directory, edit it, and load it using `psql` to restore only a selection of its original contents.



```
Directories
/
  @PG/
    main/
      akonadi/
      bacula/
      bacula1/
      git/
      master/
      postgres/
      regress/
      regress1/
      regress2/
      regress3/
      slave1/
      slave2/
      slave3/
      slave4/
```

```
Mark File             Mode       User     Group    Size Date
     akonadi/         drwx------ postgres postgres 4096 2012
     bacula/          drwx------ postgres postgres 4096 2012
     bacula1/         drwx------ postgres postgres 4096 2012
     git/             drwx------ postgres postgres 4096 2012
     master/          drwx------ postgres postgres 4096 2012
     postgres/        drwx------ postgres postgres 4096 2012
     regress/         drwx------ postgres postgres 4096 2012
     regress1/        drwx------ postgres postgres 4096 2012
     regress2/        drwx------ postgres postgres 4096 2012
     regress3/        drwx------ postgres postgres 4096 2012
     roles.sql        -rw-r----- postgres postgres   -1 2012
     slave1/          drwx------ postgres postgres 4096 2012
     slave2/          drwx------ postgres postgres 4096 2012
     slave3/          drwx------ postgres postgres 4096 2012
     slave4/          drwx------ postgres postgres 4096 2012
     slave5/          drwx------ postgres postgres 4096 2012
     tablespaces.sql  -rw-r----- postgres postgres   -1 2012
```

Fig. 2: PostgreSQL Cluster Contents During Restore

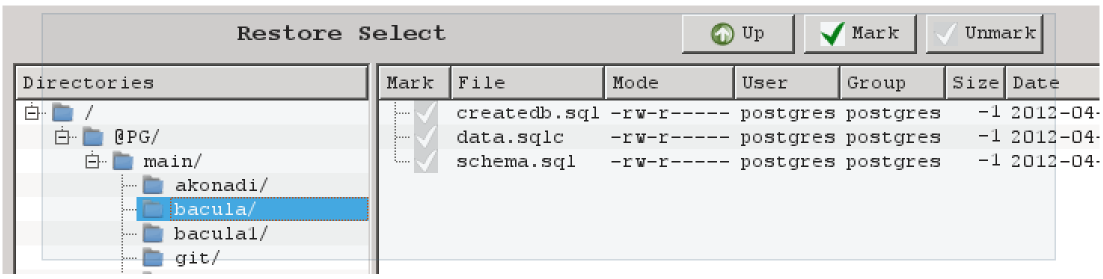
Restoring Database Structure

To restore only the database structure using the Bacula Enterprise Postgresql Plugin, the file `createdb.sql` located in the database directory needs to be selected during the restore process. To recreate the SQL database schema, the `schema.sql` file is used which contains all commands needed to recreate the database schema. The `schema.sql` file must be restored to disk and loaded manually into the database using the `psql` command.

Restoring Single Database

To restore a single database with the Bacula Enterprise Postgresql Plugin, the appropriate files from the database directory are selected during the restore process.

To restore the database with its original name, the selection should only contain the data file (`data.sqlc` or `data.sql`). If the `createdb.sql` file is also selected, harmless messages might be printed during the restore.



```
Directories
/
  @PG/
    main/
      akonadi/
      bacula/
      bacula1/
      git/
```

```
Mark File          Mode       User     Group    Size Date
     createdb.sql  -rw-r----- postgres postgres   -1 2012-04-
     data.sqlc     -rw-r----- postgres postgres   -1 2012-04-
     schema.sql    -rw-r----- postgres postgres   -1 2012-04-
```

Fig. 3: Database Contents During Restore

To restore a single database to a new name, the two files `createdb.sql` and `data.sqlc` (or `data.sql`) must be selected. The **where** parameter is used to specify the new database name. If **where** is set to a single word consisting of only `a..z`, `0-9` and `_`, Bacula will create the specified database and restore the data into it.

```
* restore where=baculaold
...
cwd is: /
$ cd /@PG/main/bacula
cwd is: /@PG/main/bacula/
$ m data.sqlc
$ m createdb.sql
$ ls
schema.sql
*data.sqlc
*createdb.sql
```

If the restore process has an error such as `ERROR: database "xxx" already exists`, the `createdb.sql` can be skipped in the restore selection.

If the **replace** parameter is set to **never**, Bacula will check the database list, and will abort the Job if the database currently restored already exists.

Using **replace=always** is not recommended.

If the **where** parameter is a directory (containing `/`), Bacula will restore all files into this directory. Doing so, it is possible to use `pg_restore` directly and restore only particular contents, such as triggers, tables, indexes, etc.

Note: Some databases such as `template1`, `postgresql` or databases with active users can not be replaced.

Restoring Dump Files to Directory

To restore SQL dumps to a directory, the **where** parameter needs to be set to indicate an existing directory.

```
* restore where=/tmp
```

Restoring Single Table

To restore a single item such as a table, it is currently needed to restore the dump file to a directory and use the `pg_restore` command.

Restoring Complete Cluster Using PITR

Useful information for this disaster recovery scenario can be found in the PostgreSQL manual, for example at: http://www.postgresql.org/docs/9.1/static/continuous-archiving.html

The overall process is as follows:

1. Stop the server, if it's running.
2. If the space to do so is available, the whole Cluster data directory and any tablespaces should be copied to a temporary location in case they are needed later.

   It is very important to use the "-a" parameter to copy the files, so that the files retain their ownership

```
root@dc-u24Postgres16db-test:/var/lib/postgresql/16/main# cp -a pg_wal/*␣
↪/tmp/pg_wal.backup/
root@dc-u24Postgres16db-test:/var/lib/postgresql/16/main# ls -lhtr /tmp/
↪pg_wal.backup/
total 97M
-rw------- 1 postgres postgres  16M Jul  2 17:18 000000010000000000000001
-rw------- 1 postgres postgres  16M Jul  2 17:18 000000010000000000000002
-rw------- 1 postgres postgres  349 Jul  2 17:18␣
↪000000010000000000000002.00000028.backup
-rw------- 1 postgres postgres  16M Jul  2 17:26 000000010000000000000003
-rw------- 1 postgres postgres  16M Jul  2 17:26 000000010000000000000004
-rw------- 1 postgres postgres  370 Jul  2 17:26␣
↪000000010000000000000004.00000028.backup
-rw------- 1 postgres postgres  16M Jul  2 17:36 000000010000000000000005
drwx------ 2 postgres postgres 4.0K Jul  2 17:36 archive_status
-rw------- 1 postgres postgres  16M Jul  2 17:36 000000010000000000000006
root@dc-u24Postgres16db-test:/var/lib/postgresql/16/main#
```

Note: This precaution will require having enough free space to hold two copies of the existing databases. If enough space is not available, at least the contents of the `pg_wal` subdirectory of the Cluster data directory should be copied, as it may contain logs which were not archived before the system went down.

3. Clean out all existing files and subdirectories below the Cluster data directory and the root directories of any tablespaces being used.
4. Restore the database files from the backups. If tablespaces are used, it is strongly recommended to verify that the symbolic links in `pg_tblspc/` were correctly restored. The **PrefixLinks** restore Job option can be useful here.
5. Any files present in `pg_wal` can be removed; these came from the backup and are therefore probably obsolete rather than current. Normally, this directory should be empty after a restore.
6. If there are unarchived WAL segment files that were saved in the step 2, they need to be copied back into `pg_wal/` (it is best to copy, not move them, so that the unmodified ones are available if a problem occurs and the process needs to be done again).
7. The recovery command file `recovery.conf.sample` inside the Cluster data directory may need to be edited and renamed to `postgresql.recovery.conf`. It may be useful to temporarily modify `pg_hba.conf` to prevent ordinary users from connecting until the recovery has been verified.
8. Start the server. The server will go into recovery mode and proceed to read through the archived WAL files it needs. Should the recovery be terminated because of an external error, the server can simply be restarted and it will continue recovery. Upon completion of the recovery process, the server will rename `recovery.conf` to `recovery.done` (to prevent accidentally re-entering recovery mode in case of a later crash) and then commence normal database operations.

```
# su postgres
$ cd /path/to/the/data/directory
$ mv recovery.conf.sample recovery.conf
$ vi recovery.conf
$ pg_ctl -D $PWD start
```

9. The contents of the databases restored should be verified to ensure it was recovered to the desired

state. If not, return to step 1.

10. If all is well, users can be allowed to connect by restoring `pg_hba.conf` to its normal contents.

Warning: About tablespaces and symlinks:

When applying logs, PostgreSQL needs to create the tablespace directory to re-create the tablespace, and PostgreSQL doesn't support the relocation. So, when replaying logs, it will overwrite or fail on this operation.

7 Limitations

- Postgres developers didn't implement the backup routines with data deduplication in mind, therefore the results may not be ideal. However the `cluster` command may be used at times to enhance the deduplication ratio as it physically reorders the data according to the index information. However, notice that this command requires exclusive lock while it is running, and also may quite heavy on CPU and I/O resources.
- In PITR mode, each Job should contain a single Cluster. To backup multiple clusters installed on the same client, multiple jobs are needed.
- In Dump mode, PostgreSQL version 8.3 and earlier may not support automatic restore through the Bacula Enterprise PostgreSQL plugin. The `pg_restore` program can produce the following error: `WARNING: invalid creation date in header`. In this case, it is needed to restore data to a directory, and run `pg_restore` in a terminal.
- The **clean** option requires PostgreSQL 9.1 or later.
- The backup will include all files in the data directory and tablespaces, including the configuration files and any additional files placed in the directory by third parties, except certain temporary files managed by PostgreSQL. But only regular files and directories are copied, except that symbolic links used for tablespaces are preserved. Symbolic links pointing to certain directories known to PostgreSQL are copied as empty directories. Other symbolic links and special device files are skipped. See: https://www.postgresql.org/docs/current/protocol-replication.html for the precise details.
- The `restart` command has limitations with plugins, as it initiates the Job from scratch rather than continuing it. Bacula determines whether a Job is restarted or continued, but using the `restart` command will result in a new Job.