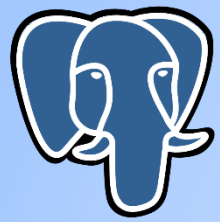
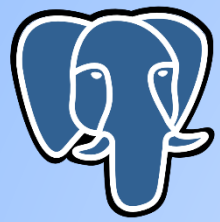


PostgreSQL

Module 13 (Part 1)

Backup and Recovery



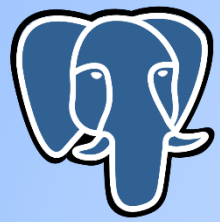
Module Overview

- Understanding and controlling crash recovery
- Planning backups
- Hot logical backup of one database
- Hot logical backup of all databases
- Backup of database object definitions
- Standalone hot physical database backup
- Hot physical backup and continuous archiving
- Recovery of all databases
- Recovery to a point in time

PostgreSQL

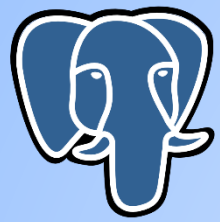
Terminology

- Few terms which are very important for designing your backup-recovery and disaster-recovery processes
  - Write-Ahead Log (WAL): PostgreSQL writes information to a series of write-ahead log files, in segments 16 MB in size, before making corresponding changes to the database itself
  - Archiving: Means archiving your transaction/xlog/wal segments/WAL
  - Backup: A process in which you take a backup of your database. It can be logical where you take a backup of tables and rows or it can be physical where you backup the datafiles and other configuration files needed by your database/instance/cluster
  - Crash Recovery: You had a crash of your database service/server and your database is recovering from the same. Can be achieved with your WAL segments/WAL itself
  - Point-in-Time Recovery (PITR): If you have a base database and a series of WAL files, you can apply only some of WAL files and then stop recovering information from those WAL files.



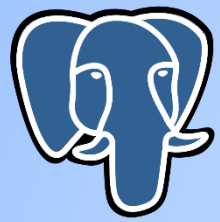
Understanding And Controlling Crash Recovery

- Crash recovery is the PostgreSQL subsystem that saves us, should the server crash or fail as part of a system crash.
- PostgreSQL will immediately restart and attempt to recover using the transaction log or **Write- Ahead Log** (**WAL**).
- The WAL consists of a series of files written to the pg_xlog subdirectory of the PostgreSQL data directory.
- Crash recovery replays the WAL, Recovery starts from points in the WAL known as **checkpoints**.
- The duration of a crash recovery depends on the number of changes in the transaction log since the last checkpoint
- A checkpoint can be either immediate or scheduled
  - CHECKPOINT command



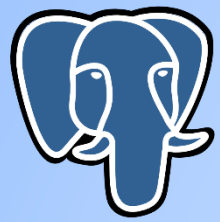
Understanding And Controlling Crash Recovery

- Two parameters control the occurrence of scheduled checkpoints
  - checkpoint_timeout : time until the next checkpoint
    - Default: 5minute
  - max_wal_size: amount of WAL data that will be written before checkpoint
    - Default: max_wal_size is set to 1 GB
- wal_keep_segments: specifies the number of 16 MB WAL files to be retained in the pg_xlog directory
  - 16 MB x wal_keep_segments of space
- Recovery continues until the end of the transaction log
- There is no defined end point, recovery always ends with some kind of error - "**the next record does not exist (yet).**"



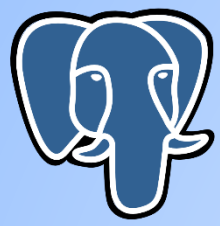
Planning Backups

- The type of backup you take influences the type of recovery that is possible.
- Consider the following main aspects:
  - Full or partial database?
  - Everything or just object definitions only?
- Main backup options are the following:
  - Logical backup, using pg_dump
  - Physical backup, which is a filesystem backup
- The pg_dump utility comes in two main flavors: pg_dump and pg_dumpall. pg_dump has a -F option for producing backups in various file formats
- file system backup
  using pg_start_backup() and pg_stop_backup()

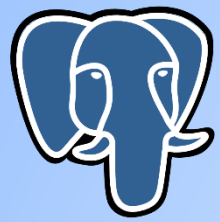


Hot Logical Backup Of One Database

- Logical backup makes a copy of the data in the database by dumping the content of each table.
- The command to do this is simple, as follows:
  - **pg_dump –F c –f dumpfile**
  - **Or pg_dump -F c > dumpfile**
- The pg_dump utility produces a single output file.
- The pg_dump archive file, also known as **custom format**, is lightly compressed by default.
- If you are making a script dump, you can do a dump verbose, as follows:
  - **pg_dump –v**
- Note that pg_dump does not dump roles (such as users and groups). Those two are only dumped by pg_dumpall.

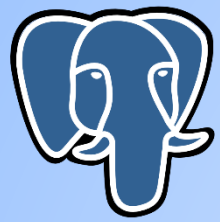


- Pg_dump—can take the backup of specific table, database, and schema in specific format like t (tar), p (plain text), d (directory).
- Command to backup database:
  - pg_dump -d databasename -f file path –F format
  - pg_dump –d db2 –f /var/lib/pgsql/9.3/backups/db2bak –F p
- Command to take backup of table:
  - pg_dump -d databasename –t tablename -f file path –F format
  - pg_dump –d db2 -t demo1 –f /var/lib/pgsql/9.3/backups/demo1bak –F t
- Command to take backup of schema:
  - Pg_dump -d databasename -n schemaname -f file path –F format
  - Pg_dump –d db2   -n dbo –f /var/lib/pgsql/9.3/backups/dboschbak –F d



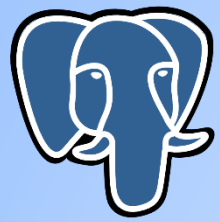
Hot Logical Backup Of All Databases

- To back up all databases, you may be told you need to use the **pg_dumpall** utility.
- Some drawbacks of pg_dumpall command:
  - If you use pg_dumpall, then the output produced is in a script file.
  - The dumps of individual databases are not consistent to a particular point in time.
  - The pg_dumpall utility produces dumps of each database one after another. This means that pg_dumpall is slower than running multiple pg_dump tasks in parallel,
  - Options for pg_dumpall are similar in many ways to pg_dump, though not all of them exist, so some things aren't possible.



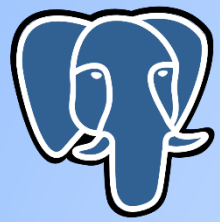
Hot Logical Backup Of All Databases

- pg_dumpall- can also take backup of all database in plain text format:
- Command to backup database:
  - pg_dumpall -f file path
  - pg_dumpall -f /var/lib/pgsql/9.3/backups/alldbbak



Backup Of Database Object Definitions

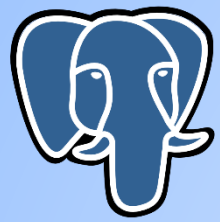
- It's useful to get a dump of the object definitions that make up a database.
- The basic command to dump the definitions only is the following:
  - **pg_dumpall --schema-only > myscriptdump.sql**
- If you want to dump PostgreSQL role definitions, you can use this command:
  - **pg_dumpall --roles-only > myroles.sql**



PostgreSQL

Online Backup

- Online backup is a way to get a backup under instance startup condition
- Execute pg_start_backup function for the instance, and back up all the files of the database cluster
  - When this function is executed, WAL offset value at the time of backup start appears
  - The label file "{PGDATA}/backup_label" is also created
  - In the label file, start time and WAL of information backup are listed
  - postgres=# SELECT pg_start_backup(now()::text) ;
  - $ cat data/backup_label
- These operations can be done automatically by pg_basebackup command
- When the backup is complete, execute pg_stop_backup function
  - SELECT pg_stop_backup() ;

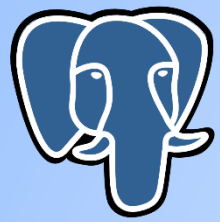


Hot physical backup

- The purpose of continuous archiving is to allow us to recover to any point in time from the time of the backup.
- The key point here is that we must have both the base backup and the archive in order to recover.
- If you compress WAL files regularly, the files produced by PostgreSQL 9.5 can be compressed better than those produced by earlier versions.

- Standalone hot physical backup:
  - cd /var/lib/pgsql/9.4/data/
  - mkdir –p ../../standalone – this command create a directory outside the data directory.
  - vi postgresql.conf
    - archive_mode = on
    - archive_command = ‘cp –i %p ../../standalone/archive/%f'
    - wal_level = replica
  - mkdir ../../standalone/archive – create a directory inside standalone directory
  - restart the postgres server: /usr/pgsql-9.4/bin/pg_ctl restart

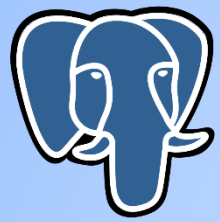
- psql
  - This command runs the checkpoint and create the backup label inside data directory
    - select pg_start_backup('backupname');
  - Use a tar command to compress all data directory to make a database base backup of all the files and sub directory of data folder except pg_xlog
    - tar -cv --exclude="pg_xlog/*" \-f ../../standalone/bak.tar /var/lib/pgsql/9.5/data
  - This command delete the backup label and stops the checkpoint
  - psql -c "select pg_stop_backup(), current_timestamp"
  - cp ../../standalone/archive archive/ -- Move the files to the archive subdirectory
  - tar -rf ../../standalone/bak.tar archive/ --copy archive directory to tar



- create a recovery.conf file:
  - echo “restore_command=’cp archive/%f %p’” > recovery.conf
  - echo “recovery_end_command=’rm –R archive’” >> r recovery.conf
  - tar –rf ../../standalone/bak.tar recovery.conf -- copy recovery.conf file to tar
  - Store the bak.tar somewhere safe is definitely not on the same server

Statistics overview

- The pg_stat_archiver catalogue with following fields:
  - archived_count: number of WAL files successfully archived
  - last_archived_wal: name of the last successfully archived WAL file
  - last_archived_time: timestamp of the last successfully archived WAL file
  - failed_count: number of failed WAL archiving attempts
  - last_failed_wal: WAL name of the last archiving failure
  - last_failed_time: timestamp of the last archiving failure
  - stats_reset : timestamp of the last reset of statistics
- Example :
  - postgres=# *SELECT * FROM pg_stat_archiver;*

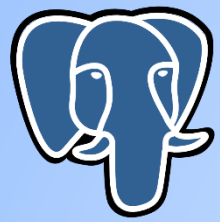


Point in time recovery

- PostgreSQL Backup Steps
  - 1. Modify postgresql.conf to support archive log
  - 2. Make a base backup (full database backup)
  - 3. Backup base backup to remote storage.
  - 4. Backup WAL (archive log files) to remote storage
- PostgreSQL Point-in-time Recovery Steps
  - 1. Extract files from base backup
  - 2. Copy files from pg_xlog folder
  - 3. Create recovery.conf file
    - restore_command = 'cp /archive/%f %p'
    - recovery_target_time = '2019-04-05 15:43:12’
    - SELECT pg_create_restore_point('my_daily_process_ended’);
      - recovery_target_name = 'my_daily_process_ended'
  - 4. Start Recover

Point in time recovery

- Steps:
  - 1. Database initialization
  - Initialize database
  - # su – postgres
  - -bash-$ /usr/pgsql-9.6/bin/initdb
  - Start the database
  - -bash-$ /usr/pgsql-9.6/bin/pg_ctl -D /var/lib/pgsql/9.6/data -l logfile start
  - 2. Make change in Postgresql configuration file (postgresql.conf)
  - archive_mode = on
  - archive_command = 'cp %p /var/lib/pgsql/9.6/wals/%f' //mkdir wals under /var/lib/pgsql/9.6/



Point in time recovery

- wal_level = replica
- Restart the database

- 3. Data Simulation & Backup Process
- -bash-$ psql
- # create table tab1 as select * from pg_class; //totally 228 records
- # select * from current_timestamp; // In my case – 2016-12-13 16:09:47.610651+05:30
- Check the log files under pg_xlog and wals directory

- 4. Create a full databse backup – base backup
- # select pg_start_backup('Full Backup');

Point in time recovery

- Use a tar command to compress all data directory to make a database base backup
- -bash-$ cd /var/lib/pgsql/9.6/
- -bash-$ tar -cvzf backups/databk.tar data
- pgdatabkup.tar this is the full database backup (base backup) including Postgresql configuration , system and all others files and folder.
- # select pg_stop_backup();

- 5. Create tables
- # create table tab2 as select * from pg_class;
- # select current_timestamp; //In my case, it is 2016-12-13 16:13:18.762008+05:30
- Check the tables with \d

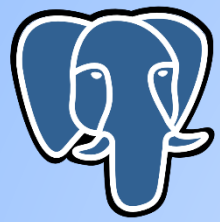
Point in time recovery

- 6. We have to do something in order to make our PostgreSQL server go down.
- -bash-$ /usr/pgsql-9.6/bin/pg_ctl -D /var/lib/pgsql/9.6/data stop // stop the server
- or
- -bash-$ kill -9 $(head -1 /var/lib/pgsql/9.6/data/postmaster.pid) // kill the postmaster

- 7.Recovery Process
- Rename data to data.bad.data, assume database file in data folder was damaged
- -bash-$ mv data data.bad.data

Point in time recovery

- 8. Unzip / extract files from databk.tar. It will extract the data directory to this current location
- -bash-$ tar -xvzf backups/databk.tar
- -bash-$ cd data/
- -bash-$ rm -rf pg_xlog/*
- Compare wals folder files and data.bad.data/pg_xlog files. Copy log files from pg_xlog folder. Some log files still located in data.bad.data pg_xlog folder (those log files hanv't archive yet) during disaster happening, we need to copy the log file back and recover it as much as possible.
- And copy files from data.bad.data/pg_xlog to /var/lib/pgsql/9.6/data/pg_xlog, like below
- -bash-$ cp ../ data.bad.data/pg_xlog/000000010000000000000004 /var/lib/pgsql/9.6/data/pg_xlog/
- -bash-$ cp ../ data.bad.data/pg_xlog/000000010000000000000005 /var/lib/pgsql/9.6/data/pg_xlog/

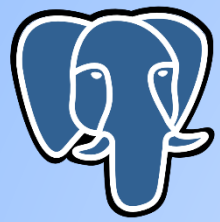


Point in time recovery

- 9. Create a recovery.conf file and put it under /var/lib/pgsql/9.6/data
- restore_command = 'cp /var/lib/pgsql/9.6/wals/%f %p'
- recovery_target_time = '2010-06-01 16:59:14.27452+01'

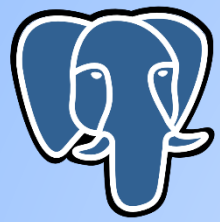
- 10. Give the permission and Start the database
- -bash-$ chown -R postgres.postgres /var/lib/pgsql/9.6/data/pg_xlog/
- -bash-$ chown -R postgres.postgres recovery.conf

- 12. Start database and output log file to /var/lib/pgsql/9.6/data/pg.log
- -bash-$ /usr/pgsql-9.6/bin/pg_ctl start -D /var/lib/pgsql/9.6/data/ -l logfile
- -bash-$ psql
- postgres=# \d



Recovery of the database

- Psql command to restore plain text backup file.
- Command to restore database:
  - Create database “newdb” first
  - psql –d databasename –U username –f filename
  - psql –d newdb –U postgres -f /var/lib/pgsql/9.3/db2bak
- Command to restore table:
  - pg_restore -d databasename   filename
  - pg_restore -d adventureworks /var/lib/pgsql/9.3/t1tablebak
- Command to restore schema :
  - pg_restore -d databasename filename
  - pg_restore -d postgres /var/lib/pgsql/9.3/dboschemabak



Conclusion

- Most people admit that backups are essential, though they also devote a very small amount of time to thinking about the topic.
- Understanding and controlling crash recovery. You need to understand what happens if the database server crashes so that you can understand when you might need to recover.