Doing PITR Right

PGConf.EU 2013
Dublin, Ireland

Stephen Frost
sfrost@snowman.net

Stephen Frost

- PostgreSQL
  - Major Contributor, Committer
  - Implemented Roles in 8.3
  - Column-Level Privileges in 8.4
  - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
  - Principal Database Engineer
  - Online Digital Media Company
  - We're Hiring! - techjobs@resonateinsights.com

Do you read...

- planet.postgresql.org

What is PITR?

- Backup Strategy using PG's Write-Ahead-Log (WAL)
  - All changes written to WAL first
  - WAL is used for crash recovery
- PITR requires
  - Full backup
  - WAL files since last full backup
- Full backup can be done while DB is online
- (Configuration may require DB restart)

Why PITR?

- What about pg_dump?
  - Single-threaded (well, it was..)
  - Not practical for large-scale databases
  - Keeps a very long running transaction open..
- Restore can be parallel, but still very slow
  - Data has to be re-parsed
  - Indexes must be rebuilt
- But we have replication!
  - Drop a table on the master?
  - Corrupted / bad data?

Getting Ready for PITR

- Configure PG for archiving *first*!
  - (and check that it's working!)
  - Needs to be done before taking a full backup
- postgresql.conf
  - wal_level - hot_standby (or archive..)
  - archive_mode - on
  - archive_command
- May change performance coming from minimal
- No real reason to use archive.. use hot_standby

archive_command

- Simple - NEVER overwrite files, check for them first

```
test ! -f /mnt/server/archivedir/%f && \
cp %p /mnt/server/archivedir/%f
```

- Always return true (0) only on success
- Non-zero will cause PG to retry
- Advanced - Test, test, test! Verify return codes.

```
/path/to/my_script.sh %p %f
```

- Monitor archiving, disk space, etc!
- Do not allow partial copy; will cause later failure.
- If PG can't write WAL (no space)- it will STOP.

Backing up PG

- Before copying files, run:
  - psql -c "select pg_start_backup('mylabel',true);"
  - 'mylabel' can be anything
  - Second argument defines checkpoint behavior
    - "true" forces immediate / fast
    - "false" allows "lazy" / spread out
- backup_label file
  - Stores the label used in pg_start_backup
  - Includes starting WAL file, etc.
  - Removed by pg_stop_backup()

Backing up PG

- Copy all files in the PG 'data' directory
  - Use rsync or tar
  - Be sure to include all tablespace directories!
  - (tablespaces are symlink'd out of pg_tblspc)
  - Config files, PG log files, etc.
  - Exclude pg_xlog, postmaster.pid, postmaster.opts
- When done, run:
  - psql -c "select pg_stop_backup();"
  - Forces a final WAL switch

pg_basebackup

- Makes backing up WAY easier / simpler
- Configure PG for archiving *first*!
  - (and check that it's working!)
- Uses the PG replication protocol
  - Needs max_wal_senders set >0
  - Streams data files through PG port
- Set up replication user in pg_hba.conf
- Do NOT use regular superuser

pg_basebackup Options

- -D - directory for output files
  - Tablespaces go to where they are on the master
- -F - format (plain or tar)
  - Can't also stream XLOG (yet..)
- -X - XLOG include method (fetch or stream)
- -l - Label to use like in pg_start_backup
- -z - compression
- -c - checkpoint lazy / fast
- -P - Cute progress info
- Remember to address config files, log files, etc.

pg_receivexlog

- Used to stream just XLOG files
- Independent of pg_basebackup
- Uses PG replication protocol also
- Continuous streaming- no archive_timeout needed
- Options
  - -D - Directory to dump XLOG files to
- Still need archive_command
  - Check that WAL archived
  - sleep 5 && test -f /mnt/server/archivedir/%f
  - Prevents recycling before XLOG archived

WAL-e

- Heroku tool to push PG backups to S3
- http://github.com/wal-e/wal-e
- Includes
  - Compression
  - Encryption
  - Full base backups && WAL
  - Restores base backup w/ WAL
- Primary backup method of Heroku
- http://heroku.com

Restoring!

- Test your backups!
- By actually doing a *restore*!
- Test regularly! (At least once a year..)
- Consider multiple scenarios
  - Restore from off-line storage (tape, etc)
  - Pull backup from off-site location
  - Fail-over from 2nd / redundant site
  - (and actually restore from a backup)

Restoring with PITR

- Restore full backup first
- Ideally to another location / server
- pg_xlog should be empty or non-existant (create it)
- Verify tablespace symlinks and files
- If the old system exists still
  - Copy pg_xlog files from old system to new
  - (May allow restore beyond last archived WAL)

resonate

recovery.conf

- Create a recovery.conf in data directory
- restore_command - similar to archive_command
  - Retrives archived WAL
  - %f - Filename/XLOG to be restored
  - %p - Location to restore file to
  - Return zero on success
  - Less than 126 for 'normal' error
  - 126 or above for 'fatal' error

Recovery Target

- recovery_target_(name|time|xid|inclusive|timeline)
  - name - pg_create_restore_point()
  - time - Timestamp to recover up until
  - xid - Specific XID, up-to-and-including
  - inclusive (of time or XID)
  - timeline - Specify timeline to restore into
- recovery_end_command
  - Command to run upon completion of restore
  - Can perform clean-up, etc

Simple recovery.conf

- recovery.conf
  - restore_command = 'cp /mnt/server/archivedir/%f "%p"'
  - recovery_target_time = '2013-10-31 10:00'
  - pause_at_recovery_target = false
- Recovers up to specified time (including that time)
- Immediately moves into 'on-line' mode

Advanced PITR restore

- recovery.conf
  - restore_command = '/path/to/myscript %f %p'
  - recovery_target_xid = 1234
  - pause_at_recovery_target = true
- Need to log XIDs
  - Not all transactions get real XIDs
  - Virtual XIDs can not be used
- Pauses recovery until pg_xlog_replay_resume()
  - Needs to have hot_standby enabled
  - Must have specific recovery_target set

Thank you!

Stephen Frost
sfrost@snowman.net
@net_snow