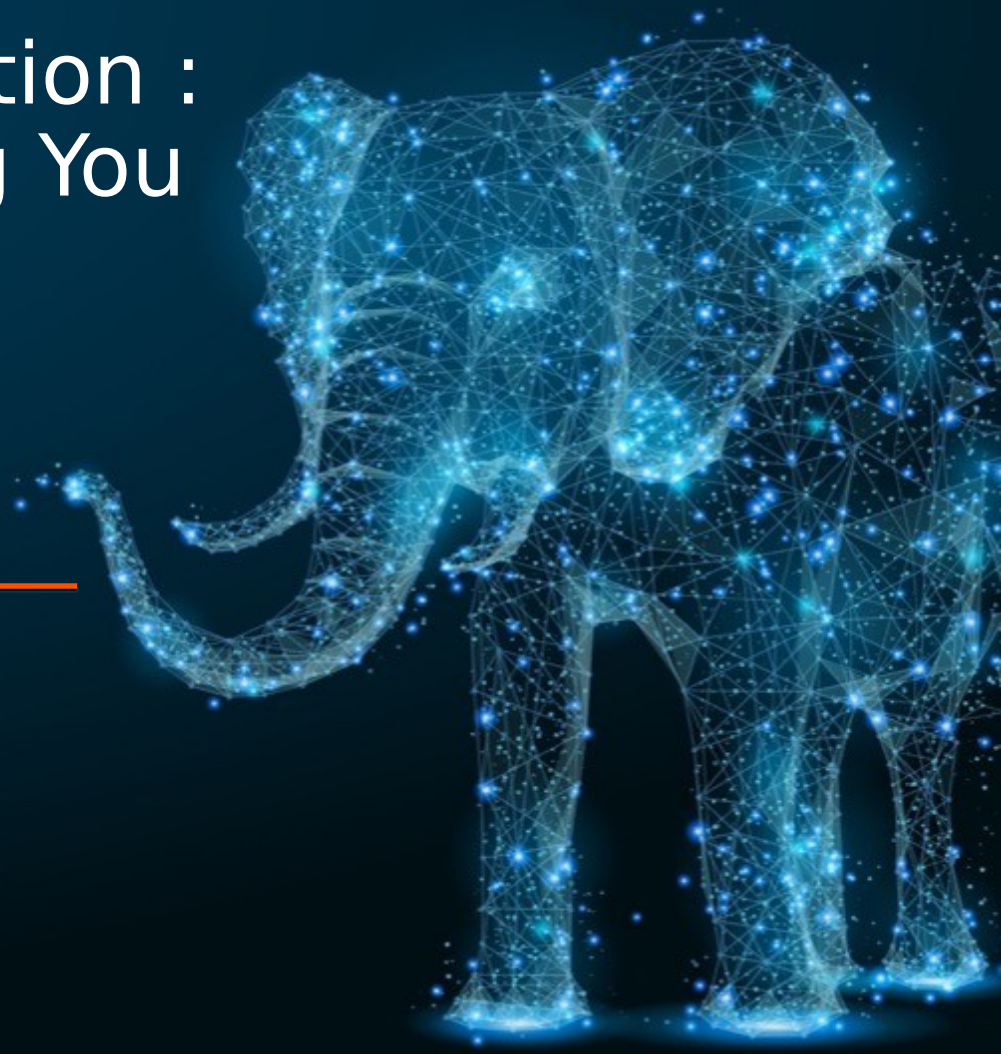
PostgreSQL Replication : (Almost) Everything You Want To Know



Devrim Gündüz

devrim@gunduz.org

DevrimGunduz

DevrimGunduzTR

SELF INTRODUCTION

- Using Red Hat (and then Fedora) since 1996.
- Using PostgreSQL since 1998.
- Started building RPMs in 2002, took over the project in 2004.
- Planet PostgreSQL: 2004 → https://planet.PostgreSQL.org
- PostgreSQL Major Contributor: Responsible for PostgreSQL YUM and ZYPP repositories.
- Working at EnterpriseDB since 2011
- Living in London, UK.
- **The** Guy With **The** PostgreSQL Tattoo!

SOCIAL MEDIA

Please tweet:

#PostgreSQL

Please follow:

@PostgreSQL

History of replication in PostgreSQL

Time travel

HISTORY OF REPLICATON IN POSTGRESQL

- Trigger based:
  - Slony
  - Bucardo
  - Londiste

HISTORY OF REPLICATION IN POSTGRESQL

- 8.2: Warm standby
- 9.0: Initial release of in-core Streaming replication (Physical replication)
- 9.1: Synchronous replication, pg_basebackup
- 9.2: Cascading replication
- 9.3: Follow timeline switch
- 9.4: Logical decoding, replication slots
- 9.6: Multiple sync replication, quorum, remote_apply

HISTORY OF REPLICATON IN POSTGRESQL

- 10: Initial release of in-core logical replication
  - pg_basebackup: stream by default
  - Replication-ready by default
- 11: Logical replication supports TRUNCATE
- 12: Removal of recovery.conf!

HISTORY OF REPLICATON IN POSTGRESQL

- 13: pg_stat_progress_pgbasebackup, logical replication of partitioned tables, online changes of primary_conninfo and primary_slot_name
- 14: online change of restore_command, streaming of long transactions with logical replication,
- 15: logical replication: 2PC, row/column filtering, improved statistics, skipping transactions

HISTORY OF REPLICATON IN POSTGRESQL

- 16 (not-released-yet-for-production)
  - Removal of promote_trigger_file
  - Logical replication: Parallel apply

It all starts with WAL

Some basics

WHAT IS WAL?

- Write Ahead Log:

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),
  - 16 MB in most of the installations (can be configured, --with-wal-segsize)

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),
  - 16 MB in most of the installations (can be configured, --with-wal-segsize)
  - initdb has a --wal-segsize parameter

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),
  - 16 MB in most of the installations (can be configured, --with-wal-segsize)
  - initdb has a --wal-segsize parameter
  - initdb --wal-segsize=64 ← in MB

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),
  - 16 MB in most of the installations (can be configured, --with-wal-segsize)
  - initdb has a --wal-segsize parameter
  - initdb --wal-segsize=64 ← in MB
  - 8 kB page size (can be configured, --with-wal-blocksize during configure)

WHAT IS WAL?

- Write Ahead Log:
  - Logging of transactions
  - a.k.a. xlog in ancient times (transaction log),
  - 16 MB in most of the installations (can be configured, --with-wal-segsize)
  - initdb has a --wal-segsize parameter
  - initdb --wal-segsize=64 ← in MB
  - 8 kB page size (can be configured, --with-wal-blocksize during configure)
  - pg_resetwal –wal-segsize=64 ← in MB

WHAT IS LSN?

- Log Sequence Number

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit) (We'll need this info soon)

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit) (We'll need this info soon)
- Per docs: "Pointer to a location in WAL file"

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit) (We'll need this info soon)
- Per docs: "Pointer to a location in WAL file"
- LSN: Block ID + Segment ID (See next slides)

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit) (We'll need this info soon)
- Per docs: "Pointer to a location in WAL file"
- LSN: Block ID + Segment ID (See next slides)
- During recovery, LSN on the page and LSN in the WAL file are compared.

WHAT IS LSN?

- Log Sequence Number
  - Position of the record in WAL file.
  - Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit) (We'll need this info soon)
- Per docs: "Pointer to a location in WAL file"
- LSN: Block ID + Segment ID (See next slides)
- During recovery, LSN on the page and LSN in the WAL file are compared.
- The larger one wins.

WAL FILE NAMING

- 24 chars, hex.

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb
  - 2nd 8 chars: Block ID

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb
  - 2nd 8 chars: Block ID
  - 3rd 8 chars: Segment ID

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb
  - 2nd 8 chars: Block ID
  - 3rd 8 chars: Segment ID
- 000000010000000000000001 → 000000010000000000000002

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb
  - 2nd 8 chars: Block ID
  - 3rd 8 chars: Segment ID

- 000000010000000000000001 → 000000010000000000000002
- … 00000001000000000000000FF → 000000010000000100000000

WAL FILE NAMING

- 24 chars, hex.
  - 1st 8 chars: timelineID
    - 00000001 is the timelineID created by initdb
  - 2nd 8 chars: Block ID
  - 3rd 8 chars: Segment ID

- 000000010000000000000001 → 000000010000000000000002
- … 00000001000000000000FF → 000000010000000100000000
- ...and 00000001000000010000FF → 000000010000000200000000

EDB POSTGRES

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!
- Full page writes IYF

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!
- Full page writes IYF
  - Header data + entire page as a WAL record during the first change of each page after every checkpoint: Backup block / full page image

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!
- Full page writes IYF
  - Header data + entire page as a WAL record during the first change of each page after every checkpoint: Backup block / full page image
  - During replay, backup block overwrites data.

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!
- Full page writes IYF
  - Header data + entire page as a WAL record during the first change of each page after every checkpoint: Backup block / full page image
  - During replay, backup block overwrites data.
  - Enabled by default.

FULL PAGE WRITES

- A WAL record cannot be replayed on a page which is corrupted during bgwriter and/or checkpointer, because of hardware failure, OS crash, kernel failure, etc.
  - A failure can cause parts of old data still remain on the data page!
- Full page writes IYF
  - Header data + entire page as a WAL record during the first change of each page after every checkpoint: Backup block / full page image
  - During replay, backup block overwrites data.
  - Enabled by default.
- Please turn it off, if you want to throw a lot of money to PostgreSQL support companies. Otherwise, don’t do so ;)

FULL PAGE WRITES

- Increases WAL I/O

FULL PAGE WRITES

- Increases WAL I/O
  - PostgreSQL writes header data + the entire page as WAL record, when a page changes after **every** checkpoint.

FULL PAGE WRITES

- Increases WAL I/O
  - PostgreSQL writes header data + the entire page as WAL record, when a page changes after **every** checkpoint.
    - Increasing checkpoint_timeout and / or max_wal_size helps.

FULL PAGE WRITES

- Increases WAL I/O
  - PostgreSQL writes header data + the entire page as WAL record, when a page changes after **every** checkpoint.
    - Increasing checkpoint_timeout and / or max_wal_size helps.
      - Low values has a side effect: More WAL activity, per above.-

FULL PAGE WRITES

- Increases WAL I/O
  - PostgreSQL writes header data + the entire page as WAL record, when a page changes after **every** checkpoint.
    - Increasing checkpoint_timeout and / or max_wal_size helps.
      - Low values has a side effect: More WAL activity, per above.-
  - Full-page image, backup block.

FULL PAGE WRITES

- Increases WAL I/O
  - PostgreSQL writes header data + the entire page as WAL record, when a page changes after **every** checkpoint.
    - Increasing checkpoint_timeout and / or max_wal_size helps.
      - Low values has a side effect: More WAL activity, per above.-
  - Full-page image, backup block.
- PostgreSQL can even recover itself from write failures (not hw failures, though)

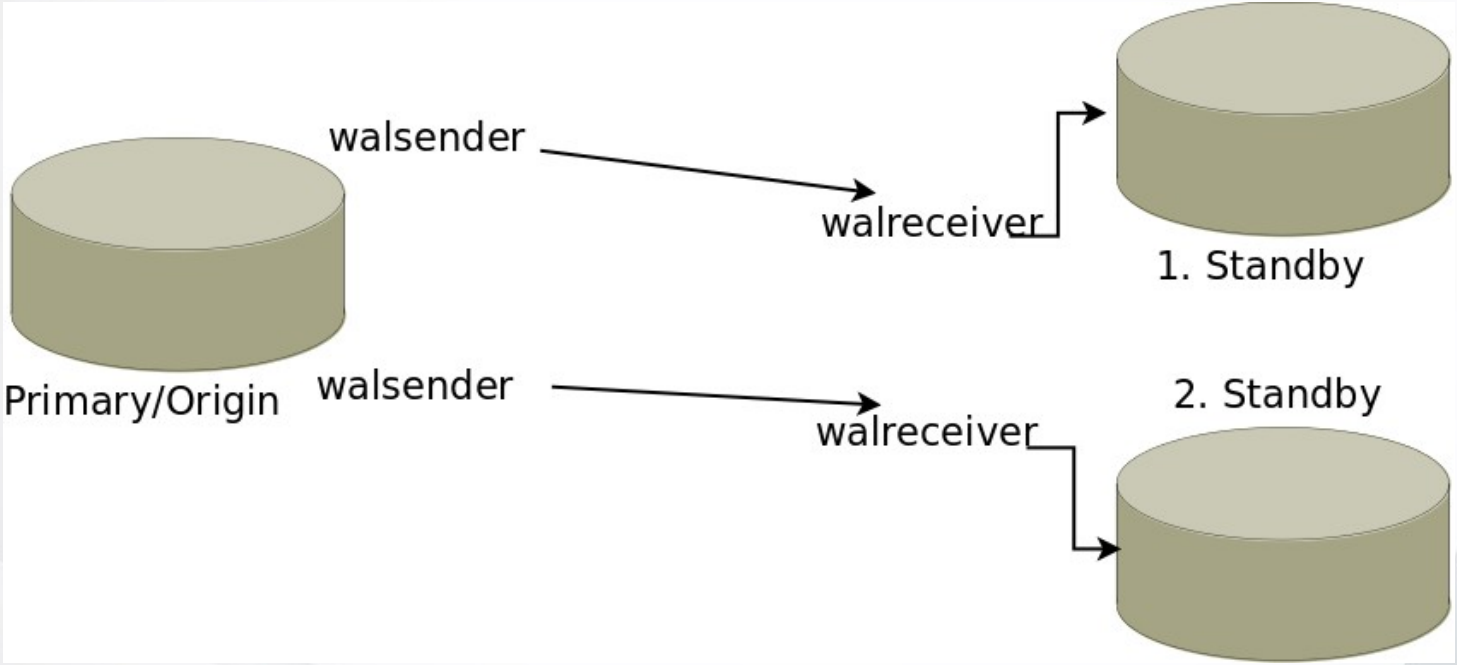
FULL PAGE WRITES

- Compression : pglz, lz4(v15+) and zstd(v15+)

Replication basics

Terminology

REPLICATION BASICS



REPLICATION BASICS

- primary, origin

REPLICATION BASICS

- primary, origin
- standby, subscriber

REPLICATION BASICS

- primary, origin
- standby, subscriber
- ~~master~~, ~~slave~~. Please.

REPLICATION BASICS

- Base backup

REPLICATION BASICS

- Base backup
- walsender

REPLICATION BASICS

- Base backup
- walsender
- walreceiver

REPLICATION BASICS

- Base backup
- walsender
- walreceiver
- Replication slot

REPLICATION BASICS

- Base backup
- walsender
- walreceiver
- Replication slot
- Logical decoding

Replication parameters

Primary server

REPLICATION PARAMETERS: PRIMARY

- synchronous_commit (on, off, local, remote_write, remote_apply)
- max_wal_senders
- wal_keep_size (v13+)
- synchronous_standby_names (FIRST, ANY)
- wal_level

Replication parameters

Standby server

REPLICATION PARAMETERS: STANDBY

- primary_conninfo
- primary_slot_name
- ~~promote_trigger_file~~
- hot_standby
- hot_standby_feedback
- recovery_min_apply_delay (time delayed standby)

Streaming replication

General features

STREAMING REPLICATION: GENERAL FEATURES

- Replication of whole cluster

STREAMING REPLICATION: GENERAL FEATURES

- Replication of whole cluster
- WAL-logged transactions are replicated

STREAMING REPLICATION: GENERAL FEATURES

- Replication of whole cluster
- WAL-logged transactions are replicated
- Works on the PostgreSQL port

STREAMING REPLICATION: GENERAL FEATURES

- Replication of whole cluster
- WAL-logged transactions are replicated
- Works on the PostgreSQL port
- No built-in auto failover/failback

STREAMING REPLICATION: GENERAL FEATURES

- Replication of whole cluster
- WAL-logged transactions are replicated
- Works on the PostgreSQL port
- No built-in auto failover/failback
  - Patroni, repmgr
  - Closed source solutions are also available

STREAMING REPLICATION: REPLICATION USER

- Separate user for replication
  - CREATE ROLE replicauser PASSWORD ‘foobar’ **REPLICATION** LOGIN;
  - pg_hba.conf

Streaming replication: Taking base backup

pg_basebackup

BASE BACKUP: BASICS

- **The** prerequisite for setting up streaming replication
- Should be on the same OS/patch level
- Physical backup of everything in the instance
- Taken from primary to standby(s)
- Also used for PITR/backup
- pg_hba.conf (on primary)

BASE BACKUP: PG_BASEBACKUP

- pg_basebackup
  - -D
  - -Fp (default)
  - -R
  - -X stream (default)
  - -c fast/spread (default)
  - -C -S slot_name
  - -P

BASE BACKUP: PG_BASEBACKUP

- -t (target) (v15+)
  - Default: client
  - server:/path/to/data (pg_write_server_files)
  - blackhole (testing only :))
- -p
- -h
- -U

BASE BACKUP: PG_BASEBACKUP

- Example:

  pg_basebackup -D /var/lib/pgsql/12/repdata -Fp -R -c fast -C -S blamemagnus -P -h 192.168.100.10 -p 5412 -U blamemagnus

Replication configuration: Standby server

REPLICATION CONFIGURATION: STANDBY

- recovery.conf < 12, postgresql.auto.conf and postgresql.conf >= 12
  - pg_basebackup -R
  - application_name in primary_conninfo (useful, and also needed for sync replication)

Logical replication

General features

LOGICAL REPLICATION: GENERAL FEATURES

- Not a replacement of streaming replication

LOGICAL REPLICATION: GENERAL FEATURES

- Not a replacement of streaming replication
- Different use cases

LOGICAL REPLICATION: GENERAL FEATURES

- Not a replacement of streaming replication
- Different use cases
- wal_level=logical

LOGICAL REPLICATION: GENERAL FEATURES

- Different features

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions
  - Single table/database replication

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions
  - Single table/database replication
  - Replication of set of tables

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions
  - Single table/database replication
  - Replication of set of tables
  - Replication of set of rows

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions
  - Single table/database replication
  - Replication of set of tables
  - Replication of set of rows
  - Replication of set of columns

LOGICAL REPLICATION: GENERAL FEATURES

- Different features
  - Replication between different major versions
  - Single table/database replication
  - Replication of set of tables
  - Replication of set of rows
  - Replication of set of columns
  - Writeable replica

LOGICAL REPLICATION: RESTRICTIONS

- Schema/DDL cannot be replicated

LOGICAL REPLICATION: RESTRICTIONS

- Schema/DDL cannot be replicated
- Large objects are not replicated

LOGICAL REPLICATION: RESTRICTIONS

- Schema/DDL cannot be replicated
- Large objects are not replicated
- Sequences are not replicated

LOGICAL REPLICATION: RESTRICTIONS

- Schema/DDL cannot be replicated
- Large objects are not replicated
- Sequences are not replicated
- Views, materialized views, foreign tables are not replicated

LOGICAL REPLICATION: EXAMPLES

- CREATE TABLE t1 (c1 int);
- CREATE PUBLICATION pgpub FOR TABLE t1;
- Alternatives:
  - CREATE PUBLICATION pgpub FOR TABLE t1,t2;
  - CREATE PUBLICATION pgpub FOR ALL TABLES;
  - CREATE PUBLICATION pgpub FOR TABLE t1

    WITH (publish = 'insert');

LOGICAL REPLICATION: EXAMPLES

- CREATE PUBLICATION pgpub FOR TABLE t1, TABLES IN SCHEMA schema1;
- CREATE PUBLICATION pgpub FOR TABLE t1 WHERE (c1 > 20);
- CREATE PUBLICATION pgpub FOR TABLE t1 (c1, c2);
- ALTER SUBSCRIPTION pgsub SKIP (lsn = 0/24D0216)

LOGICAL REPLICATION (SUBSCRIBER) : EXAMPLES

- Create tables on standby first. pg_dump --schema will help.
- CREATE SUBSCRIPTION pgsub CONNECTION 'dbname=postgres host=localhost user=replicauser port=5416' PUBLICATION pgpub;
- Initial data copy is done.

Some important points

TIPS

- Cascading replication?
- Issues on standby server
  - Replication delays
  - Network / hardware problems
- What happens when replica is dropped?

Replication monitoring

REPLICATION MONITORING: PRIMARY

- pg_stat_replication
- pg_replication_slots

REPLICATION MONITORING: STANDBY

- pg_stat_wal_receiver;
- pg_stat_recovery_prefetch (PostgreSQL 15+)
  - Depends on recovery_prefetch parameter in v15+
- pg_is_in_recovery()

PHOTO TIME

@CheerPostgreSQL

QUESTIONS & DISCUSSION

EDB POSTGRES

THANK YOU

Devrim Gündüz
devrim@gunduz.org
Twitter: @DevrimGunduz

EDB POSTGRES