Потоковая репликация на практике.

PgConf.Russia 2016, Moscow

Лесовский Алексей
lesovsky@pgco.me

PostgreSQL-Consulting.com

Содержание.

Материалы: http://goo.gl/Yy4UzH

I. Как устроена репликация.

II. Настройка репликации.

Практика.

III. Мониторинг и эксплуатация.

IV. Проблемы и решения.

V. Switchiver и Failover.

Практика.

Часть I. Как устроена репликация.

Что такое репликация и её разновидности.

Write Ahead Log. REDO и реализация REDO в PostgreSQL.

Общий взгляд на архитектуру репликации в PostgreSQL.

Особенности работы процессов отвечающих за репликацию.

Что такое репликация.

Копирование и синхронизация нескольких копий объекта.

Изменение одной копии распространяются на другие копии.

Репликация бывает физическая и логическая.

Логическая репликация.

Плюсы:

- Умеет работать между разными версиями и архитектурами.
- Умеет реплицировать отдельные таблицы и наборы таблиц.

Минусы:

- Крайне сложно реализовать синхронную репликацию.
- Дополнительная нагрузка на CPU (триггеры).

Примеры:

- Slony, Londiste (Skytools), Bucardo, Pglogical.

Физическая репликация.

Плюсы:

- Минимальный накладные расходы на использование ресурсов.
- Легкая настройка и сопровождение.

Минусы:

- Standby сервера доступны только на чтение.
- Не умеет работать между разными версиями и архитектурами.
- Нет возможности реплицировать отдельные таблицы и наборы таблиц.

Write Ahead Log. REDO.

Фиксирование всех изменений в базе (Durability в ACID).

При COMMIT делается синхронизация (flush) REDO буферов на диск.

Журнал REDO содержит историю всех изменений в БД.

Любое изменение в БД сначала фиксируется в REDO.

Журнал REDO используется:

- при аварийном восстановлении;
- в резервном копировании и Point In Time Recovery;
- в репликации.

Write Ahead Log. REDO implementation in PostgreSQL.

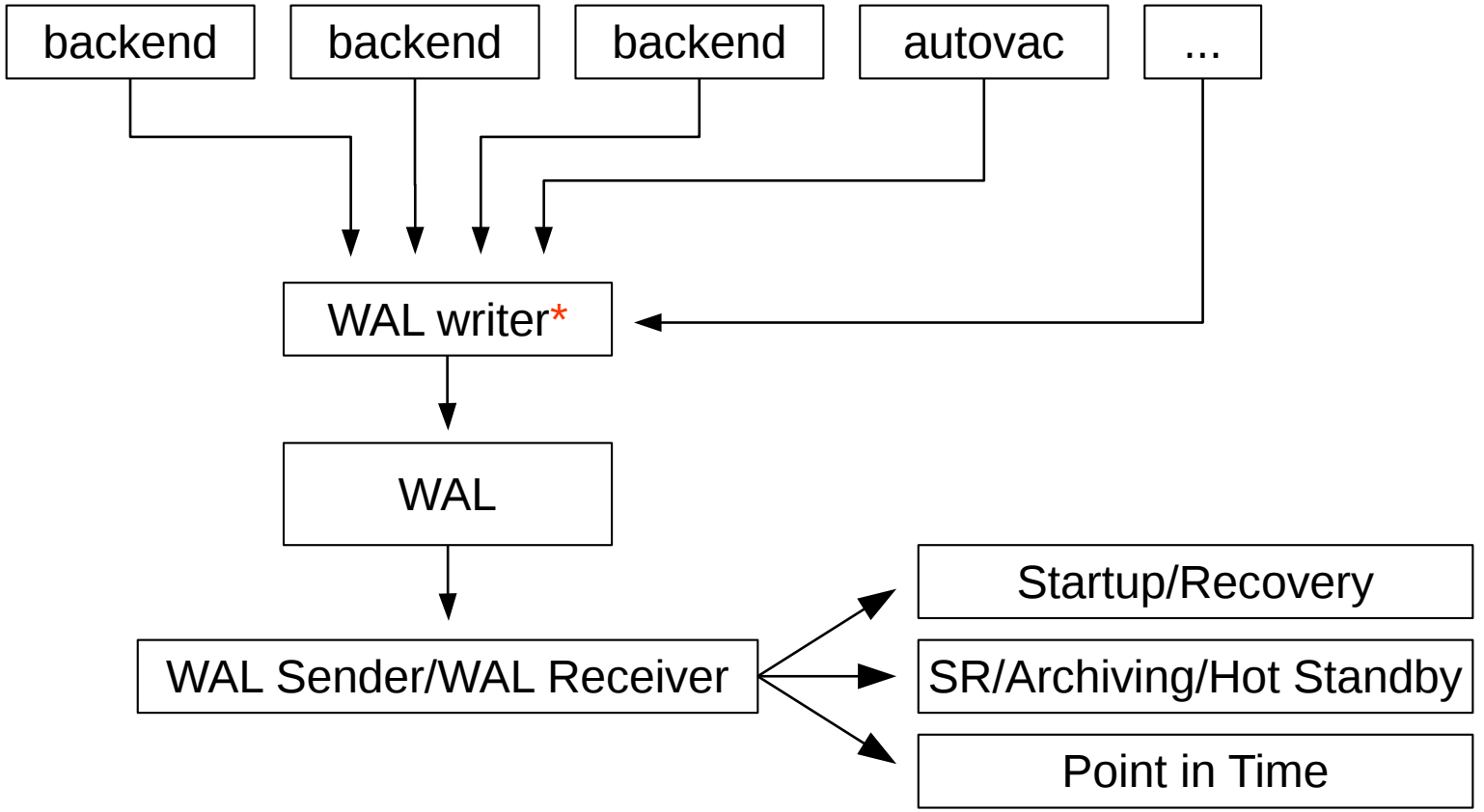
В PostgreSQL, REDO известен как Write Ahead Log (WAL).

WAL гарантирует запись изменений БД, до момента изменения самих данных.

Как это гарантировать?

- LSN (log sequence number) - местоположение записи внутри WAL журнала;
- каждая страница маркируется LSN последней записи журнала которая затрагивает страницу;
- bufmgr перед записью измененной страницы, должен убедиться что журнал сброшен на диск до LSN указанного в странице.

В общих чертах.



* - может отсутствовать

Startup process.

Задача startup процесса запустить БД (после чего он завершается).

В standby режиме он инициирует бесконечный replay loop.

При запуске REDO читаем recovery.conf.

REDO:

- чтение сегментов из pg_xlog/archive;
- запуск wal receiver и чтение XLOG с мастера;

При достижении consistency (min recovery ending location) разрешаем принимать подключения и запускаем checkpointer/bgwriter.

Следование оставшимся recovery параметрам из recovery.conf.

Еще больше конкретики смотрите в StartupXLOG() функции.

WAL Sender process.

Postmaster на каждого клиента создает процесс — backend.

WAL sender это тоже backend (с флагом am_walsender).

Бэкенд запускает exec_replication_command().

exec_replication_command() может делать разные вещи:

- создание/удаление слотов репликации;
- запуск basebackup;
- запуск физической/логической репликации.

В последнем случае, бэкенд отправляет XLOG сегменты клиенту.

Или спит когда новых XLOG нет.

WAL Receiver process.

Startup процесс поочередно проверяет источники XLOG.

Startup процесс инициирует запуск WAL receiver'a.

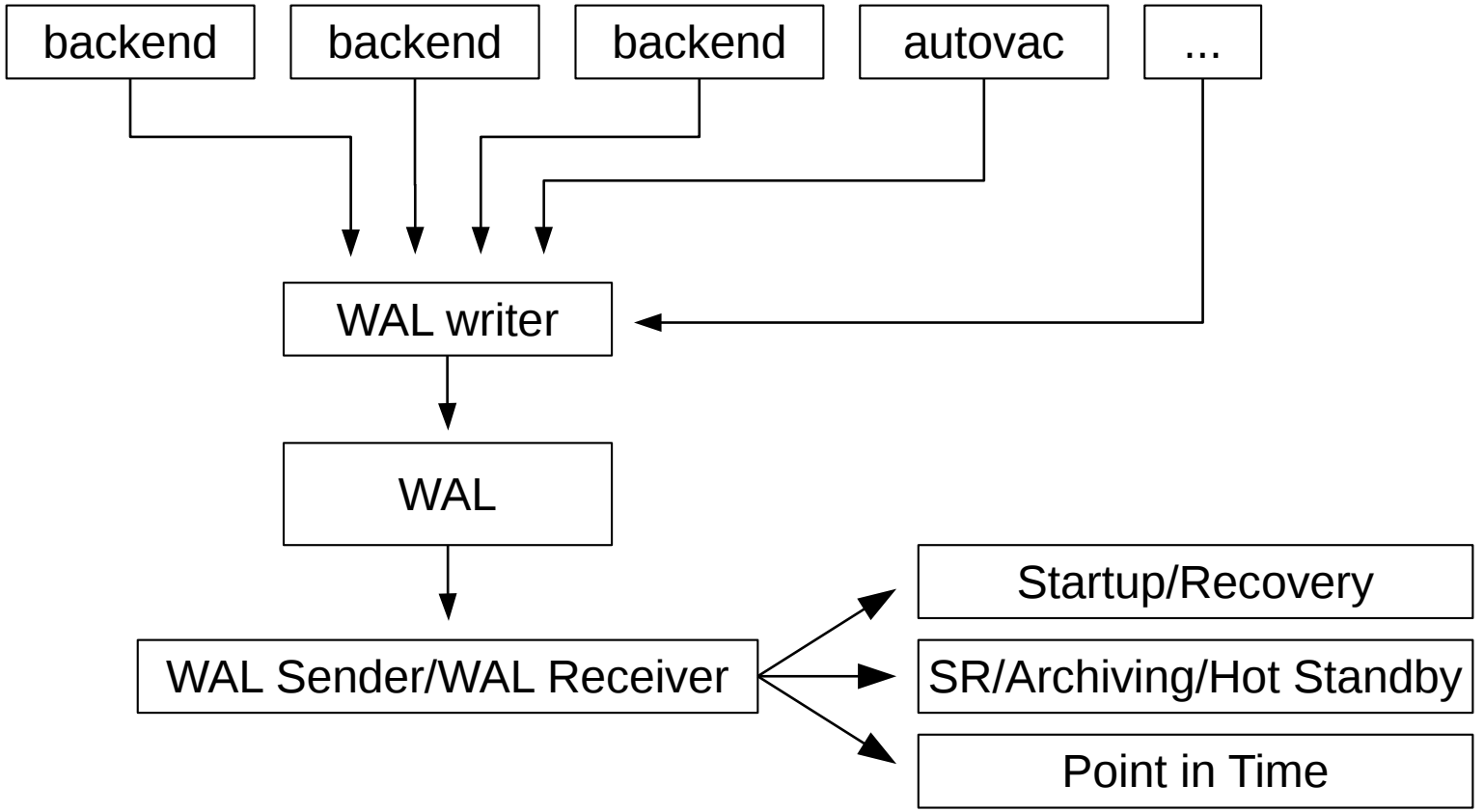
Нужен recovery.conf + primary_conninfo.

WAL receiver:

- определяет стартовую позицию для передачи XLOG;
- подключается к мастеру и передает стартовую позицию;
- принимает XLOG сегменты и записывает их на диск;
- обновляет переменную в shared memory (WalRcv→receivedUpto);
- отправляет статистику + фидбек.

Startup процесс использует переменную чтобы понимать до какой позиции следует воспроизводить XLOG.

Вопросы?


backend
backend
backend
autovac
...
WAL writer
WAL
WAL Sender/WAL Receiver
Startup/Recovery
SR/Archiving/Hot Standby
Point in Time

Часть II. Настройка репликации.

Варианты настройки.

Подготовка серверов к запуску репликации.

Используемые инструменты.

Запуск репликации и проверка успешности запуска.

Специфические особенности.

Варианты настройки.

Синхронная/Асинхронная репликация.

Каскадная репликация.

Uni-directional/Bi-directional.

Общий алгоритм.

Подготовка мастера.

Копирование DATADIR.

Подготовка standby.

Запуск standby.

Проверка результата.

Настройка мастера.

Отдельный пользователь для репликации.

Редактирование postgresql.conf.

Редактирование pg_hba.conf.

Создание слотов репликации (только при необходимости).

Настройка мастера.

Отдельный пользователь для репликации.

- CREATE ROLE ... WITH LOGIN REPLICATION PASSWORD '…';

Редактирование postgresql.conf.

Редактирование pg_hba.conf.

Создание слотов репликации (только при необходимости).

Настройка мастера.

Отдельный пользователь для репликации.

Редактирование postgresql.conf.

- wal_level = hot_standby
- max_wal_senders > 0
- Перезапуск PostgreSQL

Редактирование pg_hba.conf.

Создание слотов репликации (только при необходимости).

Настройка мастера.

Отдельный пользователь для репликации.

Редактирование postgresql.conf.

Редактирование pg_hba.conf.

- host replication username client_addr/mask authtype
- host replication replica 10.1.0.99/32 md5
- pg_reload_conf()

Создание слотов репликации (только при необходимости).

Настройка мастера.

Отдельный пользователь для репликации.

Редактирование postgresql.conf.

Редактирование pg_hba.conf.

Создание слотов репликации (только при необходимости).

- max_replication_slots > 0
- pg_create_physical_replication_slot('slotname');
- primary_slot_name = 'slotname' (recovery.conf)

Копирование DATADIR.

pg_basebackup (с версии 9.1)

-h, --host=...; -p, --port=...; -U, --username=...; -d, --dbname=...; -D, --pgdata=...

-c, --checkpoint=fast | spread

-X, --xlog-method=fetch | stream – stream с 9.2

-R, --write-recovery-conf – с 9.3

-r, --max-rate=… – с 9.4

--xlogdir=… – с 9.4

-T, --tablespace-mapping=olddir=newdir – с 9.4

-P, --progress

pg_basebackup -P -R -X stream -c fast -h 127.0.0.1 -U replica -D /pgdb

Копирование DATADIR. Альтернативы pg_basebackup.

Копирование через cp, scp, tar, rsync...

Snapshots:

- ZFS send/receive;
- LVM + dd.

pg_start_backup() + pg_stop_backup().

Настройка Standby.

Идентичность файлов конфигурации.

Файлы конфигурации:

- postgresql.conf;
- recovery.conf.

Настройка Standby.

Идентичность файлов конфигурации:

- Зачем?
- Как?

Файлы конфигурации:

- postgresql.conf;
- recovery.conf.

Настройка Standby.

Идентичность файлов конфигурации.

Файлы конфигурации (postgresql.conf):

- hot_standby = on;
- max_standby_streaming_delay;
- wal_receiver_status_interval;
- hot_standby_feedback;
- wal_receiver_timeout;

Настройка Standby.

Идентичность файлов конфигурации.

Файлы конфигурации (recovery.conf):

- primary_conninfo = 'host=… port=…'
- standby_mode = on
- primary_slot_name = 'slotname'
- trigger_file = '...'
- recovery_min_apply_delay.

Запуск Standby.

pg_ctl — родная утилита от PostgreSQL.

pg_ctlcluster — враппер над pg_ctl в Debian/Ubuntu.

sysvinit, upstart, openrc, systemd…

Проверка успешности запуска.

Наличие процессов wal sender и wal receiver.

Проверка журнала событий.

Проверка через обычное подключение (psql).

Системное представление pg_stat_replication.

Разные особенности.

DATADIR, конфиги и Debian-based vs. RHEL-based.

pg_ctlcluster и ошибка невозможности подключения.

Подозрительно малое количество процессов в выводе ps.

Итоги. Практическое назначение реплик.

Масштабируемость на чтение.

Полнотекстовый поиск.

Аналитика.

Реплика не значит backup.

Вопросы.

Настройка репликации. Практическая часть.

root password: pgconf2016

# su - postgres	– все задачи делаем от имени postgres
$ ps auxf	– что вообще мы имеем?
$ pwd	– где мы?
$ ls -l 9.5	– где все будет происходить.

Подготовка мастера.

$ vi 9.5/data/postgresql.conf

- listen_addresses = '*' – слушаем на всех интерфейсах.
- wal_level = hot_standby – подробность WAL журнала.
- max_wal_senders = 4 – лимит на walsender'ы.
- hot_standby = on – разрешить read-only запросы.

Подготовка мастера.

$ psql – создаем отдельного пользователя.

```
CREATE ROLE replica WITH LOGIN REPLICATION PASSWORD 'rep123';
```

$ vi .pgpass – настраиваем файл паролей.

```
*:*:*:replica:rep123
```

$ chmod 600 .pgpass

$ vi 9.5/data/pg_hba.conf – прописываем правила авторизации.

```
host    replication     replica         127.0.0.1/32            md5
```

$ pg_ctl -D 9.5/data/ -m fast restart – применяем изменения в силу.

Создание standby сервера.

$ pg_basebackup -P -R -c fast -X stream -h 127.0.0.1 -U replica -D 9.5/replica

- -c fast — сделать принудительный checkpoint.
- -X stream — скопировать новые XLOG через выделенное содениние.
- -R — создать минимальный recovery.conf

$ vi 9.5/replica/postgresql.conf – правим номер порт.

port = 5433

$ pg_ctl -D 9.5/replica/ start – запускаем реплику.

Проверка.

```
$ ps auxf                                  – наличие sender/receiver процессов.
$ psql -p 5433                             – проверяем статус на standby.
      select pg_is_in_recovery();          – режим восстановления на standby.
$ psql                                     – проверяем статус на мастере.
      select * from pg_stat_replication ;  – наличие статистики со standby.
```

Вопросы.

Это все.

Часть III. Мониторинг и эксплуатация.

«Настроил и забыл» - это про потоковую репликацию.

Мониторинг:

- внутренняя статистика;
- вспомогательные функции;
- примеры запросов.

Эксплуатация:

- добавление и удаление реплик;
- приостановка репликации;
- добавление и удаление слотов.

Мониторинг.

Системные представление (views):

- pg_stat_replication
- pg_stat_replication_slots

Мониторинг. Вспомогательные функции.

pg_is_in_recovery()

pg_current_xlog_location()

pg_last_xact_replay_timestamp()

pg_last_xlog_receive_location()

pg_last_xlog_replay_location()

pg_xlog_location_diff()

Мониторинг. Примеры запросов.

Мониторинг репликации на мастере.

```
select
  pid, client_addr,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),sent_location)) as
pending_xlog,
  pg_size_pretty(pg_xlog_location_diff(sent_location,write_location)) as write,
  pg_size_pretty(pg_xlog_location_diff(write_location,flush_location)) as flush,
  pg_size_pretty(pg_xlog_location_diff(flush_location,replay_location)) as replay,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),replay_location)) as
total_lag
from pg_stat_replication;
```

```
  pid  |  client_addr   | pending_xlog |  write   |  flush  |  replay    | total_lag
-------+----------------+--------------+----------+---------+------------+-----------
 21015 | 127.0.0.1      | 0 bytes      | 0 bytes  | 0 bytes | 48 bytes   | 48 bytes
  2067 | 192.168.200.4 | 12 GB        | 30 MB    | 0 bytes | 156 kB     | 12 GB
 18635 | 192.168.100.2 | 0 bytes      | 48 bytes | 0 bytes | 590 MB     | 590 MB
```

Мониторинг.

Для standby:

pg_current_xlog_location() → pg_last_xlog_receive_location()

Объем WAL:

- SELECT pg_xlog_location_diff(pg_current_xlog_location, '0/0');

Лаг в секундах:

- SELECT now() - pg_last_xact_replay_timestamp();

Эксплуатация.

Добавление новых или отключение standby узлов.

Временная приостановка репликации и возобновление.

Эксплуатация.

Добавление новых или отключение standby узлов.

- max_wal_senders
- max_replication_slots
- pg_create_physical_replication_slot()
- pg_drop_replication_slot()

Временная приостановка репликации и возобновление.

Эксплуатация.

Добавление новых или отключение standby узлов.

Временная приостановка репликации и возобновление.

- pg_is_xlog_replay_paused()
- pg_xlog_replay_pause()
- pg_xlog_replay_resume()

Вопросы.

Часть IV. Проблемы и решения.

Лаг репликации.

Полная остановка репликации.

Сетевые и дисковые проблемы.

100% использование места на диске.

Конфликты при восстановлении.

Bloat таблиц и индексов.

pg_xlog/ bloat.

Лаг репликации.

Симптомы:

- данные на standby отличаются от того что на мастере.

Причины:

- долгие запросы на standby, много записи на мастере;
- проблемы с оборудованием.

Решение:

- оптимизация приложения.

Networking и Storage.

Network lag:

- full_page_writes = off;
- ssh tunnels с компрессией.

Storage lag:

- full_page_writes =off;
- filesystem barriers;
- writethrough/writeback;
- RAID BBU learning;
- ionice (только для cfq elevator).

Полная остановка репликации.

Симптомы:

- Recovery process использует 100% CPU;
- Лаг увеличивается.

Причины:

- Тяжелые update/delete, много autovacuum.

Решение:

- увеличение wal_keep_segments;
- временное отключение full_page_writes;
- расстановка приоритетов через ionice и renice.

100% использование места на диске.

Причины:

- Replication slots и выключенный standby → накопление XLOG сегментов;

Решение:

- Удаление слота и использование wal_keep_segments.

Хак на скорую руку:

- Filesystem's reserved blocks percentage и tune2fs.

Конфликты при восстановлении.

Как возникают конфликты:

- Autovacuum;
- XLOG replay.

Решение:

- hot_standby_feedback = on;
- Увеличение max_standby_streaming_delay.

Bloat таблиц и индексов.

Причина:

- Длинные транзакции на standby.

Решение:

- pgstattuple;
- VACUUM FULL, pgcompacttable, pg_reorg...;

pg_xlog/ bloat на standby.

Симптомы:

- Разный размер pg_xlog/ и количество XLOG сегментов.

Решение:

- Уменьшить checkpoint_timeout;
- Уменьшить checkpoint_completion_target.

Вопросы.

Часть V. Switchover и Failover.

Что это?

Зачем и когда это нужно?

Как это делать?

Вводная информация.

Switchover и Failover.

Зачем:

- обновление ПО, ОС, оборудования.
- отказ оборудования.

Switchover.

Запуск chekpoint на мастере.

Проверка лага репликации.

Выключение мастера.

Удаление recovery.conf и рестарт standby.

Switchover.

Плюсы:

- Быстрое подключение старого мастер в качестве standby;
- Отсутствие потерянных транзакций.

Минусы:

- После рестарта требуется прогрев кэша;
- pg_prewarm extension (с версии 9.4).

Failover.

Создание триггерного файла

- recovery.conf: trigger_file = '…'
- После изменения recovery.conf нужен рестарт.

Использование pg_ctl

- pg_ctl -D ... promote

Failover.

Плюсы:

- быстро;
- не нужно время на рестарт;
- не нужен прогрев кэша.

Минусы:

- риск потери транзакций;
- нельзя использовать старый мастер в качестве standby (до 9.5).

Повторное использование старого мастера.

Switchover:

- создание recovery.conf и запуск.

Failover:

- переинициализация (до 9.5);
- pg_rewind (с 9.5).
  - timeline должен различаться у мастера и standby.
  - старый мастер должен быть корректно завершен.
  - таки иногда бывают issues.
  - pg_rewind --target-pgdata=9.5/main --source-server="host=10.0.0.1"

Вопросы.

Switchover. Практическая часть.

```
$ vi 9.5/replica/postgresql.conf          – заранее поправим конфиг.
      port = 5432
$ mv 9.5/replica/recovery.conf /tmp/      – удаляем recovery.conf
$ psql
> CHECKPOINT;                             – сократим время будущего рестарта.
$ pg_ctl -D 9.5/data -m fast stop         – выключаем мастер.
$ pg_ctl -D 9.5/replica -m fast restart   – провозглашем нового мастера.
$ tail -f 9.5/replica/pg_log/postgresql-Wed.log
$ ps auxf
```

Switchover. Как быть со старым мастером?

```
$ vi 9.5/data/postgresql.conf          – правим конфиг.
      port = 5433
$ mv /tmp/recovery.conf 9.5/data/      – создаем recovery.conf.
$ pg_ctl -D 9.5/data start             – запускаем.
$ ps auxf                              – проверяем.
```

Failover. Подготовка.

$ vi 9.5/data/postgresql.conf

$ vi 9.5/replica/postgresql.conf

- wal_log_hints = on
- wal_keep_segments = 32

$ pg_ctl -D 9.5/data -m fast restart

$ pg_ctl -D 9.5/replica -m fast restart

Failover.

```
$ pg_ctl -D 9.5/replica -m immediate stop
```
– «крэшим» мастер.

```
$ pg_ctl -D 9.5/data promote
```
– создаем новый мастер.

```
$ psql -p 5433
```
– вносим изменения.

```
    create database test;
```

Failover. А что со старым мастером?

```
$ pg_ctl -D 9.5/replica start
$ pg_ctl -D 9.5/replica stop
$ pg_rewind -D 9.5/replica --source-server="host=127.0.0.1 port=5433"
$ vi 9.5/replica/postgresql.conf
        port = 5432
$ mv 9.5/replica/recovery.done 9.5/replica/recovery.conf
$ vi 9.5/replica/recovery.conf
        port = 5433
$ pg_ctl -D 9.5/replica start
$ ps auxf
```

Вопросы.

Спасибо за внимание.

Алексей Лесовский, PostgreSQL Consulting.

Лаунж зона, стойка PGDay