Logi WAL: włączenie i archiwizacja

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17, Linux i Windows** **Czas: ~20 min**

Backup pełny mówi, jak wyglądała baza w chwili jego wykonania — archiwum WAL mówi, co się z nią działo od tamtej chwili do teraz, i to ono jest jedynym sposobem na odtworzenie stanu bazy sprzed pięciu minut zamiast sprzed ostatniej północy.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
LSN	Log Sequence Number: rosnący numer pozycji w dzienniku transakcyjnym, po którym silnik rozpoznaje kolejność zmian
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

TYLKO ODCZYT

Utworzenie testowych logów transakcyjnych: pg_start_backup / pg_stop_backup

```
SELECT pg_start_backup('testing_archiving');
```

Rozpoczynamy tryb backupu na gorąco, co wymusza pełny zapis kolejnych stron do WAL (`full_page_writes`) i pozwala bezpiecznie skopiować pliki danych mimo trwających na nich zapisów — przydatne też jako sposób na wygenerowanie kontrolnego ruchu w `pg_wal/`, żeby sprawdzić, czy archiwizacja skonfigurowana wyżej faktycznie działa.

Przykład:

```
 pg_start_backup
-----------------
 0/3000028
(1 row)
```

```
SELECT pg_stop_backup();
```

Kończymy tryb backupu i wymuszamy przełączenie na nowy segment WAL, co powinno natychmiast wywołać `archive_command` dla segmentu zamkniętego w trakcie testu.

Przykład:

```
 pg_stop_backup
-----------------
 0/3000130
(1 row)
```

Uważaj: `pg_start_backup` i `pg_stop_backup` w tej postaci są oznaczone jako przestarzałe od PostgreSQL 15 na rzecz `pg_backup_start()` i `pg_backup_stop()` — starsze nazwy wciąż działają dla zgodności wstecznej, ale nowa dokumentacja i nowe skrypty powinny używać nazw bez podkreślnika `start_backup`.

Wersja	Rozpoczęcie backupu	Zakończenie backupu
do PostgreSQL 14	`pg_start_backup(label)`	`pg_stop_backup()`
od PostgreSQL 15	`pg_backup_start(label)`	`pg_backup_stop()`

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

TYLKO ODCZYT

Wymuszenie przełączenia segmentu WAL: pg_switch_wal

```
SELECT pg_switch_wal();
```

Zamykamy bieżący segment WAL natychmiast, bez czekania aż się zapełni albo aż upłynie `archive_timeout` — wygodne przy ręcznym teście archiwizacji, kiedy nie chcemy czekać na naturalne wypełnienie segmentu.

Przykład:

```
 pg_switch_wal
---------------
 0/3000178
(1 row)
```

Uważaj: do PostgreSQL 9.6 ta sama funkcja nazywała się `pg_switch_xlog()` — nazwa zmieniła się razem z przemianowaniem katalogu `pg_xlog` na `pg_wal` w wersji 10, kiedy projekt ujednolicił terminologię „xlog" na „WAL" we wszystkich nazwach funkcji i katalogów.

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

TYLKO ODCZYT

Sprawdzenie bieżącej pozycji WAL: pg_current_wal_lsn

```
SELECT pg_current_wal_lsn();
```

> **Pojęcie: LSN (Log Sequence Number).** Unikalny, rosnący wskaźnik pozycji w strumieniu WAL — dwie liczby szesnastkowe oddzielone ukośnikiem, oznaczające plik segmentu i przesunięcie w jego środku. LSN jest tym, czego używa się do porównania, jak daleko replika nadąża za instancją główną, i tym, do czego odwołuje się odtwarzanie do punktu w czasie.

Przykład:

```
 pg_current_wal_lsn
---------------------
 0/3000210
(1 row)
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **LSN** = Log Sequence Number: rosnący numer pozycji w dzienniku transakcyjnym, po którym silnik rozpoznaje kolejność zmian

Z życia wzięte

Z ŻYCIA WZIĘTE

Ustawiamy `archive_command` na pg01, żeby wreszcie mieć archiwum WAL pod bazę Zabbixa. Katalog docelowy na dysku backupu okazuje się niezapisywalny dla konta `postgres`, więc archiwizacja po cichu się nie udaje, a `pg_wal` rośnie noc po nocy. Zauważamy to dopiero, gdy wolumen ostrzega o zapełnieniu w dziewięćdziesięciu procentach.

Czym jest WAL w PostgreSQL

Pojęcie: WAL. Log transakcyjny, do którego PostgreSQL zapisuje każdą zmianę danych, zanim ta zmiana trafi do właściwych plików tabel — mechanizm opisany ogólnie na stronie Log transakcyjny: WAL, redo, transaction log. W PostgreSQL segmenty WAL leżą w katalogu `pg_wal/` wewnątrz `PGDATA`, opisanym na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy, i domyślnie mają po 16 MB każdy.

Pojęcie: archiwizacja WAL. Kopiowanie zamkniętych segmentów WAL w bezpieczne miejsce, zanim silnik je nadpisze — bez archiwizacji PostgreSQL po pewnym czasie zwyczajnie kasuje stare segmenty, żeby nie zapełnić dysku, i historia zmian starsza niż aktualnie przechowywane segmenty znika bezpowrotnie. Włączenie archiwizacji jest warunkiem koniecznym backupu przyrostowego i odtwarzania do punktu w czasie, tematu Odtwarzanie do punktu w czasie (PITR).

Konfiguracja postgresql.conf dla archiwizacji logów

```
wal_level = replica
```

Pojęcie: wal_level. Ilość informacji zapisywanej do WAL — `minimal` zapisuje tylko tyle, ile potrzeba do odtworzenia po awarii, bez możliwości replikacji ani archiwizacji; `replica` dokłada informacje potrzebne do replikacji fizycznej i do archiwizacji; `logical` dokłada dodatkowo dane potrzebne do replikacji logicznej. Do archiwizacji WAL potrzebny jest co najmniej poziom `replica`.

Uważaj: wartość `archive`, spotykana w starszych konfiguracjach sprzed PostgreSQL 9.6, była osobnym poziomem — od 9.6 wzwyż `archive` nadal jest akceptowane jako alias, ale wewnętrznie oznacza dokładnie to samo co `replica`. Nowe instalacje powinny wprost używać `replica`, nie polegać na przestarzałym aliasie.

```
archive_mode = on
```

Włącza samo uruchamianie mechanizmu archiwizacji. Dopuszczalne wartości to `off`, `on` i `always` — ta ostatnia archiwizuje WAL także na serwerze repliki, nie tylko na głównym.

```
archive_command = 'cp %p /backup/db_log/%f'
```

> **Pojęcie: archive_command.** Polecenie powłoki wywoływane przez instancję dla każdego zamkniętego segmentu WAL — to na nas spoczywa napisanie polecenia, które faktycznie skopiuje plik w bezpieczne miejsce; PostgreSQL sam niczego nie archiwizuje poza wywołaniem tego polecenia. Dwa placeholdery są tu kluczowe: `%p` to pełna ścieżka do segmentu, który ma zostać skopiowany, `%f` to sama nazwa pliku, bez ścieżki.

```
archive_timeout = 1800
```

Wymusza przełączenie na nowy segment WAL po podanej liczbie sekund, nawet jeśli bieżący segment nie zapełnił się jeszcze danymi — bez tego przy niewielkim ruchu transakcyjnym ostatnie zmiany mogłyby czekać w niezamkniętym segmencie godzinami, niezarchiwizowane. Wartość `0` wyłącza wymuszanie.

```
max_wal_size = 1GB
min_wal_size = 80MB
```

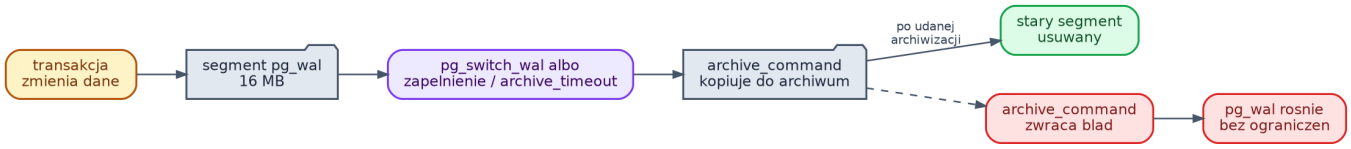
Miękki górny i dolny limit łącznego rozmiaru segmentów WAL trzymanych w `pg_wal/` między checkpointami — nie są to twarde granice, ale wskazówka dla checkpointera, kiedy odpalić kolejny punkt kontrolny, żeby WAL nie urósł bez kontroli.

Poniższy komentarz z oryginalnego `postgresql.conf` przypomina, że część parametrów WAL wymaga restartu instancji, nie samego przeładowania opisanego na stronie postgresql.conf i pg_hba.conf:

```
# WRITE-AHEAD LOG
#checkpoint_timeout = 5min      # range 30s-1d
                                # (change requires restart)
```

Uważaj: to konkretne ostrzeżenie w komentarzu dotyczy w oryginalnej dokumentacji nie samego `checkpoint_timeout` (ten jest przeładowywalny), tylko sąsiadujących z nim parametrów

określających rozmiar pamięci dzielonej na WAL — `wal_level` i `wal_buffers` wymagają restartu, `checkpoint_timeout`, `archive_command` i `archive_timeout` wystarczy przeładować.



Segment WAL zamyka się przy zapełnieniu albo archive_timeout, archive_command kopiuje go do archiwum, a stary segment znika dopiero po udanej archiwizacji (kliknięcie otwiera pełny rozmiar)

Wersja dla Windows

Ten sam mechanizm, ta sama zmienna `archive_command`, inna składnia polecenia kopiującego — Windows nie zna `cp`, więc `archive_command` wywołuje `move` albo `copy` z osobnym cudzysłowowaniem ścieżek zawierających spacje i ukośniki wsteczne:

```
archive_command = 'move "%p" "G:\PostgreSQL\logs\%f"'
```

Element	Linux	Windows
Polecenie kopiujące	`cp`	`move` albo `copy`
Separator ścieżki	`/`	`\`, wymaga podwojenia w niektórych kontekstach albo ujęcia w cudzysłów
Cudzysłowowanie ścieżki	opcjonalne, chyba że ścieżka zawiera spacje	zwykle wymagane z powodu dwukropka po literze dysku i możliwych spacji

Częste błędy

Włączenie archive_mode bez działającego archive_command. Jeśli polecenie archiwizujące zwraca błąd albo katalog docelowy nie istnieje, instancja nie usuwa starych segmentów WAL, dopóki archiwizacja się nie powiedzie — `pg_wal/` rośnie bez ograniczeń, aż do zapełnienia dysku i zatrzymania bazy.

Testowanie archive_command na katalogu bez uprawnień zapisu dla konta postgres. Polecenie w `archive_command` wykonuje się jako użytkownik systemowy, na którym działa instancja — katalog docelowy musi być dla niego zapisywalny, inaczej każda próba archiwizacji kończy się

cichą porażką widoczną tylko w logu serwera.

Powiązane: Log transakcyjny: WAL, redo, transaction log · Architektura PostgreSQL: klaster, baza, schemat, procesy · pgBackRest: podstawowe komendy · Odtwarzanie do punktu w czasie (PITR)

Źródła: dokumentacja PostgreSQL, rozdział „Reliability and the Write-Ahead Log" i „Continuous Archiving and Point-in-Time Recovery"

Revision #44
Created 2026-09-24 23:08:09 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:33 UTC by Tomasz Konieczny