Backup i odtwarzanie

WAL, pg_dump i pgBackRest.

- Logi WAL: włączenie i archiwizacja
- pg_dump, pg_dumpall, pg_restore
- pgBackRest: podstawowe komendy

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
```

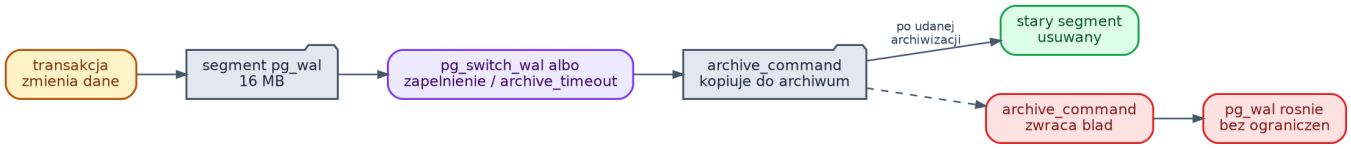
Miękki górny i dolny limit łącznego rozmiaru segmentów WAL trzymanych w `pg_wal/` między checkpointami — nie są to twarde granice, ale wskazówka dla checkpointera, kiedy odpalić kolejny punkt kontrolny, żeby WAL nie urósł bez kontroli.

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"

pg_dump, pg_dumpall, pg_restore

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~25 min**

Backup logiczny bierze dane, nie pliki — wynikiem jest zestaw poleceń `CREATE`/`COPY`, które da się odtworzyć na innej wersji silnika, na innym systemie operacyjnym, a nawet tylko dla jednej tabeli z całej bazy.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

ZMIENIA SYSTEM

Zrzut jednej tabeli: pg_dump -t

```
pg_dump -t NazwaSchematu.NazwaTabeli -h NazwaSerwera -U NazwaUsera NazwaBazy >
SciezkaDoPliku
```

Zrzucamy strukturę i dane dokładnie jednej tabeli, bez reszty bazy — wygodne, gdy potrzebujemy tylko punktowej kopii przed ryzykowną operacją na tej jednej tabeli.

Przykład:

```
pg_dump -t sprzedaz.zamowienia -h pg01 -U nazwa_usera produkcja >
/home/nazwa_usera/zamowienia_dump.sql
```

Polecenie przy sukcesie nic nie wypisuje na ekran — cały wynik trafia do wskazanego pliku.

pg_dump = zrzut bazy PostgreSQL do pliku

□□ TYLKO ODCZYT

Import zrzutu w formacie plain: psql

```
psql -h NazwaSerwera -U NazwaUsera NazwaBazy < SciezkaDoPliku
```

Odtwarzamy zrzut tekstowy zwykłym klientem `psql`, tak jakby jego treść wpisano ręcznie w sesji — dokładnie ten sam mechanizm, co metakomenda `\i` opisana na stronie psql: klient wiersza poleceń.

Przykład:

```
psql -h pg01 -U nazwa_usera produkcja < /home/nazwa_usera/zamowienia_dump.sql
```

Przykład:

```
CREATE TABLE
COPY 48213
ALTER TABLE
```

□□ **psql** = PostgreSQL SQL: klient bazy PostgreSQL

□□ OSTROŻNIE · NIEODWRACALNE

Wyczyszczenie tabeli przed importem nadpisującym: TRUNCATE

```
TRUNCATE TABLE NazwaTabeli;
```

Jeśli import ma nadpisać całą zawartość tabeli, a nie dołożyć wiersze do istniejących, opróżniamy tabelę przed importem — inaczej `COPY` ze zrzutu doda swoje wiersze obok już istniejących, co przy kluczach głównych bez `ON CONFLICT` skończy się błędem duplikatu klucza. Mechanizm `TRUNCATE` opisuje strona Podstawowe operacje na bazie i użytkownikach.

Przykład:

```
TRUNCATE TABLE
```

□□ TYLKO ODCZYT

Zrzut jednego schematu w formacie custom: pg_dump -n -Fc

```
pg_dump -U NazwaUsera -d NazwaBazy -n NazwaSchematu -Fc -f SciezkaDoPliku
```

Zrzucamy cały schemat, wraz z danymi wszystkich jego tabel, w formacie binarnym gotowym pod migrację do innej bazy — dokładnie to zapytanie stoi za migracją danych ze schematu do osobnej bazy docelowej.

Przykład:

```
pg_dump -U nazwa_usera -d produkcja -n sprzedaz -Fc -f my_schema.dump
```

pg_dump = zrzut bazy PostgreSQL do pliku

TYLKO ODCZYT

Odtworzenie zrzutu custom lokalnie: pg_restore

```
pg_restore -U NazwaUsera -d NazwaBazy -n NazwaSchematu SciezkaDoPliku
```

Odtwarzamy zrzut w bazie na tym samym serwerze, na którym uruchamiamy polecenie — baza docelowa musi już istnieć, `pg_restore` sam jej nie tworzy.

Przykład:

```
pg_restore -U nazwa_usera -d docelowa -n sprzedaz my_schema.dump
```

pg_restore = odtwórz zrzut bazy PostgreSQL zapisany w formacie custom albo katalogowym

TYLKO ODCZYT

Odtworzenie zrzutu custom na zdalnej bazie: pg_restore -h -p

```
pg_restore -U NazwaUsera -h NazwaSerwera -p NumerPortu -d NazwaBazy -n NazwaSchematu
SciezkaDoPliku
```

Ten sam mechanizm, ale połączenie idzie do innego serwera — dokładnie to, czego potrzebuje migracja danych z jednej instancji do drugiej, na przykład przy przenoszeniu środowiska testowego z serwera produkcyjnego.

Przykład:

```
pg_restore -U nazwa_usera -h pg02 -p 5432 -d docelowa -n sprzedaz my_schema.dump
```

□□ **pg_restore** = odtwórz zrzut bazy PostgreSQL zapisany w formacie custom albo katalogowym

□□ TYLKO ODCZYT

Zrzut samej struktury, bez danych: pg_dump --schema-only

```
pg_dump --schema-only -U NazwaUsera -d NazwaBazy -Fc -f SciezkaDoPliku
```

Zrzucamy wyłącznie definicje obiektów — `CREATE TABLE`, indeksy, ograniczenia — bez ani jednego wiersza danych. Przydatne przy przygotowaniu pustej kopii bazy pod środowisko testowe albo przy dokumentowaniu struktury schematu.

Przykład:

```
pg_dump: last built-in OID is 16383
```

□□ **pg_dump** = zrzut bazy PostgreSQL do pliku

□□ TYLKO ODCZYT

Zrzut samych danych, bez struktury: pg_dump --data-only

```
pg_dump --data-only -U NazwaUsera -d NazwaBazy -Fc -f SciezkaDoPliku
```

Odwrotność poprzedniego — zakłada, że struktura docelowa już istnieje, i przenosi wyłącznie zawartość tabel. Częsty wybór przy odświeżaniu danych w środowisku testowym, którego struktura schematu jest już aktualna.

□□ **pg_dump** = zrzut bazy PostgreSQL do pliku

□□ TYLKO ODCZYT

Zrzut równoległy: pg_dump -Fd -j

```
pg_dump -U NazwaUsera -d NazwaBazy -Fd -j NumerPortu -f SciezkaDoPliku
```

> **Pojęcie: dump równoległy.** Format `directory` pozwala `pg_dump` zrzucać wiele tabel jednocześnie, oddzielnymi procesami — flaga `-j` (jobs) ustala, ile procesów roboczych ma pracować naraz. Na bazie z wieloma dużymi, niezależnymi tabelami skraca to czas backupu proporcjonalnie do liczby rdzeni, jakie oddamy zadaniu — nie działa dla

formatów `plain` ani `custom`, tylko dla `directory`.

Przykład dla czterech procesów:

```
pg_dump -U nazwa_usera -d produkcja -Fd -j 4 -f /backup/produkcja_dump/
```

pg_dump = zrzut bazy PostgreSQL do pliku

TYLKO ODCZYT

Odtworzenie równoległe: pg_restore -j

```
pg_restore -U NazwaUsera -d NazwaBazy -j NumerPortu SciezkaDoPliku
```

Analogicznie do dumpa, `pg_restore` odtwarza wiele tabel jednocześnie, jeśli źródłem jest zrzut w formacie `directory` — dla formatu `custom` równoległość działa tylko częściowo, bo mimo że plik jest jeden, `pg_restore` potrafi rozdzielić między procesy ładowanie danych i budowę indeksów.

Przykład:

```
pg_restore -U nazwa_usera -d docelowa -j 4 /backup/produkcja_dump/
```

pg_restore = odtwórz zrzut bazy PostgreSQL zapisany w formacie custom albo katalogowym

ZMIENIA SYSTEM

Zrzut wszystkich ról klastra: pg_dumpall --roles-only

```
pg_dumpall --roles-only -U NazwaUsera > SciezkaDoPliku
```

`pg_dump` pojedynczej bazy nie zawiera definicji ról ani ich haseł — role są wspólne dla całego klastra, nie własnością jednej bazy. Przy odtwarzaniu bazy na nowym klastrze, w którym te same role jeszcze nie istnieją, ten zrzut wykonujemy przed odtworzeniem samych danych.

Przykład:

```
pg_dumpall --roles-only -U postgres > /backup/role.sql
```

◫◫ ZMIENIA SYSTEM

Zrzut wszystkich tablespace'ów klastra: pg_dumpall --tablespaces-only

```
pg_dumpall --tablespaces-only -U NazwaUsera > SciezkaDoPliku
```

Analogicznie do ról, definicje tablespace'ów — czyli wskazań na katalogi na dysku spoza `PGDATA` — należą do całego klastra, nie do pojedynczej bazy, i trzeba je odtworzyć osobno, zanim odtworzymy bazy, które z tych tablespace'ów korzystają.

Przykład:

```
pg_dumpall --tablespaces-only -U postgres > /backup/tablespaces.sql
```

Z życia wzięte

◫◫ Z ŻYCIA WZIĘTE

Przed ryzykowną migracją panelu Oli robimy `pg_dump` samej tabeli `zamowienia` z bazy produkcyjnej na pg01, zamiast całej bazy. Import na środowisku testowym kończy się błędem duplikatu klucza, bo zapomnieliśmy wcześniej zrobić `TRUNCATE` tabeli docelowej. Po wyczyszczeniu tabeli drugi import przechodzi bez problemu.

Backup logiczny a backup fizyczny

Pojęcie: backup logiczny. Zrzut danych w postaci poleceń SQL albo ustrukturyzowanego formatu pośredniego, wykonany przez odpytanie bazy, a nie przez skopiowanie plików z dysku — w odróżnieniu od backupu fizycznego, jakim zajmuje się pgBackRest: podstawowe komendy, backup logiczny nie wymaga zatrzymania instancji ani identycznej wersji silnika przy odtwarzaniu. Ogólny podział backupów rozwija strona Backup pełny, różnicowy, przyrostowy, logi — `pg_dump` w tym podziale jest zawsze backupem pełnym pojedynczej bazy albo jej fragmentu, nigdy przyrostowym.

`pg_dump` zrzuca jedną bazę danych, `pg_dumpall` zrzuca cały klaster wraz z rolami i tablespace'ami, których `pg_dump` celowo nie dotyka, bo to obiekty wspólne dla całego klastra, opisane na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy, a nie własność jednej bazy. `pg_restore` odtwarza zrzut zapisany w jednym z formatów binarnych `pg_dump` — zrzutu w formacie tekstowym `pg_restore` nie obsługuje, ten odtwarza się zwykłym `psql`.

Formaty zrzutu

Format	Flaga	Charakterystyka	Odtwarzanie
plain	`-Fp` (domyślny)	zwykły tekst SQL, czytelny i edytowalny w edytorze	`psql < plik.sql`
custom	`-Fc`	format binarny, skompresowany, pozwala na selektywne odtworzenie wybranych obiektów	`pg_restore`
directory	`-Fd`	katalog z osobnym plikiem na każdą tabelę, jedyny format wspierający równoległy dump i restore jednocześnie	`pg_restore`
tar	`-Ft`	archiwum tar, binarne jak custom, ale bez kompresji i bez możliwości zmiany kolejności obiektów przy odtwarzaniu	`pg_restore`

Format `plain` sprawdza się przy małych bazach i przy zrzutach, które chcemy przejrzeć albo zmodyfikować ręcznie przed odtworzeniem. Format `custom` jest domyślnym wyborem do backupu produkcyjnego — kompresja oszczędza miejsce, a `pg_restore` pozwala odtworzyć tylko wybrane tabele ze zrzutu całej bazy bez ponownego dumpowania. Format `directory` wybieramy, gdy zależy nam na czasie — to jedyny format, w którym zarówno zrzut, jak i odtworzenie mogą działać wieloma procesami naraz.

Częste błędy

Próba odtworzenia zrzutu custom klientem psql. Format binarny `-Fc` nie jest tekstem SQL — `psql < plik.dump` kończy się nieczytelnym błędem, bo próbuje zinterpretować binarną zawartość jako polecenia SQL. Formaty `custom`, `directory` i `tar` odtwarzamy wyłącznie `pg_restore`.

Pominięcie pg_dumpall --roles-only przy migracji na nowy klaster. Odtworzenie samej bazy na świeżym klastrze bez wcześniejszego odtworzenia ról kończy się błędami `role "NazwaUsera" does not exist` przy próbie przywrócenia uprawnień zapisanych w zrzucie.

Powiązane: Backup pełny, różnicowy, przyrostowy, logi · pgBackRest: podstawowe komendy · Architektura PostgreSQL: klaster, baza, schemat, procesy · Podstawowe operacje na bazie i użytkownikach

Źródła: dokumentacja PostgreSQL, rozdział „pg_dump", „pg_dumpall" i „pg_restore"

pgBackRest: podstawowe komendy

Poziom: podstawy **Dotyczy: pgBackRest 2.x, PostgreSQL 9.6–17** **Czas: ~25 min**

`pg_dump` zrzuca dane przez SQL, w jednym punkcie w czasie — pgBackRest kopiuje pliki na poziomie systemu i dokłada do tego archiwum WAL, dzięki czemu potrafi odtworzyć bazę nie tylko do chwili backupu, ale do dowolnej sekundy między jednym backupem a drugim.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
RPO	Recovery Point Objective: ile danych wolno stracić przy awarii
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

ZMIENIA SYSTEM

Utworzenie nowej stanzy: stanza-create

```
pgbackrest --stanza=NazwaStanzy stanza-create --log-level-console=info
```

Zakładamy w repozytorium strukturę katalogów dla nowej stanzy — wykonujemy raz, przy pierwszej konfiguracji backupu dla danego klastra.

Przykład:

```
pgbackrest --stanza=produkcja stanza-create --log-level-console=info
```

Przykład:

```
P00   INFO: stanza-create command begin
P00   INFO: stanza-create for stanza 'produkcja' on repo1
P00   INFO: stanza-create command end: completed successfully
```

pgbackrest = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Sprawdzenie poprawności konfiguracji stanzy: check

```
pgbackrest --stanza=NazwaStanzy check
```

Sprawdzamy, czy pgBackRest potrafi połączyć się z instancją, zapisać do repozytorium i czy archiwizacja WAL działa poprawnie — pierwsze polecenie do uruchomienia po każdej zmianie konfiguracji, zanim spróbujemy właściwego backupu.

Przykład:

```
P00   INFO: check command begin
P00   INFO: WAL segment 000000010000000000000003 successfully archived
P00   INFO: check command end: completed successfully
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **pgbackrest** = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Wykonanie backupu pełnego: backup --type=full

```
pgbackrest --stanza=NazwaStanzy backup --type=full
```

Kopiujemy wszystkie pliki bazy od zera — najdłuższy i najbardziej miejscochłonny typ backupu, wykonywany zwykle raz na tydzień albo raz na miesiąc, w zależności od rozmiaru bazy i ustalonego RPO, tematu strony RPO i RTO: dwie liczby, od których zaczyna się każdy plan backupu.

Przykład:

```
P00   INFO: backup command begin
P00   INFO: full backup size = 4.7GB
P00   INFO: new backup label = 20260924-030000F
P00   INFO: backup command end: completed successfully
```

RPO = Recovery Point Objective: ile danych wolno stracić przy awarii · **RTO** = Recovery Time Objective: jak długo wolno nie działać przy awarii · **pgbackrest** = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Wykonanie backupu różnicowego: backup --type=diff

```
pgbackrest --stanza=NazwaStanzy backup --type=diff
```

> **Pojęcie: backup różnicowy.** Kopiuje tylko to, co zmieniło się od ostatniego backupu pełnego — każdy kolejny backup różnicowy rośnie, aż do następnego pełnego, ale odtworzenie wymaga tylko dwóch backupów: ostatniego pełnego i najnowszego różnicowego.

Przykład:

```
P00   INFO: backup command begin
P00   INFO: diff backup size = 340MB
P00   INFO: new backup label = 20260925-030000F_20260925-030512D
P00   INFO: backup command end: completed successfully
```

pgbackrest = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Wykonanie backupu przyrostowego: backup --type=incr

```
pgbackrest --stanza=NazwaStanzy backup --type=incr
```

> **Pojęcie: backup przyrostowy.** Kopiuje tylko to, co zmieniło się od poprzedniego backupu, niezależnie od tego, czy tamten był pełny, różnicowy czy również

przyrostowy — najmniejszy i najszybszy z trzech typów, ale odtworzenie wymaga całego łańcucha backupów od ostatniego pełnego aż do wybranego przyrostowego.

Przykład:

```
P00   INFO: backup command begin
P00   INFO: incr backup size = 62MB
P00   INFO: new backup label = 20260925-030000F_20260926-030214I
P00   INFO: backup command end: completed successfully
```

pgbackrest = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Sprawdzenie dostępnych backupów: info

```
pgbackrest info
```

Wyświetlamy listę stanz skonfigurowanych w repozytorium wraz z historią backupów każdej z nich — pierwsze polecenie przy planowaniu odtworzenia, żeby wiedzieć, do jakich punktów w czasie w ogóle można wrócić.

Przykład:

```
stanza: produkcja
    status: ok
    cipher: none

    db (current)
        wal archive min/max (14): 000000010000000000000001/000000010000000000000009

        full backup: 20260924-030000F
            timestamp start/stop: 2026-09-24 03:00:00 / 2026-09-24 03:41:12
            wal start/stop: 000000010000000000000001 / 000000010000000000000001
            database size: 4.7GB, database backup size: 4.7GB

        diff backup: 20260925-030000F_20260925-030512D
            timestamp start/stop: 2026-09-25 03:00:00 / 2026-09-25 03:04:30
            database size: 4.8GB, database backup size: 340MB
```

pgbackrest = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

ZMIENIA SYSTEM

Odtworzenie z ostatniego backupu: restore

```
pgbackrest --stanza=NazwaStanzy restore
```

Odtwarzamy `PGDATA` z najnowszego dostępnego backupu i dogrywamy WAL z archiwum aż do ostatniego zarchiwizowanego segmentu — instancja musi być zatrzymana, a katalog danych pusty albo usunięty przed uruchomieniem tego polecenia.

Przykład:

```
P00   INFO: restore command begin
P00   INFO: restore backup set 20260925-030000F_20260925-030512D
P00   INFO: restore command end: completed successfully
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **pgbackrest** = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

ZMIENIA SYSTEM

Odtworzenie do punktu w czasie: restore --type=time --target

```
pgbackrest --stanza=NazwaStanzy --type=time --target="2026-09-24 10:00:00" restore
```

> **Pojęcie: odtwarzanie do punktu w czasie.** Zamiast zatrzymać się na ostatnim dostępnym backupie, pgBackRest odtwarza backup pełny, a następnie odtwarza z archiwum WAL wszystkie zmiany aż do wskazanej sekundy — nie ani chwili dalej. To jest jedyny sposób na cofnięcie bazy do stanu sprzed konkretnego, znanego zdarzenia, na przykład tuż przed błędnym `UPDATE` bez `WHERE`, temat rozwinięty na stronie Odtwarzanie do punktu w czasie (PITR).

Przykład:

```
P00   INFO: restore command begin
P00   INFO: restore backup set 20260924-030000F
```

```
P00   INFO: recovery type is time, target is '2026-09-24 10:00:00'
P00   INFO: restore command end: completed successfully
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **PITR** = Point-In-Time Recovery: odtworzenie bazy do dowolnej chwili, np. 14:31 · **pgbackrest** = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

TYLKO ODCZYT

Usunięcie przeterminowanych backupów: expire

```
pgbackrest --stanza=NazwaStanzy expire
```

> **Pojęcie: expire (wygaszanie retencji).** Usuwa z repozytorium backupy starsze niż retencja zdefiniowana parametrami `repo1-retention-*` w `pgbackrest.conf`, wraz z segmentami WAL, które nie są już potrzebne żadnemu z pozostałych backupów. `expire` uruchamiany jest zwykle automatycznie po każdym backupie, ale da się go wywołać też ręcznie.

Przykład:

```
P00   INFO: expire command begin
P00   INFO: expire full backup 20260910-030000F
P00   INFO: expire command end: completed successfully
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **pgbackrest** = Linux: narzędzie do kopii fizycznych PostgreSQL z archiwizacją WAL, tu do odtworzenia bazy na maszynie testowej

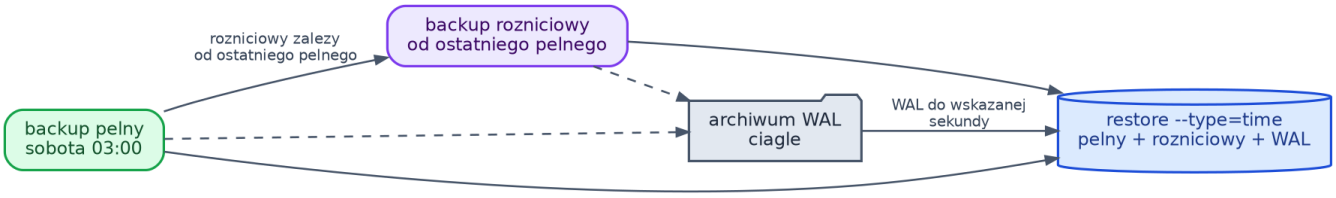
Z życia wzięte

Z ŻYCIA WZIĘTE

Wieczorem w sobotę pgBackRest robi pełny backup bazy Zabbixa na pg01, a przez kolejny tydzień dokłada tylko backupy różnicowe. W czwartek Ola przypadkowo kasuje dane z panelu i pytamy `pgbackrest info`, do jakiego punktu w czasie w ogóle możemy wrócić. Odtwarzamy backup pełny plus najnowszy różnicowy i dogrywamy WAL do minuty tuż przed błędnym poleceniem Oli.

Czym jest pgBackRest

> **Pojęcie: pgBackRest.** Narzędzie do backupu fizycznego i archiwizacji WAL dla PostgreSQL, zarządzające całym cyklem: backupem pełnym, backupami różnicowymi i przyrostowymi, kompresją, retencją i odtwarzaniem, łącznie z odtwarzaniem do punktu w czasie. W przeciwieństwie do `pg_dump`, opisanego na stronie pg_dump, pg_dumpall, pg_restore, pgBackRest nie odpytuje bazy przez SQL — kopiuje bezpośrednio pliki danych z `PGDATA`, korzystając z tego samego trybu backupu na gorąco, którego mechanizm pokazuje `pg_start_backup` / `pg_stop_backup` na stronie Logi WAL: włączenie i archiwizacja.



Backup pełny w sobotę, backup różnicowy zależny od niego i ciągłe archiwum WAL, odtworzenie do punktu w czasie łączy wszystkie trzy (kliknięcie otwiera pełny rozmiar)

	pg_dump	pgBackRest
Poziom działania	logiczny, przez SQL	fizyczny, na poziomie plików
Wynik	dane w formacie przenośnym między wersjami	kopia plików danego klastra
Backup przyrostowy	nie istnieje	tak, na poziomie zmienionych bloków
Odtwarzanie do punktu w czasie	nie	tak, dzięki archiwum WAL
Migracja między wersjami głównymi	tak	nie — odtwarza dokładnie tę samą wersję, z której backup powstał

Ogólny podział typów backupu — pełny, różnicowy, przyrostowy — i to, kiedy który wybrać, rozwija strona Backup pełny, różnicowy, przyrostowy, logi. pgBackRest jest narzędziem, które w PostgreSQL realizuje wszystkie trzy typy jednym spójnym mechanizmem retencji.

Konfiguracja pgbackrest.conf

> **Pojęcie: stanza.** W terminologii pgBackRest to nazwana konfiguracja jednego klastra PostgreSQL objętego backupem — jeden plik `pgbackrest.conf` może trzymać konfigurację kilku stanz, po jednej sekcji na każdy klaster, jeśli na serwerze repozytorium backupujemy więcej niż jedną instancję.

Konfiguracja backupów leży domyślnie w `/etc/pgbackrest/pgbackrest.conf`. Przykład konfiguracji z zapisem na dedykowanym wolumenie `/pgbackrest` i retencją czternastu dni:

```
[global]
repo1-path=/pgbackrest
repo1-retention-full-type=time
repo1-retention-full=14
process-max=2
log-level-console=info
log-level-file=debug

[produkcja]
pg1-path=/postgres/14/data
pg1-port=5432
pg1-user=postgres
```

Sekcja `[global]` obejmuje ustawienia wspólne dla wszystkich stanz na danym serwerze repozytorium, sekcja nazwana tak jak stanza (`[produkcja]` w przykładzie) wskazuje, gdzie leży `PGDATA` konkretnego klastra i jakim kontem systemowym pgBackRest ma się do niego łączyć.

Parametr	Znaczenie
`repo1-path`	katalog repozytorium backupów
`repo1-retention-full-type`	tryb retencji — `time` (dni) albo `count` (liczba backupów pełnych)
`repo1-retention-full`	liczba dni albo backupów pełnych do zachowania, zależnie od trybu wyżej
`process-max`	liczba równoległych procesów przy backupie i odtwarzaniu
`log-level-console` / `log-level-file`	poziom szczegółowości logów na konsoli i w pliku
`pg1-path`	ścieżka `PGDATA` klastra objętego backupem
`pg1-port`	port instancji
`pg1-user`	konto systemowe, którym pgBackRest łączy się z instancją

Częste błędy

Wykonanie stanza-create bez wcześniej działającej archiwizacji WAL. `stanza-create` sprawdza, czy `archive_command` z Logi WAL: włączenie i archiwizacja w ogóle działa — jeśli nie, polecenie kończy się błędem, zanim jeszcze dojdzie do pierwszego backupu.

Odtwarzanie backupu przyrostowego bez zachowania całego łańcucha. Backup przyrostowy jest bezużyteczny bez wszystkich poprzedzających go backupów aż do ostatniego pełnego — usunięcie pojedynczego ogniwa łańcucha (np. ręcznie, mimo działającego `expire`) unieważnia wszystkie kolejne po nim backupy różnicowe i przyrostowe.

Powiązane: Backup pełny, różnicowy, przyrostowy, logi · Logi WAL: włączenie i archiwizacja · Odtwarzanie do punktu w czasie (PITR) · pg_dump, pg_dumpall, pg_restore

Źródła: dokumentacja pgBackRest, `pgbackrest.org/user-guide.html`