Logi serwera: konfiguracja i czytanie

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~20 min**

Log serwera jest jedynym miejscem, w którym widać, co instancja robiła w chwili, gdy nikt na nią nie patrzył — deadlock sprzed dwóch godzin, checkpoint wymuszony w środku nocy, plik tymczasowy, który zapełnił dysk o czwartej rano.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
HTML	HyperText Markup Language: format tekstowy stron WWW, tu używany jako format raportu GPO do przeglądania w przeglądarce
PID	Process ID: numer procesu w systemie

Gdzie trafiają logi

> **Pojęcie: logging_collector.** Osobny proces w tle, opisany szerzej na stronie Architektura PostgreSQL: klaster, baza, schemat, procesy, który przechwytuje standardowe wyjście błędów serwera i zapisuje je do rotowanych plików na dysku. Bez włączonego `logging_collector` komunikaty lecą tam, skąd usługa została uruchomiona — zwykle do dziennika systemowego `systemd` / `journald`, nie do osobnych plików w katalogu logów.

Fiszka

Włączenie kolektora logów: logging_collector

```
logging_collector = on
```

Włączamy zapis do plików w katalogu wskazanym niżej, zamiast polegać wyłącznie na dzienniku systemowym. Parametr wymaga restartu instancji.

Wskazanie katalogu i wzorca nazwy pliku

```
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```

`log_directory` jest ścieżką względną wobec `PGDATA`, chyba że podamy ścieżkę bezwzględną. Wzorzec `log_filename` przyjmuje te same znaczniki `strftime` co polecenie `date` — nowy plik dla każdego uruchomienia instancji jest wygodny przy diagnozowaniu, ale przy aktywnej rotacji czasowej (niżej) prowadzi do bardzo wielu małych plików.

Komplet trzech dróg, którymi komunikaty mogą opuścić instancję:

Cel	log_destination	Gdzie szukać	Kiedy wybieramy
pliki tekstowe silnika	`stderr`	`PGDATA/log/*.log`	domyślny wybór, najprostszy do przeczytania ręcznie i do podania narzędziom typu pgBadger
dziennik systemowy	`syslog`	`/var/log/messages` albo dedykowany plik `rsyslog`	środowiska z centralnym zbieraniem logów po syslogu z wielu maszyn naraz
`journald`	`stderr` bez `logging_collector`, albo dedykowana integracja	`journalctl -u postgresql-15`	dystrybucje oparte o `systemd`, gdzie chcemy jednego miejsca na logi wszystkich usług bez dodatkowej konfiguracji plików

Uważaj: `syslog` i `journald` dodają własny nagłówek do każdej linii (znacznik czasu, nazwa procesu) — narzędzia takie jak pgBadger, opisane niżej, oczekują surowego formatu `stderr` i wymagają dodatkowego kroku wyciągnięcia samej treści komunikatu PostgreSQL z owijki syslogowej.

Z życia wzięte

Z ŻYCIA WZIĘTE

Baza cennik na pg01 zaczyna generować dziwne błędy zgłaszane przez dział handlowy, ale nikt nie wie, które dokładnie zapytanie ich powoduje. Włączamy `log_min_duration_statement` na tysiąc milisekund i po godzinie mamy w logu dokładną treść wolnego zapytania razem z jego czasem wykonania. Po naprawieniu zapytania wracamy do domyślnego progu, żeby log nie rósł bez potrzeby.

Parametry treści logu

Ustawienie progu logowania wolnych zapytań: log_min_duration_statement

```
log_min_duration_statement = 1000
```

Pojęcie: log_min_duration_statement. Próg czasu wykonania w milisekundach, powyżej którego zapytanie trafia do logu wraz z treścią i realnym czasem trwania — wartość `0` loguje każde zapytanie, `-1` wyłącza logowanie po czasie całkowicie. Ustawienie w okolicach 500–1000 ms jest typowym punktem startowym na produkcji: łapie realne problemy wydajnościowe bez zalewania logu każdym pojedynczym `SELECT` aplikacji.

Wybór, które polecenia logować niezależnie od czasu: log_statement

```
log_statement = 'ddl'
```

Wartość	Co loguje
`none`	nic, niezależnie od `log_min_duration_statement`
`ddl`	wyłącznie polecenia zmieniające strukturę — `CREATE`, `ALTER`, `DROP`
`mod`	`ddl` plus polecenia zmieniające dane — `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`
`all`	każde wykonane polecenie

Uważaj: `log_statement = 'all'` na produkcji z realnym ruchem potrafi wygenerować log rzędu gigabajtów dziennie i sam w sobie staje się obciążeniem I/O — sięgamy po niego czasowo, do diagnozy konkretnego problemu, nie jako stały stan.

Dobór znaczników w prefiksie każdej linii logu: log_line_prefix

```
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
```

> **Pojęcie: log_line_prefix.** Szablon poprzedzający każdą linię logu własnymi danymi kontekstowymi — bez niego log pokazuje sam komunikat, bez informacji, kto i kiedy go wywołał. Znaczniki najczęściej przydatne w codziennej diagnostyce:

Znacznik	Co wstawia
`%t`	znacznik czasu bez milisekund
`%m`	znacznik czasu z milisekundami — wymagany przez pgBadger do poprawnego sortowania zdarzeń
`%p`	PID procesu backendu, ten sam, który widać w `pg_stat_activity`
`%u`	nazwa roli, która się połączyła
`%d`	nazwa bazy, do której trwa połączenie
`%a`	nazwa aplikacji klienckiej (`application_name`), jeśli klient ją ustawił
`%h`	adres hosta klienta
`%l`	numer kolejny linii logu w ramach tej sesji

Logowanie momentów wykonania checkpointu: log_checkpoints

```
log_checkpoints = on
```

Każdy checkpoint, opisany na stronie Logi WAL: włączenie i archiwizacja, dopisuje do logu linię z czasem trwania, liczbą zapisanych buforów i tym, czy został wywołany zgodnie z harmonogramem, czy wymuszony wcześniej — dokładnie te same dwie kategorie, które zlicza `pg_stat_bgwriter` opisany na stronie Monitoring instancji.

Logowanie nawiązania i zamknięcia połączenia: log_connections, log_disconnections

```
log_connections = on
log_disconnections = on
```

Każde nawiązane i zamknięte połączenie trafia do logu wraz z czasem trwania sesji — przydatne przy diagnozowaniu aplikacji nawiązującej połączenia w pętli zamiast korzystać z puli, ale przy dużym ruchu krótkich połączeń dokłada zauważalny wolumen do logu.

Logowanie sesji czekających na blokadę: log_lock_waits

```
log_lock_waits = on
```

Sesja czekająca na zwolnienie blokady dłużej niż `deadlock_timeout` (domyślnie 1 sekunda) dostaje wpis do logu — mechanizm blokad i sam deadlock opisuje strona Blokady: współdzielone, wyłączne, eskalacja. To ten sam próg czasowy, po którym silnik sprawdza, czy oczekiwanie nie jest już zakleszczeniem.

Logowanie zapytań korzystających z plików tymczasowych: log_temp_files

```
log_temp_files = 0
```

Zapytanie, które w trakcie sortowania albo haszowania przekroczy `work_mem`, opisany na stronie postgresql.conf i pg_hba.conf, zaczyna korzystać z plików tymczasowych na dysku zamiast z samej pamięci. Wartość `0` loguje każdy taki plik niezależnie od rozmiaru, wartość dodatnia ustawia próg w kilobajtach, `-1` wyłącza logowanie całkowicie.

Logowanie długo trwających operacji autovacuum: log_autovacuum_min_duration

```
log_autovacuum_min_duration = 500
```

Przebieg autovacuum trwający dłużej niż podany próg w milisekundach trafia do logu z tym samym raportem, jaki pokazuje ręczne `VACUUM (VERBOSE, ANALYZE)` opisane na stronie VACUUM, autovacuum i bloat — bez tego parametru autovacuum działa całkowicie po cichu, chyba że coś pójdzie nie tak.

Rotacja logów

```
log_rotation_age = 1d
log_rotation_size = 100MB
log_truncate_on_rotation = off
```

> **Pojęcie: rotacja logu.** Zamknięcie bieżącego pliku logu i rozpoczęcie nowego, żeby żaden pojedynczy plik nie rósł bez końca. `log_rotation_age` wymusza nowy plik po upływie czasu, `log_rotation_size` po przekroczeniu rozmiaru — który z warunków spełni się pierwszy, ten wywołuje rotację. `log_truncate_on_rotation = on` ma sens tylko przy stałym, powtarzalnym wzorcu nazwy pliku (np. dzień tygodnia) — nadpisuje plik z poprzedniego cyklu zamiast dokładać kolejny, ograniczając łączną liczbę plików kosztem historii starszej niż jeden cykl.

Przy `log_destination = syslog` rotacją zajmuje się mechanizm systemowy (`logrotate`), nie sam PostgreSQL — powyższe trzy parametry dotyczą wyłącznie kolektora wewnętrznego.

Czytanie typowych komunikatów

Checkpoint zbyt częsty:

```
LOG:  checkpoints are occurring too frequently (18 seconds apart)
HINT:  Consider increasing the configuration parameter "max_wal_size".
```

Silnik sam podpowiada rozwiązanie w linii `HINT` — `max_wal_size` jest za mały wobec realnego ruchu transakcyjnego, przez co checkpointy wymuszane są znacznie częściej niż wynikałoby to z `checkpoint_timeout`, co widać też w rosnącym `checkpoints_req` opisanym na stronie Monitoring instancji.

Sesja czekająca na blokadę:

```
LOG:  process 24519 still waiting for ShareLock on transaction 88213401 after 1000.145 ms
DETAIL:  Process holding the lock: 24501. Wait queue: 24519.
```

Linia `DETAIL` od razu podaje PID sesji trzymającej blokadę — dokładnie ten sam numer, który zwraca `pg_blocking_pids()` opisane na stronie Sesje, długie zapytania, kasowanie kwerendy, tylko odczytany z logu zamiast z zapytania na żywo.

Plik tymczasowy:

```
LOG:  temporary file: path "base/pgsql_tmp/pgsql_tmp24501.0", size 87652352
STATEMENT:  SELECT * FROM zamowienia ORDER BY wartosc DESC
```

Rozmiar pliku w bajtach pokazuje, o ile zapytanie przekroczyło dostępny `work_mem` — linia `STATEMENT` od razu wskazuje, które konkretnie zapytanie jest winowajcą, bez konieczności włączania pełnego `log_statement = 'all'`.

pgBadger

> **Pojęcie: pgBadger.** Zewnętrzne narzędzie, napisane w Perlu, generujące z surowego logu serwera jeden, przeglądalny w przeglądarce raport HTML — wykresy najczęstszych zapytań, rozkład czasu wykonania, checkpointy, blokady i połączenia w jednym miejscu, zamiast ręcznego przeszukiwania plików tekstowych poleceniem `grep`. Wymaga, żeby `log_line_prefix` zawierał przynajmniej znacznik czasu z milisekundami (`%m`) i PID (`%p`) — bez nich narzędzie nie potrafi poprawnie posortować i pogrupować zdarzeń.

Wygenerowanie raportu HTML z logu

```
pgbadger /var/log/postgresql/postgresql-2026-09-24.log -o /tmp/raport.html
```

Przetwarzamy jeden plik logu i zapisujemy wynik jako pojedynczy plik HTML, gotowy do otwarcia w przeglądarce bez żadnej dodatkowej infrastruktury.

Przykład:

```
[=========================>] Parsed 842103 bytes of 842103 (100.00%), queries: 12481, events: 342
LOG: Ok, generating html report...
```

Wariant przetwarzający wiele plików naraz, typowy przy dziennej rotacji logu z ostatniego tygodnia:

```
pgbadger /var/log/postgresql/postgresql-2026-09-*.log -o /tmp/raport-tygodniowy.html
```

Częste błędy

log_statement = 'all' zostawione na stałe na produkcji. Przydatne przy jednorazowej diagnozie, kosztowne jako stały stan — dokłada I/O na każde, nawet najprostsze zapytanie i szybko zapełnia dysk logami.

log_line_prefix bez znacznika %m. pgBadger i podobne narzędzia potrzebują milisekund do poprawnego posortowania zdarzeń w obrębie tej samej sekundy — sam `%t` nie wystarcza przy dużym ruchu.

Zapomnienie o restarcie po włączeniu logging_collector. Parametr wymaga restartu instancji, nie samego przeładowania opisanego na stronie postgresql.conf i pg_hba.conf — sama zmiana w pliku bez restartu nie uruchamia procesu kolektora.

Powiązane: postgresql.conf i pg_hba.conf · Monitoring instancji · Sesje, długie zapytania, kasowanie kwerendy · VACUUM, autovacuum i bloat

Źródła: dokumentacja PostgreSQL, rozdział „Error Reporting and Logging" · dokumentacja projektu `pgBadger`

Revision #41
Created 2026-09-25 00:33:24 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:45 UTC by Tomasz Konieczny