Architektura i instalacja

Jak zbudowany jest klaster PostgreSQL, gdzie trzyma pliki i jak postawić go z własnym katalogiem danych.

- PostgreSQL: z Berkeley do każdej firmy
- Architektura PostgreSQL: klaster, baza, schemat, procesy

PostgreSQL: z Berkeley do każdej firmy

Poziom: podstawy **Dotyczy: PostgreSQL 9.6-17** **Czas: ~15 min**

Jedyny z czterech silników tej bazy wiedzy, który nie ma właściciela ani rachunku za licencję. Ta strona otwiera książkę „PostgreSQL": skąd wziął się ten silnik, jak trafił do Zielonej i co czeka na kolejnych stronach.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
BSD	Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń
DARPA	Defense Advanced Research Projects Agency: agencja badawcza amerykańskiej armii, jeden ze sponsorów projektu POSTGRES na Berkeley
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Skąd się wzięło

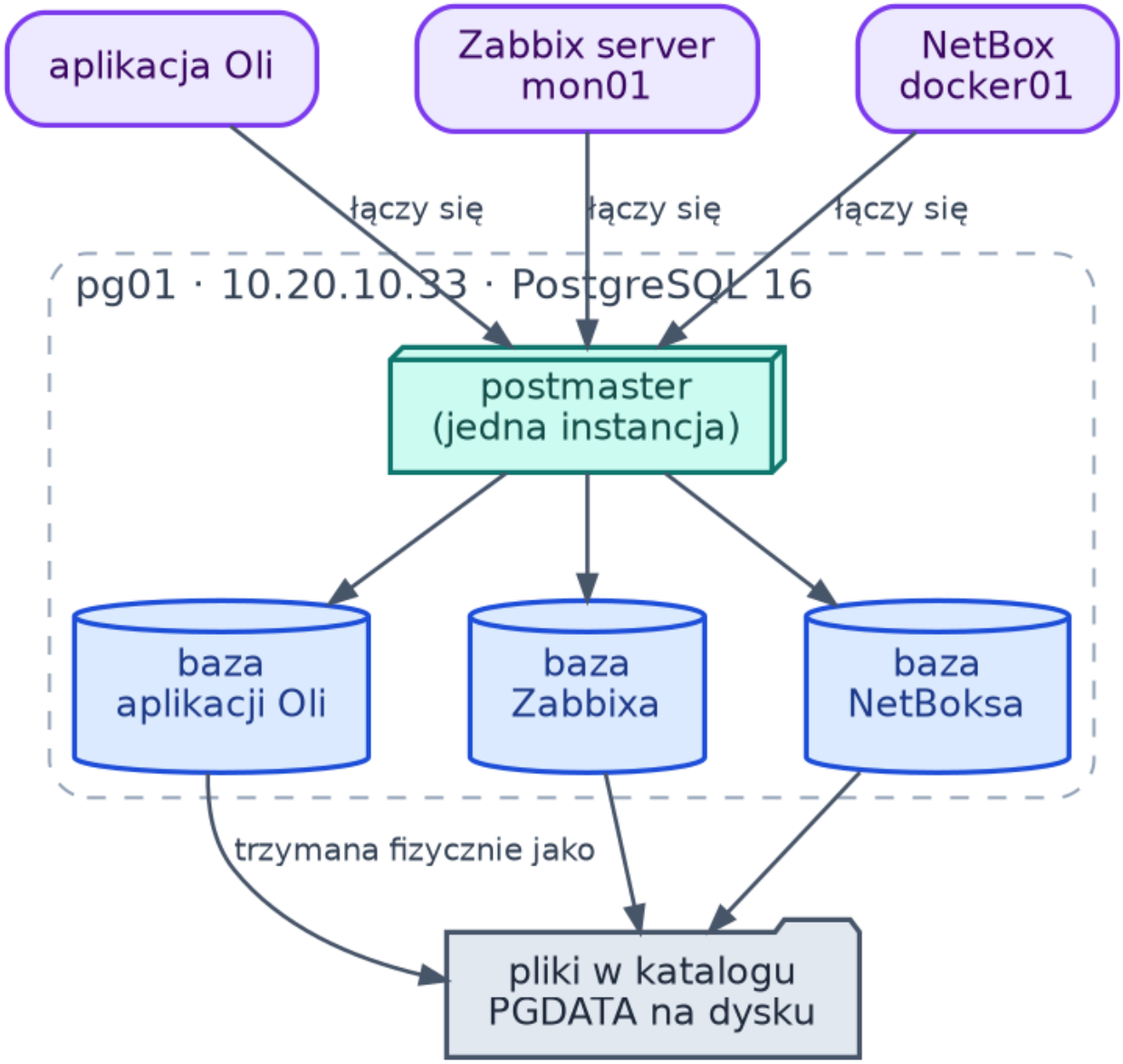
Zanim powstał PostgreSQL, na Uniwersytecie Kalifornijskim w Berkeley powstał Ingres, wczesny silnik relacyjny prowadzony przez Michaela Stonebrakera w latach siedemdziesiątych. Stonebraker skomercjalizował Ingres, sprzedał swoją firmę i wrócił na uczelnię z pomysłem na następcę: silnik, który zamiast trzymać się sztywno modelu relacyjnego, pozwoli programiście definiować własne typy danych i reguły. W 1986 roku ruszył na Berkeley projekt o nazwie POSTGRES, sfinansowany między innymi przez agencję badawczą DARPA. Nazwa mówiła wprost, czym ma być: tym, co po (post) Ingres.

Pierwsza działająca wersja POSTGRES trafiła do testów w 1987 roku, a projekt rozwijał się na uczelni przez kolejne sześć lat, doczekawszy się czterech kolejnych wydań. Silnik miał wtedy własny język zapytań, PostQUEL, niezgodny z SQL-em, który w tym czasie stawał się standardem branży. W 1994 roku dwóch studentów Berkeley, Andrew Yu i Jolly Chen, dopisało do POSTGRES interpreter SQL i wydało wynik pod nazwą Postgres95, jako darmowe, otwarte oprogramowanie.

W 1996 roku projekt zmienił nazwę na PostgreSQL, żeby oddać fakt, że silnik rozumie już SQL, zachowując jednocześnie oryginalne dziedzictwo POSTGRES. Numeracja wersji ruszyła od 6.0, kontynuując numerację z Berkeley. Od tego momentu PostgreSQL rozwija globalna społeczność ochotników i firm, nie jedna spółka: licencja PostgreSQL, bardzo zbliżona do BSD, pozwala każdemu używać, modyfikować i sprzedawać silnik bez opłaty, co odróżnia go od pozostałych trzech silników w tej bazie wiedzy.

Jak to działa w jednym obrazku

Niezależnie od tego, ile różnych aplikacji łączy się z PostgreSQL, droga jest ta sama: klient łączy się z jedną instancją, instancja kieruje połączenie do właściwej bazy w tym samym klastrze danych, a wynik pracy ląduje fizycznie jako plik w katalogu `PGDATA`.



Trzy aplikacje z Zielonej łączą się z jedną instancją PostgreSQL na pg01, każda do własnej bazy w tym samym klastrze danych (kliknięcie otwiera pełny rozmiar)

Dlaczego Zielona

> **Z ŻYCIA WZIĘTE**
>
> Kiedy zespół Oli postawił drugi wewnętrzny panel obok pierwszego, oba połączyły się bezpośrednio z tym samym `pg01`, nie sprawdzając, że w tej samej instancji siedzi już baza Zabbixa. W środku miesiąca limit `max_connections` omal nie odciął Zabbixowi monitoringu całej sieci, bo dwa nowe panele otwierały po kilkadziesiąt sesji naraz. Tomek dostał zadanie:

> postawić przed aplikacjami pulę połączeń, żeby kolejny nowy panel nie zabierał połączeń ważniejszej usłudze.

Na `pg01` PostgreSQL trzyma dziś bazy trzech różnych odbiorców: aplikacji zespołu Oli, Zabbixa monitorującego całą sieć i NetBoksa dokumentującego adresację. Żaden z nich nie musiał kupować licencji ani negocjować liczby rdzeni, bo PostgreSQL nie ma cennika. To właśnie ten brak tarcia sprawia, że w firmach takich jak Zielona PostgreSQL staje się domyślnym wyborem dla każdej nowej wewnętrznej aplikacji, nawet jeśli nikt formalnie tej decyzji nie podjął.

Jedna instancja obsługująca kilka niepowiązanych aplikacji ma jednak swoją cenę: awaria albo przeciążenie jednej bazy potrafi odbić się na sąsiadach w tym samym klastrze, bo dzielą jeden proces nadzorczy i jeden limit połączeń. Reszta tej książki pokazuje, jak rozpoznać taką sytuację, zanim zauważy ją Zabbix albo Marek z magazynu.

Mapa tej książki

Książka „PostgreSQL" składa się z dziewięciu rozdziałów, budowanych na słowniku z książki „Bazy danych: fundamenty".

Rozdział	Na jakie pytanie odpowiada
PostgreSQL: z Berkeley do każdej firmy	Skąd wziął się PostgreSQL i dlaczego trafia dziś do tylu firm
Architektura i instalacja	Z czego składa się klaster PostgreSQL i jak postawić go z własnym katalogiem danych
Połączenie i konfiguracja	Jak połączyć się klientem `psql` i co ustawić w plikach konfiguracyjnych
Rozpoznanie bazy	Jak sprawdzić rozmiary, obiekty i aktywność na nieznanej instancji
Użytkownicy i uprawnienia	Jak zarządzać rolami i kontami technicznymi
Backup i odtwarzanie	Jak działa WAL i jak zrobić kopię zapasową `pg_dump` albo pgBackRest
Wydajność i utrzymanie	Czym jest VACUUM, bloat i plan wykonania zapytania
Procedury	Instalacja, przeniesienie katalogu danych, upgrade silnika
Replikacja	Replikacja strumieniowa i logiczna
Awarie	Objaw, przyczyna, pierwsze kroki

Powiązane: Baza danych, silnik, instancja i schemat · Architektura PostgreSQL, klaster, baza, schemat, procesy

Źródła: dokumentacja PostgreSQL, „A Brief History of PostgreSQL", postgresql.org/docs/current/history.html · Wikipedia, „PostgreSQL" · Momjian B., „The History of PostgreSQL Open Source Development"

Architektura PostgreSQL: klaster, baza, schemat, procesy

Poziom: podstawy **Dotyczy: PostgreSQL 9.6–17** **Czas: ~15 min**

Jedna instancja PostgreSQL to jeden proces nadzorczy, jeden katalog na dysku i dowolna liczba baz danych w środku — ta strona pokazuje, co się kryje pod tym jednym katalogiem i jakie procesy krążą wokół niego cały czas.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
MSSQL	Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima
RAC	Real Application Clusters: klaster Oracle, kilka serwerów na jedną bazę
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

TYLKO ODCZYT

Sprawdzenie wersji silnika z poziomu SQL: version()

```
SELECT version();
```

Zwracamy pełny łańcuch wersji wraz z platformą kompilacji — przydatne, gdy trzeba dopasować dokumentację albo sprawdzić, czy poprawka bezpieczeństwa już jest zainstalowana.

Przykład:

```
                                                 version
-------------------------------------------------------------------------------------------
--------------
 PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.5.0 20210514, 64-bit
(1 row)
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych

TYLKO ODCZYT

Sprawdzenie wersji klienta z poziomu bash: psql -V

```
psql -V
```

Sprawdzamy wersję zainstalowanego klienta `psql`, niekoniecznie tę samą co wersja serwera, do którego się łączymy — klient i serwer mogą się różnić wersją poprawkową, a czasem i główną.

Przykład:

```
psql (PostgreSQL) 15.5
```

psql = PostgreSQL SQL: klient bazy PostgreSQL

TYLKO ODCZYT

Sprawdzenie katalogu danych bieżącego klastra: SHOW data_directory

```
SHOW data_directory;
```

Pytamy działającą instancję o jej własny `PGDATA` — to jest pewniejsze źródło niż zgadywanie ścieżki z dokumentacji dystrybucji, bo pokazuje katalog rzeczywiście używany przez ten konkretny proces.

Przykład:

```
   data_directory
---------------------
```

```
 /postgres/15/data
(1 row)
```

□□ **show** = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

□□ **TYLKO ODCZYT**

Sprawdzenie portu nasłuchu: SHOW port

```
SHOW port;
```

Sprawdzamy, na którym porcie TCP nasłuchuje instancja, do której jesteśmy akurat podłączeni — istotne, kiedy na jednym serwerze działa więcej niż jeden klaster.

Przykład:

```
 port
------
 5432
(1 row)
```

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**show** = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

□□ **TYLKO ODCZYT**

Sprawdzenie adresów nasłuchu: SHOW listen_addresses

```
SHOW listen_addresses;
```

Sprawdzamy, na jakich interfejsach sieciowych instancja przyjmuje połączenia — `localhost` oznacza wyłącznie gniazdo lokalne i pętlę zwrotną, `*` oznacza wszystkie interfejsy.

Przykład:

```
 listen_addresses
------------------
 *
(1 row)
```

show = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

TYLKO ODCZYT

Wylistowanie procesów silnika w systemie operacyjnym: ps

```
ps -ef | grep postgres
```

Sprawdzamy z poziomu systemu operacyjnego, jakie procesy PostgreSQL faktycznie działają — nazwa każdego procesu potomnego w wyniku odpowiada roli, jaką pełni w klastrze.

Przykład:

```
postgres    1102       1  0 08:01 ?        00:00:02 /usr/pgsql-15/bin/postgres -D
/postgres/15/data
postgres    1104    1102  0 08:01 ?        00:00:00 postgres: logger
postgres    1106    1102  0 08:01 ?        00:00:01 postgres: checkpointer
postgres    1107    1102  0 08:01 ?        00:00:00 postgres: background writer
postgres    1108    1102  0 08:01 ?        00:00:03 postgres: walwriter
postgres    1109    1102  0 08:01 ?        00:00:00 postgres: autovacuum launcher
postgres    1110    1102  0 08:01 ?        00:00:00 postgres: logical replication
launcher
postgres   14231    1102  0 09:41 ?        00:00:00 postgres: postgres produkcja
10.20.20.20(52344) idle
```

ps = process status: pokaż procesy

Z życia wzięte

Z ŻYCIA WZIĘTE

Tomek loguje się o 8:12 na pg01 i po raz pierwszy widzi wynik `ps -ef | grep postgres` z osobnym procesem systemowym na każdą sesję Zabbixa i aplikacji Oli. Liczy trzydzieści kilka procesów `postgres` i pyta, czy to normalne, że jedna baza uruchamia tyle osobnych procesów naraz. Tłumaczymy mu różnicę względem wątków MSSQL, którego uczono go na poprzedniej pracy.

Klaster danych

> **Pojęcie: klaster danych (database cluster).** W PostgreSQL to słowo nie oznacza tego, co w Oracle RAC czy w MSSQL Failover Cluster — nie chodzi o kilka serwerów fizycznych spiętych w jedną usługę. Klaster danych to pojedynczy katalog na dysku (`PGDATA`), zarządzany przez jeden proces nadzorczy, który może trzymać w sobie wiele baz danych naraz. Jedna instalacja PostgreSQL uruchomiona na jednym porcie to jeden klaster. Bazy w obrębie tego samego klastra dzielą między sobą jeden zestaw ról, jedną konfigurację uwierzytelniania i jeden limit połączeń — czego nie robią bazy w różnych klastrach, nawet uruchomionych na tym samym serwerze pod różnymi portami.
>
> **Pojęcie: instancja.** Działająca, uruchomiona kopia silnika: postmaster w pamięci razem z procesami tła, obsługująca żądania klientów. W PostgreSQL jedna instancja to zawsze jeden klaster danych, więc oba słowa chodzą tu parą, choć znaczą coś innego. Szerzej: Baza danych, silnik, instancja i schemat.

To rozróżnienie ma znaczenie praktyczne. Utworzenie nowej bazy komendą `CREATE DATABASE` nie uruchamia nowego procesu ani nie zajmuje nowego portu — to wciąż ten sam klaster, ten sam `PGDATA`, tylko jeden podkatalog więcej w `base/`. Nowa instancja, z osobnym `PGDATA` i osobnym portem, jest potrzebna dopiero wtedy, gdy chcemy oddzielić od siebie zestawy ról albo zainstalować obok siebie dwie różne wersje główne silnika — to jest właśnie sytuacja opisana na stronie Instalacja PostgreSQL z własnym katalogiem danych.

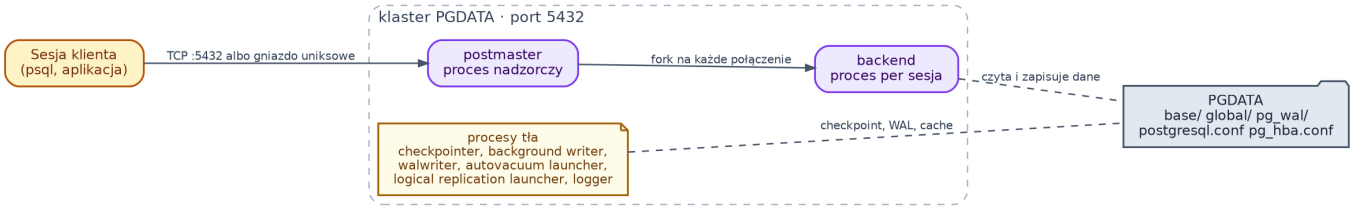
> **Pojęcie: schemat.** Przestrzeń nazw wewnątrz jednej bazy danych, grupująca tabele, widoki i inne obiekty, żeby dwie tabele o tej samej nazwie mogły istnieć obok siebie bez konfliktu. Domyślny schemat w PostgreSQL nazywa się `public`. Szerzej: Baza danych, silnik, instancja i schemat.

Baza, schemat i tabela mieszczą się wewnątrz klastra w ściśle określonym porządku, opisanym ogólnie na stronie Baza danych, silnik, instancja, schemat: cztery pojęcia, które się myli: jeden klaster, wiele baz, w każdej bazie wiele schematów, w każdym schemacie wiele tabel. Połączenie klienta zawsze celuje w jedną konkretną bazę — nie da się w jednym zapytaniu SQL odpytać dwóch baz z tego samego klastra tak, jak w MSSQL odpytuje się `bazaA.dbo.tabela` i `bazaB.dbo.tabela` jednym `SELECT`.

Postmaster i procesy potomne

>

> **Pojęcie: postmaster.** Pierwszy proces uruchamiany przy starcie klastra, widoczny w wyniku `ps` jako `/usr/pgsql-15/bin/postgres -D SciezkaDoPliku`. To on nasłuchuje na porcie TCP i na gnieździe uniksowym, przyjmuje przychodzące połączenia i dla każdego z nich odpala osobny proces potomny — proces roboczy zwany backendem. Postmaster sam nie wykonuje żadnego zapytania SQL, jego jedyną pracą jest przyjęcie połączenia, uwierzytelnienie klienta zgodnie z regułami z `pg_hba.conf` i uruchomienie procesu, który dalej obsłuży tę konkretną sesję.



Postmaster przyjmuje połączenie i odpala osobny proces backend dla każdej sesji, obok procesów tła piszących dane do katalogu PGDATA (kliknięcie otwiera pełny rozmiar)

Każda sesja klienta w PostgreSQL to osobny proces systemowy, nie wątek — inaczej niż w MSSQL, gdzie jeden proces `sqlservr.exe` obsługuje wszystkie sesje wątkami we wspólnej przestrzeni pamięci. Zaletą modelu procesowego jest izolacja: awaria jednej sesji nie zabiera ze sobą całej instancji. Ceną jest koszt pamięci i czasu na fork każdego nowego połączenia, co jest jednym z powodów, dla których duże wdrożenia PostgreSQL stawiają przed instancją pula połączeń (`PgBouncer` albo `pgpool-II`) zamiast łączyć się bezpośrednio z każdej aplikacji.

Procesy tła

> **Pojęcie: proces tła (background process).** Proces potomny postmastera, który nie obsługuje żadnej konkretnej sesji klienta, tylko wykonuje w tle jedno konkretne zadanie utrzymaniowe dla całego klastra, cyklicznie albo w reakcji na zdarzenie. Te procesy działają od momentu startu instancji do jej zatrzymania, niezależnie od tego, czy ktokolwiek jest akurat połączony.

> **Pojęcie: replikacja.** Mechanizm kopiowania zmian z jednej instancji (primary) na drugą (replika), tak żeby replika trzymała u siebie bieżącą kopię tych samych danych, zwykle z niewielkim opóźnieniem. PostgreSQL rozróżnia replikację strumieniową całego klastra i replikację logiczną wybranych tabel, opisane dalej w tej książce. Szerzej: Baza danych, silnik,

> instancja i schemat.

Procesy tła obecne w każdym uruchomionym klastrze:

- **checkpointer** — okresowo zrzuca na dysk wszystkie zabrudzone strony z pamięci podręcznej (`shared_buffers`) i zapisuje w `pg_wal` punkt kontrolny, od którego w razie awarii zacznie się odtwarzanie z logu WAL, tematu rozwiniętego na stronie Log transakcyjny: WAL, redo, transaction log. Częstotliwość reguluje parametr `checkpoint_timeout`, opisany na stronie postgresql.conf i pg_hba.conf.
- **background writer** — w tle, między checkpointami, systematycznie zapisuje na dysk niewielkie porcje zabrudzonych stron, żeby checkpoint nie musiał odkładać naraz gigabajtów danych.
- **WAL writer** — zapisuje bufor WAL z pamięci na dysk, niezależnie od tego, czy jakaś transakcja akurat robi `COMMIT` — dzięki temu zapis WAL nie czeka wyłącznie na commit pojedynczej sesji.
- **autovacuum launcher** — cyklicznie sprawdza, w których tabelach odsetek martwych krotek przekroczył próg, i uruchamia dla nich robocze procesy `autovacuum worker`. Mechanizm martwych krotek i sam `VACUUM` opisuje strona VACUUM, autovacuum i bloat.
- **logical replication launcher** — nadzoruje procesy `walsender`/`walreceiver` używane przy replikacji logicznej; obecny nawet wtedy, gdy żadna replikacja logiczna nie jest skonfigurowana.
- **logger** (`syslogger`) — zbiera komunikaty z pozostałych procesów i zapisuje je do plików logu zgodnie z parametrami `log_*` opisanymi na stronie postgresql.conf i pg_hba.conf.

Uważaj: do wersji 14 włącznie na liście procesów widoczny był też stats collector, zbierający statystyki użycia przez zapis do plików tymczasowych w `pg_stat_tmp`. Od PostgreSQL 15 ten proces został usunięty — statystyki aktywności trafiają bezpośrednio do pamięci dzielonej, co przy dużym ruchu zdejmuje z dysku sporo drobnych zapisów. Widoki takie jak `pg_stat_user_tables` czy `pg_stat_activity` działają tak samo w obu modelach, zmienia się tylko mechanizm pod spodem.

Układ katalogów PGDATA

> **Pojęcie: PGDATA.** Katalog danych klastra — jedyne miejsce na dysku, które trzeba zabezpieczyć backupem, żeby odtworzyć cały klaster wraz ze wszystkimi bazami, rolami i konfiguracją. Zawartość `PGDATA` jest wewnętrzną strukturą silnika, nie jest przeznaczona do ręcznej edycji poza kilkoma plikami tekstowymi opisanymi niżej.

> **Pojęcie: tablespace.** Nazwana lokalizacja na dysku, w której silnik trzyma pliki danych, niezależna od katalogu `PGDATA`. Tabela albo indeks utworzone w niestandardowym tablespace fizycznie leżą poza domyślnym katalogiem klastra, choć logicznie nadal należą do tej samej bazy. Szerzej: Baza danych, silnik, instancja i schemat.

Katalog / plik	Zawartość
`base/`	jeden podkatalog na każdą bazę danych w klastrze, nazwany numerem OID bazy; w środku pliki z danymi tabel i indeksów
`global/`	katalogi systemowe wspólne dla całego klastra — między innymi `pg_database` (lista baz) i `pg_authid` (role i hasła)
`pg_wal/`	segmenty logu WAL; do wersji 9.6 katalog nazywał się `pg_xlog`
`pg_tblspc/`	dowiązania symboliczne do katalogów tablespace'ów utworzonych poza `PGDATA`
`pg_stat_tmp/`	pliki tymczasowe stats collectora — obecny tylko do wersji 14
`postgresql.conf`	główny plik konfiguracyjny instancji, temat strony postgresql.conf i pg_hba.conf
`pg_hba.conf`	reguły uwierzytelniania klientów, ten sam temat
`pg_ident.conf`	mapowania nazw systemowych na role bazodanowe przy metodzie `ident` / `peer`
`PG_VERSION`	plik tekstowy z numerem wersji głównej, sprawdzany przy każdym starcie — niezgodność z wersją binarki blokuje start
`postmaster.pid`	plik blokady z PID-em działającego postmastera; jego obecność przy zatrzymanej instancji bywa źródłem błędu opisanego na stronie Zmiana katalogu danych działającej instancji

Katalog `base/` nie jest jedynym miejscem na dane tabel — jeśli tabela albo indeks zostały utworzone w niestandardowym tablespace, ich pliki leżą fizycznie poza `PGDATA`, a `pg_tblspc/` trzyma tylko dowiązanie do tej lokalizacji.

Porty i gniazda

Domyślny port TCP to `5432`. Drugi klaster na tym samym serwerze potrzebuje innego portu, ustawionego parametrem `port` w jego własnym `postgresql.conf`. Poza portem TCP każda instancja domyślnie nasłuchuje też na gnieździe uniksowym (Unix domain socket), zwykle w `/var/run/postgresql` albo `/tmp` — połączenie po gnieździe pomija stos sieciowy, jest szybsze i jest

tym, czego używa `sudo -u postgres psql` bez podanego hosta.

Wersjonowanie PostgreSQL

> **Pojęcie: wersja główna i wersja poprawkowa.** Do wersji 9.6 włącznie numer wersji głównej obejmował dwie pierwsze cyfry — `9.5` i `9.6` to dwie różne wersje główne, `9.6.1` i `9.6.24` to ta sama wersja główna z różnymi poprawkami. Od wersji 10 PostgreSQL zmienił schemat numeracji — wersją główną jest sama pierwsza liczba (`10`, `14`, `15`), a druga liczba (`15.5`) to wersja poprawkowa.

Różnica między nimi jest praktyczna, nie kosmetyczna. Aktualizacja w obrębie tej samej wersji głównej, na przykład z `15.4` na `15.5`, to podmiana binarek silnika — format plików w `PGDATA` się nie zmienia, instancja startuje na nowych binarkach bez żadnej migracji danych. Zmiana wersji głównej, na przykład z `14` na `15`, zwykle zmienia wewnętrzny format katalogu systemowego i plików danych — stary `PGDATA` nie da się uruchomić nowszymi binarkami wprost, potrzebna jest migracja: `pg_upgrade`, dump i restore, albo replikacja logiczna na czas przejścia. Ten temat rozwija strona Upgrade silnika PostgreSQL.

Częste błędy

Mylenie klastra PostgreSQL z klastrem wysokiej dostępności. Rozmowa o „klastrze PostgreSQL" w kontekście replikacji i failover to inne znaczenie tego słowa niż klaster danych opisany wyżej — jedna instancja PostgreSQL to jeden klaster danych, niezależnie od tego, czy w ogóle istnieje jakakolwiek replikacja.

Traktowanie plików w PGDATA jako bezpiecznych do ręcznej edycji. Poza `postgresql.conf`, `pg_hba.conf` i `pg_ident.conf` reszta zawartości `PGDATA` jest wewnętrznym formatem silnika — ręczna ingerencja w pliki w `base/` kończy się uszkodzoną bazą, nie do naprawienia bez backupu.

Powiązane: Baza danych, silnik, instancja, schemat: cztery pojęcia, które się myli · Instalacja PostgreSQL z własnym katalogiem danych · Zmiana katalogu danych działającej instancji · Log transakcyjny: WAL, redo, transaction log · VACUUM, autovacuum i bloat

Źródła: dokumentacja PostgreSQL, rozdział „Database File Layout" i „Server Processes"