Użytkownicy i uprawnienia

Role, uprawnienia i konta techniczne dla backupu oraz monitoringu.

- Użytkownicy, role i uprawnienia
- Uwierzytelnianie i bezpieczeństwo połączeń
- Logowanie do PostgreSQL kontem z Active Directory: LDAP, LDAPS i grupy

Użytkownicy, role i uprawnienia

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

Konto Zabbixa do monitoringu, konto do backupu przez replikację i konto aplikacji do jednego schematu — trzy różne role, trzy różne zestawy uprawnień, ten sam mechanizm `CREATE ROLE` i `GRANT` pod spodem.

Fiszka

ZMIENIA SYSTEM

Konfiguracja usera zbx_monitor dla Zabbixa: PostgreSQL 10+

```
CREATE USER zbx_monitor WITH PASSWORD 'HasloUzytkownika' INHERIT;
```

Tworzymy dedykowane konto monitorujące dla Zabbixa — samo `CREATE USER` jeszcze nie daje mu żadnych uprawnień poza domyślnymi.

Przykład:

```
CREATE ROLE
```

Nadajemy rolę wbudowaną `pg_monitor`, dostępną od wersji 10 wzwyż:

```
GRANT pg_monitor TO zbx_monitor;
```

Przykład:

```
GRANT ROLE
```

Doprecyzowujemy dodatkowo odczyt widoku statystyk bazodanowych, jeśli konkretny szablon monitoringu tego wymaga:

```
GRANT SELECT ON pg_stat_database TO zbx_monitor;
```

Przykład:

```
GRANT
```

Na koniec dopisujemy w `pg_hba.conf`, opisanym na stronie postgresql.conf i pg_hba.conf, regułę pozwalającą temu kontu się zalogować z hosta, na którym stoi serwer Zabbixa:

```
host    all             zbx_monitor     10.20.10.0/24   scram-sha-256
```

Po dopisaniu reguły przeładowujemy konfigurację:

```
SELECT pg_reload_conf();
```

◫◫ ZMIENIA SYSTEM

Konfiguracja usera zbx_monitor dla Zabbixa: PostgreSQL starszy niż 10

```
CREATE USER zbx_monitor WITH PASSWORD 'HasloUzytkownika' INHERIT;
```

Na wersjach sprzed 10, gdzie rola `pg_monitor` jeszcze nie istnieje, jedynym sposobem na dostęp do pełnego zakresu widoków statystyk jest nadanie kontu monitorującemu pełnych uprawnień superusera:

```
ALTER USER zbx_monitor WITH SUPERUSER;
```

Przykład:

```
ALTER ROLE
```

Uważaj: to rozwiązanie zostaje na stałe kontem o pełnych uprawnieniach administracyjnych — przy migracji instancji na wersję 10 lub nowszą warto od razu zamienić je na węższy zestaw z `pg_monitor`, zamiast przenosić `SUPERUSER` dalej bez potrzeby.

◫◫ ZMIENIA SYSTEM

Dodanie usera backupu z prawem replikacji

```
CREATE USER base_backup_user WITH REPLICATION LOGIN;
```

Tworzymy dedykowane konto do backupu fizycznego, wykorzystywane przez narzędzia takie jak `pg_basebackup` albo pgBackRest: podstawowe komendy — atrybut `REPLICATION` daje prawo nawiązania strumienia replikacyjnego bez nadawania jakichkolwiek uprawnień do samych danych w tabelach.

Przykład:

```
CREATE ROLE
```

Dopisujemy w `pg_hba.conf` regułę zezwalającą temu kontu na połączenia replikacyjne:

```
host    replication     base_backup_user    127.0.0.1/32    scram-sha-256
```

Sprawdzamy, że rola rzeczywiście ma atrybut replikacji:

```
\du+ base_backup_user
```

Przykład:

```
                                   List of roles
    Role name     |            Attributes             | Member of | Description
------------------+-----------------------------------+-----------+-------------
 base_backup_user  | Replication                       | {}        |
(1 row)
```

□□ ZMIENIA SYSTEM

Nadanie uprawnień do schematu i tabel: GRANT

```
GRANT USAGE ON SCHEMA NazwaSchematu TO NazwaUsera;
```

Samo `USAGE` na schemacie nie daje dostępu do żadnej tabeli w środku — pozwala jedynie „wejść" do schematu, czyli odwoływać się do jego obiektów po nazwie. Bez tego uprawnienia rola nie widzi tabel schematu nawet wtedy, gdy ma na nie nadane `SELECT`.

Przykład:

```
GRANT
```

Nadanie odczytu i zapisu na konkretnej tabeli:

```
GRANT SELECT, INSERT, UPDATE ON NazwaSchematu.NazwaTabeli TO NazwaUsera;
```

Przykład:

```
GRANT
```

Nadanie tego samego zestawu na wszystkich tabelach istniejących już w schemacie, jednym poleceniem:

```
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA NazwaSchematu TO NazwaUsera;
```

Przykład:

```
GRANT
```

◫◫ ZMIENIA SYSTEM

Uprawnienia domyślne dla przyszłych obiektów: ALTER DEFAULT PRIVILEGES

> **Pojęcie: ALTER DEFAULT PRIVILEGES.** `GRANT ... ON ALL TABLES` z poprzedniego przykładu obejmuje wyłącznie tabele istniejące w chwili wykonania polecenia — każda nowa tabela utworzona później w tym samym schemacie znowu nie ma żadnych uprawnień dla naszej roli, dopóki ktoś nie nada ich ręcznie. `ALTER DEFAULT PRIVILEGES` rozwiązuje to na przyszłość, definiując uprawnienia, jakie mają być nadawane automatycznie każdej nowo tworzonej tabeli w danym schemacie.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA NazwaSchematu GRANT SELECT ON TABLES TO NazwaUsera;
```

Przykład:

```
ALTER DEFAULT PRIVILEGES
```

Uważaj: reguła obowiązuje tylko dla obiektów tworzonych przez tę samą rolę, która wykonała `ALTER DEFAULT PRIVILEGES` — jeśli tabele w schemacie tworzy inne konto, regułę trzeba ustawić z klauzulą `FOR ROLE NazwaTworcy`.

◫ TYLKO ODCZYT

Sprawdzenie nadanych uprawnień: information_schema

```
SELECT grantee, table_schema, table_name, privilege_type
FROM information_schema.role_table_grants
WHERE grantee = 'NazwaUsera';
```

Sprawdzamy, jakie uprawnienia konkretna rola faktycznie ma nadane na poszczególnych tabelach — przydatne przy audycie dostępu albo przy diagnozowaniu, dlaczego aplikacja dostaje `permission denied`.

Przykład:

```
    grantee    | table_schema | table_name  | privilege_type
---------------+--------------+-------------+----------------
 nazwa_usera   | sprzedaz     | zamowienia  | SELECT
 nazwa_usera   | sprzedaz     | zamowienia  | INSERT
 nazwa_usera   | sprzedaz     | zamowienia  | UPDATE
(3 rows)
```

Role a użytkownicy

> **Pojęcie: rola.** W PostgreSQL nie ma osobnego pojęcia „użytkownik" i osobnego „grupa" tak, jak w niektórych innych silnikach — jest jeden obiekt, rola, a to, czy działa jak konto do logowania, czy jak grupa uprawnień, zależy wyłącznie od jej atrybutów. Rola z atrybutem `LOGIN` może się nią zalogować klient — to jest to, co potocznie nazywamy użytkownikiem. Rola bez `LOGIN` służy wyłącznie jako kontener uprawnień, do którego przypisujemy inne role przez `GRANT NazwaRoli TO NazwaUsera` — to jest odpowiednik grupy.

`CREATE USER` i `CREATE ROLE` tworzą dokładnie ten sam obiekt — jedyna różnica to domyślne atrybuty: `CREATE USER` dokłada `LOGIN` automatycznie, `CREATE ROLE` nie. Dlatego `psql` po obu poleceniach zwraca ten sam komunikat, `CREATE ROLE`, co widać na stronie Podstawowe operacje na bazie i użytkownikach.

CREATE ROLE z opcjami

```
CREATE ROLE NazwaUsera WITH LOGIN PASSWORD 'HasloUzytkownika' CONNECTION LIMIT 10;
```

Tworzymy rolę do logowania z hasłem i limitem jednoczesnych połączeń — `CONNECTION LIMIT` chroni instancję przed jednym kontem aplikacyjnym, które przez błąd w puli połączeń otwiera ich nieskończenie wiele.

Przykład:

```
CREATE ROLE
```

Najczęściej używane atrybuty roli, którymi doprecyzowujemy `CREATE ROLE`:

Atrybut	Znaczenie
`LOGIN` / `NOLOGIN`	czy rolą można się zalogować
`SUPERUSER` / `NOSUPERUSER`	pełne uprawnienia administracyjne w klastrze
`CREATEDB`	prawo tworzenia nowych baz danych
`CREATEROLE`	prawo tworzenia i modyfikowania innych ról
`REPLICATION`	prawo nawiązywania połączeń replikacyjnych
`INHERIT` / `NOINHERIT`	czy rola automatycznie dziedziczy uprawnienia ról, do których należy, bez `SET ROLE`
`CONNECTION LIMIT n`	maksymalna liczba jednoczesnych połączeń tej roli
`VALID UNTIL 'data'`	data wygaśnięcia konta

Role wbudowane

> **Pojęcie: rola wbudowana (predefined role).** Zestaw ról tworzonych automatycznie przez `initdb`, dających dostęp do konkretnego, wąskiego zakresu uprawnień administracyjnych bez konieczności nadawania pełnego `SUPERUSER`. Wprowadzone w PostgreSQL 10 właśnie po to, żeby konta monitorujące i serwisowe nie musiały być superuserami.

Rola wbudowana	Co daje
`pg_monitor`	odczyt wszystkich widoków i funkcji monitorujących — łączy w sobie trzy poniższe role
`pg_read_all_stats`	odczyt wszystkich widoków statystyk (`pg_stat_*`), łącznie z tymi wymagającymi normalnie uprawnień superusera

Rola wbudowana	Co daje
`pg_read_all_settings`	odczyt wszystkich parametrów konfiguracyjnych, także tych ukrytych dla zwykłych ról
`pg_signal_backend`	prawo wywołania `pg_cancel_backend` i `pg_terminate_backend` na cudzych sesjach, opisanych na stronie Sesje, długie zapytania, kasowanie kwerendy
`pg_read_server_files`	odczyt plików z dysku serwera przez funkcje takie jak `COPY FROM`
`pg_write_server_files`	zapis plików na dysku serwera
`pg_execute_server_program`	uruchamianie programów zewnętrznych z poziomu serwera

Uważaj: role wbudowane istnieją dopiero od PostgreSQL 10. Na starszych wersjach jedynym sposobem na nadanie kontu szerokiego dostępu do statystyk monitorujących bywało tymczasowe nadanie `SUPERUSER` — rozwiązanie gorsze z punktu widzenia bezpieczeństwa, ale jedyne dostępne na tamtych wersjach, co widać w przykładzie konfiguracji `zbx_monitor` niżej.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Konto `zbx_monitor` na pg01 od lat ma nadany `SUPERUSER`, bo Zabbixa konfigurowano jeszcze na starszej wersji PostgreSQL. Przy okazji przeglądu uprawnień zamieniamy je na rolę `pg_monitor`, węższą i wystarczającą do tego samego monitoringu. Zabbix dalej widzi wszystkie potrzebne statystyki, ale bez pełnych uprawnień administracyjnych na koncie serwisowym.

Częste błędy

Nadanie GRANT na tabele bez wcześniejszego USAGE na schemacie. Aplikacja dostaje `permission denied for schema`, mimo że uprawnienia na samej tabeli wyglądają poprawnie — brakującym ogniwem jest `USAGE` na schemacie nadrzędnym.

Zakładanie, że uprawnienia obejmą też przyszłe tabele. `GRANT ... ON ALL TABLES IN SCHEMA` jest migawką stanu w chwili wykonania — bez `ALTER DEFAULT PRIVILEGES` każda kolejna migracja dodająca nowe tabele wymaga ponownego, ręcznego `GRANT`.

Powiązane: Podstawowe operacje na bazie i użytkownikach · postgresql.conf i pg_hba.conf · pgBackRest: podstawowe komendy · Sesje, długie zapytania, kasowanie kwerendy

Źródła: dokumentacja PostgreSQL, rozdział „Database Roles" i „Predefined Roles"

Uwierzytelnianie i bezpieczeństwo połączeń

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

Strona postgresql.conf i pg_hba.conf pokazuje, czym `pg_hba.conf` jest i jak wygląda pojedyncza reguła — ta strona rozwija to, co z punktu widzenia bezpieczeństwa jest istotne: kolejność reguł, realną ocenę każdej metody uwierzytelniania, migrację na `scram-sha-256` i szyfrowanie samego połączenia.

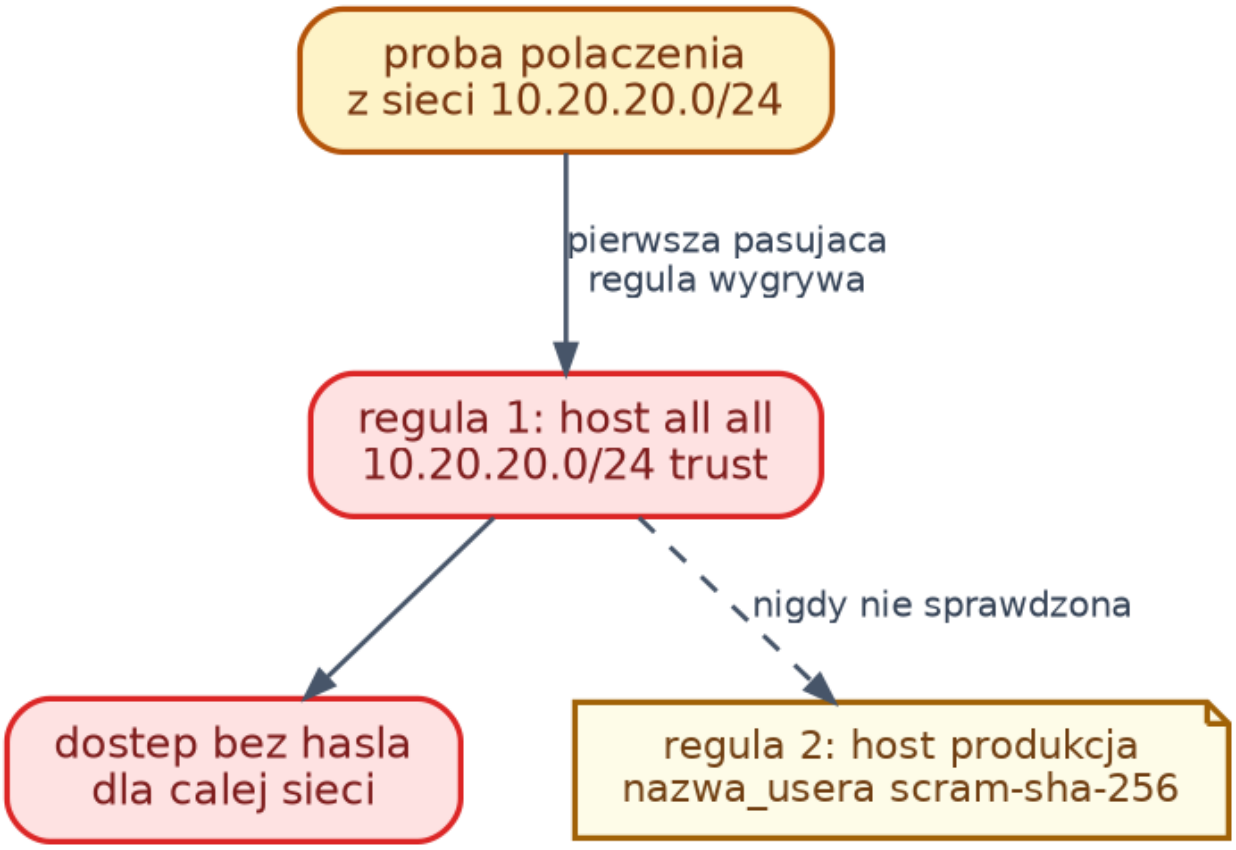
Skróty użyte na tej stronie (10)

Skrót	Co znaczy
AD	Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
JDBC	Java Database Connectivity: standardowy sposób, w jaki program w Javie łączy się z bazą danych
LDAPS	LDAP szyfrowany TLS, zwykle port 636
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
SSL	Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
TCP/IP	zestaw protokołów internetu: IP przenosi pakiety między adresami, TCP pilnuje, żeby dane doszły w całości i w kolejności
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Kolejność reguł decyduje

`pg_hba.conf` czytany jest od góry do dołu, a instancja zatrzymuje się na pierwszej regule pasującej do bazy, użytkownika, adresu i typu połączenia — reszta pliku, nawet jeśli zawiera regułę dopasowaną dokładniej, nigdy nie zostaje sprawdzona. To najczęstsze źródło błędnej konfiguracji: reguła szeroka, umieszczona wyżej niż reguła szczegółowa, przechwytuje połączenie, zanim ta

druga zdąży zadziałać.



Reguła trust dopasowana jako pierwsza przechwytuje połączenie, węższa reguła scram-sha-256 pod nią nigdy nie jest sprawdzana (kliknięcie otwiera pełny rozmiar)

Przykład błędnej kolejności — reguła w drugiej linii nigdy się nie wykona, bo pierwsza już dopasowała wszystko:

```
# TYPE  DATABASE        USER            ADDRESS         METHOD
host    all             all             10.20.20.0/24   trust
host    produkcja       nazwa_usera     10.20.20.0/24   scram-sha-256
```

Zamiana kolejności obu linii sprawia, że wąska reguła dostaje szansę zadziałać pierwsza, a szeroka `trust` zostaje regułą zamykającą, nie otwierającą.

Fiszka

Sprawdzenie, jak silnik faktycznie zinterpretował plik: pg_hba_file_rules

```
SELECT line_number, type, database, user_name, address, auth_method, error
FROM pg_hba_file_rules
ORDER BY line_number;
```

> **Pojęcie: pg_hba_file_rules.** Widok katalogowy, dostępny od PostgreSQL 10, pokazujący każdą regułę pliku tak, jak ją sparsował silnik, wraz z numerem linii i kolumną `error` — jeśli reguła ma błąd składniowy albo odwołuje się do nieistniejącej bazy czy roli, ten widok pokazuje to bez konieczności szukania komunikatu w logu serwera. Sprawdzamy go zawsze po edycji pliku, jeszcze przed przeładowaniem, żeby złapać literówkę na sucho.

Przykład:

```
 line_number | type  | database |   user_name   |    address    |  auth_method  | error
-------------+-------+----------+---------------+---------------+---------------+-------
           1 | local | {all}    | {postgres}    |               | peer          |
           2 | host  | {all}    | {zbx_monitor} | 10.20.10.0/24 | scram-sha-256 |
           3 | host  | {all}    | {all}         | 0.0.0.0/0     | reject        |
(3 rows)
```

Kolumna `error` pusta na każdym wierszu oznacza, że plik parsuje się bez zastrzeżeń — reguła z literówką w nazwie metody pojawiłaby się tu z opisem błędu, mimo że sam plik nadal wczytuje się bez wyjątku przerywającego start instancji.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Podczas prac serwisowych na pg01 dopisujemy tymczasową regułę `trust` dla całej sieci biura, żeby szybciej przetestować połączenie NetBoksa. Reguła trafia nad istniejącą regułą `scram-sha-256`, więc od tej chwili każdy z sieci biurowej loguje się bez hasła, a węższa reguła pod spodem nigdy się nie wykonuje. Pan Wojtek, audytor, znajduje to pół roku później i pyta, od kiedy tak jest.

Kolumny reguły: typ, baza, użytkownik, adres, metoda

Pierwsza kolumna każdej linii określa rodzaj połączenia, do którego reguła się odnosi, i to ona decyduje, czy kolumna adresu w ogóle ma znaczenie:

Typ	Połączenie	Kolumna adresu
`local`	gniazdo uniksowe, tylko ten sam host	pominięta, nie dotyczy
`host`	TCP, zaszyfrowane lub nie	wymagana, adres albo zakres CIDR
`hostssl`	wyłącznie TCP z wymuszonym TLS	wymagana
`hostnossl`	wyłącznie TCP bez TLS	wymagana

Uważaj: `hostnossl` bywa mylona z zapomnianym `hostssl`. Reguła `hostnossl` istnieje po to, żeby świadomie dopuścić słabsze, nieszyfrowane połączenie z konkretnej, zaufanej sieci — obecność takiej reguły w produkcyjnym pliku bez wyraźnego powodu jest sygnałem, że ktoś kopiował fragment konfiguracji bez zrozumienia, co robi.

Metody uwierzytelniania — kiedy właściwa, kiedy jest luką

> **Pojęcie: metoda uwierzytelniania.** Ostatnia kolumna reguły, decydująca, jak instancja potwierdza tożsamość klienta, który już przeszedł dopasowanie typu, bazy, użytkownika i adresu. Wybór metody nie jest kosmetyczny — dwie reguły identyczne poza tą jedną kolumną mogą różnić się między bezpiecznym dostępem a otwartą luką.

Metoda	Kiedy jest właściwa	Kiedy jest luką
`trust`	połączenie lokalne, na tym samym hoście, do którego dostęp systemowy już oznacza zaufanie	na regule pasującej do adresu szerszego niż pojedynczy host — wtedy każdy, kto nawiąże połączenie TCP, loguje się bez hasła
`peer`	połączenia lokalne przez gniazdo uniksowe, gdzie konto systemowe i rola bazodanowa mają tę samą nazwę	próba użycia jej dla połączeń `host` — `peer` działa wyłącznie dla `local`, silnik odrzuci taką regułę

Metoda	Kiedy jest właściwa	Kiedy jest luką
`ident`	połączenia TCP z sieci, w której działa zaufany, kontrolowany serwer ident	zaufanie odpowiedzi serwera ident z sieci, nad którą nie mamy kontroli — sam protokół ident nie jest niczym uwierzytelniony ani szyfrowany
`md5`	kompatybilność wsteczna z klientami starszymi niż PostgreSQL 10	jako domyślny wybór na nowej instalacji — hasz MD5 jest podatny na ataki słownikowe offline, jeśli ktoś przechwyci go z sieci lub z `pg_authid`
`scram-sha-256`	domyślny, zalecany wybór dla każdego połączenia hasłowego od PostgreSQL 10 wzwyż	brak — jedyna sytuacja graniczna to klient ze starą biblioteką `libpq` (przed 10), która tej metody nie obsługuje
`cert`	połączenia, w których klient przedstawia certyfikat X.509 podpisany przez zaufany urząd, zwykle w środowiskach z pełnym PKI wewnętrznym	wystawienie certyfikatu bez odpowiadającego mu ograniczenia w `pg_hba.conf` do `hostssl` — `cert` bez wymuszenia TLS na tej samej linii nie ma sensu, bo weryfikacja certyfikatu jest częścią uzgadniania TLS
`ldap`	środowisko z centralnym katalogiem LDAP/AD, gdzie chcemy jednego miejsca zarządzania hasłami	użycie trybu prostego wiązania (`simple bind`) bez `ldapscheme=ldaps` — hasło leci do serwera LDAP jawnym tekstem po sieci
`reject`	reguła zamykająca plik, jawnie odrzucająca wszystko, co nie dopasowało się do żadnej wcześniejszej, bardziej szczegółowej reguły	nigdy nie jest luką sama w sobie — jej brak na końcu pliku bywa mylnie odczytywany jako otwarcie dostępu, choć domyślne zachowanie i tak odrzuca połączenie bez dopasowanej reguły

Migracja z md5 na scram-sha-256

> **Pojęcie: password_encryption.** Parametr `postgresql.conf` określający, jakim algorytmem silnik hašuje hasło w chwili jego ustawienia — `ALTER USER ... PASSWORD` czy `CREATE ROLE ... PASSWORD`. Zmiana tego parametru nie przehaszowuje istniejących haseł wstecz: konto, które miało hasło ustawione, gdy obowiązywał `md5`, tak zostaje, dopóki ktoś tego hasła nie zmieni ponownie.

Sprawdzenie bieżącego algorytmu kodowania haseł

```
SHOW password_encryption;
```

Sprawdzamy, jaki algorytm instancja stosuje dla nowo ustawianych haseł, zanim zaczniemy migrację.

Przykład:

```
 password_encryption
---------------------
 md5
(1 row)
```

Zmiana domyślnego algorytmu na scram-sha-256

```
ALTER SYSTEM SET password_encryption = 'scram-sha-256';
SELECT pg_reload_conf();
```

Ustawiamy `scram-sha-256` jako algorytm dla każdego hasła ustawianego od tej pory — samo to polecenie nie rusza żadnego już istniejącego hasła.

Przykład:

```
ALTER SYSTEM
 pg_reload_conf
-----------------
 t
(1 row)
```

Wymuszenie ponownego ustawienia hasła: ALTER USER

```
ALTER USER NazwaUsera PASSWORD 'NoweHaslo';
```

Dopiero ponowne ustawienie hasła przehaszowuje je nowym algorytmem — to jedyny sposób, żeby istniejące konto faktycznie przeszło na `scram-sha-256`, bez usuwania i tworzenia roli od nowa.

Przykład:

```
ALTER ROLE
```

Weryfikujemy, że hasz w katalogu zaczyna się od `SCRAM-SHA-256$`, a nie od `md5`:

```
SELECT rolname, left(rolpassword, 14) FROM pg_authid WHERE rolname = 'NazwaUsera';
```

Przykład:

```
   rolname   |      left
-------------+----------------
 nazwa_usera  | SCRAM-SHA-256$
(1 row)
```

Ostatnim krokiem migracji jest podmiana metody `md5` na `scram-sha-256` w samych regułach `pg_hba.conf` i przeładowanie konfiguracji, opisane niżej — dopóki reguła wciąż mówi `md5`, silnik zaakceptuje też stare, nieprzehaszowane jeszcze konta, ale nowy hasz nadal działa poprawnie z obiema metodami do czasu pełnego przełączenia.

Uważaj: migracja przeprowadzona wyłącznie zmianą `password_encryption`, bez wymuszenia zmiany hasła na każdym koncie, zostawia stare konta na MD5 bezterminowo — użytkownicy nie mają powodu zmieniać działającego hasła, jeśli nikt im tego nie zakomunikuje.

SSL — szyfrowanie połączenia

> **Pojęcie: certyfikat serwera.** Para plików — klucz prywatny i certyfikat publiczny — którymi instancja dowodzi klientowi swojej tożsamości podczas nawiązywania połączenia szyfrowanego, dokładnie tak samo jak robi to serwer WWW przy HTTPS. Bez tej pary `ssl = on` nie ma czego użyć do uzgodnienia szyfrowania.

Włączenie SSL po stronie serwera

```
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
```

Wskazujemy plik certyfikatu i klucza prywatnego, oba domyślnie szukane w katalogu `PGDATA`, jeśli ścieżka nie jest podana bezwzględnie. Parametr `ssl` wymaga restartu instancji, nie samego przeładowania.

Sprawdzenie, czy SSL jest włączony po stronie serwera

```
SHOW ssl;
```

Przykład:

```
 ssl
-----
 on
(1 row)
```

Certyfikat wystawiony przez zewnętrzny urząd trafia do `ssl_ca_file`, jeśli chcemy dodatkowo weryfikować certyfikaty klienckie metodą `cert` opisaną wyżej.

sslmode po stronie klienta

> **Pojęcie: sslmode.** Parametr połączenia klienta (`psql`, `libpq`, sterowniki JDBC/ODBC), niezależny od ustawień serwera, decydujący, jak bardzo klient nalega na szyfrowanie i weryfikację tożsamości serwera. Serwer może mieć `ssl = on`, ale to klient, przez `sslmode`, decyduje, czy z tego skorzysta.

sslmode	Szyfrowanie	Weryfikacja certyfikatu serwera
`disable`	nigdy	nie dotyczy
`allow`	tylko jeśli serwer wymusza TLS, w innym wypadku bez szyfrowania	nie
`prefer` (domyślny)	próbuje TLS jako pierwsze, wraca do zwykłego połączenia, jeśli się nie uda	nie
`require`	zawsze, połączenie odrzucone bez TLS	nie — podatne na atak pośrednika podszywającego się pod serwer
`verify-ca`	zawsze	tak, certyfikat musi być podpisany przez zaufany urząd
`verify-full`	zawsze	tak, certyfikat i dodatkowo nazwa hosta w certyfikacie muszą się zgadzać z adresem, do którego się łączymy

Uważaj: `require` chroni przed podsłuchaniem transmisji, ale nie przed podszyciem się pod serwer, bo nie sprawdza, czyj to certyfikat — dopiero `verify-ca` i `verify-full` dają pełną gwarancję, że łączymy się z właściwym serwerem, a nie z kimś pomiędzy.

Wymuszenie połączenia szyfrowanego z klienta: psql sslmode

```
psql "host=pg01 dbname=produkcja user=nazwa_usera sslmode=verify-full"
```

Wymuszamy najsilniejszy wariant weryfikacji — połączenie zakończy się błędem zamiast po cichu spaść do zwykłego TCP, jeśli certyfikat nie pasuje albo nazwa hosta się nie zgadza.

Sprawdzenie, czy bieżące połączenie jest szyfrowane

```
\conninfo
```

Metakomenda `psql`, opisana szerzej na stronie psql: klient wiersza poleceń, pokazuje w jednej linii, czy aktualna sesja korzysta z TLS.

Przykład:

```
You are connected to database "produkcja" as user "nazwa_usera" on host "pg01" at port "5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
```

Równoważna informacja z poziomu SQL, przydatna do sprawdzenia szyfrowania cudzej, a nie własnej sesji:

```
SELECT pid, ssl, client_addr, cipher FROM pg_stat_ssl WHERE pid = pg_backend_pid();
```

Przykład:

```
  pid  | ssl | client_addr |         cipher
-------+-----+-------------+------------------------
 24501 | t   | 10.20.20.20 | TLS_AES_256_GCM_SHA384
(1 row)
```

Kolumna `ssl` na `f` przy połączeniu, które powinno być szyfrowane, oznacza, że reguła `pg_hba.conf` dopuściła je jako zwykłe `host`, a nie `hostssl`.

Przeładowanie reguł bez restartu

Zmiana w `pg_hba.conf`, niezależnie czy dotyczy kolejności, kolumn czy metody, wymaga wyłącznie przeładowania, nigdy restartu — mechanizm opisuje dokładnie strona postgresql.conf i pg_hba.conf, fiszką `pg_reload_conf()`. Istniejące, już nawiązane połączenia nie są tym przeładowaniem w żaden sposób dotknięte — reguły są sprawdzane wyłącznie w chwili nawiązywania nowego połączenia.

Częste błędy

Zostawienie reguły `hostnossl` albo `trust` po pracach serwisowych. Reguła otwarta tymczasowo na czas migracji czy awarii, o czym przypomina strona postgresql.conf i pg_hba.conf, zapomniana w produkcji jest realną luką, nie kosmetycznym niedopatrzeniem.

Zmiana password_encryption bez wymuszenia zmiany haseł. Parametr działa tylko na przyszłość — konta z hasłem ustawionym wcześniej zostają na starym algorytmie bezterminowo, dopóki nikt nie każe im hasła zmienić.

Wybór ldap bez wymuszenia ldaps. Prosty tryb wiązania bez szyfrowania transportu wysyła hasło do serwera katalogowego jawnym tekstem, mimo że samo połączenie z bazą może być szyfrowane TLS-em. Pełną konfigurację z AD Zielonej (search+bind, LDAPS, konto techniczne, grupy i dostęp czasowy) opisuje strona Logowanie do PostgreSQL kontem z Active Directory: LDAP, LDAPS i grupy.

Powiązane: postgresql.conf i pg_hba.conf · Użytkownicy, role i uprawnienia · psql: klient wiersza poleceń · Logi serwera: konfiguracja i czytanie

Źródła: dokumentacja PostgreSQL, rozdział „Client Authentication" i „Secure TCP/IP Connections with SSL"

Logowanie do PostgreSQL kontem z Active Directory: LDAP, LDAPS i grupy

Poziom: średnio **Dotyczy: PostgreSQL 16 na pg01, Active Directory ad.zielona.example**

Czas: ~35 min

PostgreSQL potrafi sprawdzić hasło nie u siebie, tylko w Active Directory, więc programista loguje się do bazy tym samym hasłem co do komputera. Strona pokazuje oba tryby metody `ldap` (simple bind i search+bind), szyfrowanie LDAPS, konto techniczne, zaufanie do certyfikatu `CA-Zielona`, związek grup z AD z rolami w bazie i dostęp czasowy przez członkostwo w grupie. Wracamy tu przy każdej nowej bazie, do której ma się logować ktoś z zespołu Oli.

ZAŁOŻENIA SCENARIUSZA

kontrolery `dc01` i `dc02` mają certyfikat LDAPS wystawiony przez urząd `CA-Zielona`, a `pg01` ma włączone `ssl = on` z własnym certyfikatem, tak jak opisuje strona Uwierzytelnianie i bezpieczeństwo połączeń. W lesie `ad.zielona.example` działa funkcja Privileged Access Management (decyzja autora z 11 października 2026), więc członkostwo w grupie może mieć czas życia. Programiści łączą się z sieci biura 10.20.20.0/24.

Po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona dokładamy tu w ten sam sposób zaufanie do `Zielona-Root-CA`, a `CA-Zielona.crt` zostaje obok do wygaszenia starego urzędu.

Komendy `ldapsearch`, `openssl` i `psql` wykonujemy w CLI bash na `pg01` jako użytkownik z uprawnieniami sudo, komendy SQL w `psql` kontem `postgres`, a komendę PowerShell na stacji administratora z modułem ActiveDirectory.

Skróty użyte na tej stronie (12)

Skrót	Co znaczy

AD	Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DN	Distinguished Name: pełny, jednoznaczny adres jednego obiektu w katalogu LDAP, na przykład konta albo grupy
GLPI	Gestionnaire Libre de Parc Informatique: otwarty system do zgłoszeń, inwentarza i bazy wiedzy IT; w Zielonej działa w kontenerze na docker01
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
LDAP	Lightweight Directory Access Protocol: protokół do odpytywania katalogu użytkowników (AD, FreeIPA)
LDAPS	LDAP szyfrowany TLS, zwykle port 636
PAM	Pluggable Authentication Modules: wymienne moduły logowania w Linuksie, decydują na przykład o sile wymaganego hasła; w Active Directory także Privileged Access Management: funkcja lasu pozwalająca nadawać członkostwo w grupie na określony czas
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
SSL	Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS
StartTLS	Start Transport Layer Security: polecenie, które zamienia zwykłe, nieszyfrowane połączenie (np. LDAP na porcie 389) w szyfrowane już po jego nawiązaniu
UPN	User Principal Name: login w formie adresu, np. jan.nowak@zielona.example

Fiszka

TYLKO ODCZYT

Sprawdzenie certyfikatu LDAPS kontrolera domeny: openssl s_client

```
openssl s_client -connect NazwaKontroleraDC:636 -CAfile /etc/pki/tls/certs/ca-bundle.crt
-verify_return_error </dev/null | grep "Verify return code"
```

> (gdzie `NazwaKontroleraDC` to pełna nazwa kontrolera, np. `dc01.ad.zielona.example`; nazwa musi być w certyfikacie, więc nie podajemy adresu IP)

Pierwszy test przed dotknięciem `pg_hba.conf`: czy port 636 odpowiada i czy `pg01` ufa certyfikatowi kontrolera. Wynik inny niż `0 (ok)` oznacza, że PostgreSQL też się nie połączy.

Przykład:

```
Verify return code: 0 (ok)
```

LDAPS = LDAP szyfrowany TLS, zwykle port 636 · **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

ZMIENIA SYSTEM

Dodanie certyfikatu CA-Zielona do zaufanych na pg01: update-ca-trust

```
sudo cp CA-Zielona.crt /etc/pki/ca-trust/source/anchors/
sudo update-ca-trust
```

Rocky Linux 9 dokłada certyfikat urzędu do wspólnego pliku `/etc/pki/tls/certs/ca-bundle.crt`, z którego korzystają `openssl`, `ldapsearch` i PostgreSQL. Certyfikat `CA-Zielona.crt` eksportujemy z `dc01` w formacie Base64, tak jak przy innych aplikacjach z LDAPS.

LDAPS = LDAP szyfrowany TLS, zwykle port 636 · **cp** = copy: skopiuj · **update-ca-trust** = update CA trust: odświeżenie systemowego magazynu zaufanych urzędów certyfikacji w Rocky Linux

TYLKO ODCZYT

Wskazanie pliku urzędów dla biblioteki OpenLDAP: /etc/openldap/ldap.conf

```
TLS_CACERT      /etc/pki/tls/certs/ca-bundle.crt
TLS_REQCERT     demand
```

PostgreSQL rozmawia z AD przez bibliotekę OpenLDAP, a ta czyta zaufane urzędy z `TLS_CACERT` [4]. `TLS_REQCERT demand` to wartość domyślna (połączenie bez poprawnego certyfikatu jest przerywane), wpisujemy ją jawnie, żeby nikt nie zmienił jej na `never` „na chwilę”.

AD = Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie

ZMIENIA SYSTEM

Utworzenie konta technicznego do wyszukiwania: New-ADUser

```
New-ADUser -Name "svc_pg01_ldap" -SamAccountName "svc_pg01_ldap" -Path
"OU=KontaTechniczne,DC=ad,DC=zielona,DC=example" -AccountPassword (Read-Host -
AsSecureString "Haslo") -Enabled $true -CannotChangePassword $true -Description
"PostgreSQL pg01: wyszukiwanie kont, tylko odczyt"
```

Konto zwykłego użytkownika domeny bez żadnych dodatkowych grup: domyślne prawo odczytu katalogu wystarcza do wyszukania konta i jego grup. Hasło od razu trafia do Vaultwardena, bo za chwilę wpiszemy je do `pg_hba.conf`.

OU = Organizational Unit: folder w Active Directory grupujący konta i komputery · **DC** = Domain Controller: serwer, który trzyma Active Directory i loguje użytkowników

TYLKO ODCZYT

Wyszukanie konta tak, jak zrobi to PostgreSQL: ldapsearch

```
ldapsearch -H ldaps://NazwaKontroleraDC -D
"CN=svc_pg01_ldap,OU=KontaTechniczne,DC=ad,DC=zielona,DC=example" -W -b
"DC=ad,DC=zielona,DC=example" "(&(sAMAccountName=nazwa_usera)(memberOf=DNGrupy))" dn
```

> (gdzie `nazwa_usera` to login domenowy, a `DNGrupy` to pełna nazwa grupy, np. `CN=GG_Baza_Cennik_RO,OU=Grupy,DC=ad,DC=zielona,DC=example`)

To ten sam bind i ten sam filtr, który wpiszemy do `pg_hba.conf`. Dokładnie jeden wpis `dn:` oznacza, że PostgreSQL znajdzie użytkownika; zero wpisów oznacza, że osoby nie ma w grupie albo filtr ma literówkę.

Przykład (fragment):

```
dn: CN=Ola Wisniewska,OU=Uzytkownicy,DC=ad,DC=zielona,DC=example
# numEntries: 1
```

CN = Common Name: główna nazwa w certyfikacie, np. adres serwera · **OU** = Organizational Unit: folder w Active Directory grupujący konta i komputery · **DC** = Domain Controller: serwer, który trzyma Active Directory i loguje użytkowników · **ldapsearch** = LDAP search: wyszukuje wpisy w katalogu LDAP (np. w Active Directory) według filtra, tu do sprawdzenia, czy konto przechodzi filtr z pg_hba.conf

TYLKO ODCZYT

Reguła search+bind przez LDAPS: pg_hba.conf

```
hostssl  cennik  +cennik_ro  10.20.20.0/24  ldap \
    ldapserver="dc01.ad.zielona.example dc02.ad.zielona.example" ldapscheme=ldaps \
    ldapbasedn="DC=ad,DC=zielona,DC=example" \
    ldapbinddn="CN=svc_pg01_ldap,OU=KontaTechniczne,DC=ad,DC=zielona,DC=example" \
    ldapbindpasswd="HasloKontaTechnicznego" \

ldapsearchfilter="(&(sAMAccountName=$username)(memberOf=CN=GG_Baza_Cennik_RO,OU=Grupy,DC
=ad,DC=zielona,DC=example))"
```

Jedna reguła w pięciu liniach: PostgreSQL 16 łączy linie zakończone ukośnikiem wstecznym [1]. `hostssl` wymusza szyfrowanie między klientem a `pg01`, `ldapscheme=ldaps` między `pg01` a kontrolerem, a `+cennik_ro` ogranicza regułę do ról należących do roli grupowej `cennik_ro`. Regułę wstawiamy nad szerszymi regułami dla tej sieci.

LDAPS = LDAP szyfrowany TLS, zwykle port 636

ZMIENIA SYSTEM

Utworzenie roli grupowej i roli dla konta domenowego: CREATE ROLE

```
CREATE ROLE cennik_ro NOLOGIN;
GRANT CONNECT ON DATABASE cennik TO cennik_ro;
GRANT USAGE ON SCHEMA public TO cennik_ro;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO cennik_ro;
CREATE ROLE "nazwa_usera" LOGIN IN ROLE cennik_ro;
```

> (gdzie `nazwa_usera` to login domenowy małymi literami, w cudzysłowie, bo zawiera kropkę, np. `"ola.wisniewska"`)

Rola konta domenowego nie ma hasła, bo hasło sprawdza AD. Prawa dostaje przez rolę grupową, więc przy nowej osobie dodajemy tylko jedną rolę, a nie powtarzamy `GRANT`.

AD = Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie

TYLKO ODCZYT

Sprawdzenie, jak silnik odczytał regułę: pg_hba_file_rules

```
SELECT line_number, type, database, user_name, auth_method, error
FROM pg_hba_file_rules
WHERE auth_method = 'ldap' OR error IS NOT NULL;
```

Sprawdzamy przed przeładowaniem: pusta kolumna `error` przy regule `ldap` znaczy, że składnia i nazwy opcji są poprawne. Widok domyślnie czyta tylko superużytkownik [1].

Przykład:

```
 line_number |  type   | database |  user_name   | auth_method | error
-------------+---------+----------+--------------+-------------+-------
          12 | hostssl | {cennik} | {+cennik_ro} | ldap        |
(1 row)
```

□□ TYLKO ODCZYT

Przeładowanie reguł: pg_reload_conf()

```
SELECT pg_reload_conf();
```

Reguły z `pg_hba.conf` wchodzą w życie po przeładowaniu, bez restartu; otwarte sesje działają dalej po staremu.

□□ TYLKO ODCZYT

Test logowania kontem domenowym: psql

```
psql "host=pg01.ad.zielona.example dbname=cennik user=nazwa_usera sslmode=verify-full"
```

Pyta o hasło domenowe. Udane logowanie potwierdza cały łańcuch: regułę, LDAPS, konto techniczne, filtr z grupą i rolę w bazie.

□□ **LDAPS** = LDAP szyfrowany TLS, zwykle port 636 · **psql** = PostgreSQL SQL: klient bazy PostgreSQL

□□ TYLKO ODCZYT

Zakończenie sesji po końcu dostępu czasowego: pg_terminate_backend

```
SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE usename = 'nazwa_usera' AND datname = 'cennik';
```

PostgreSQL sprawdza grupę tylko przy logowaniu, więc sesja otwarta przed wygaśnięciem członkostwa trwa do rozłączenia. W dniu końca dostępu kończymy ją sami, jeśli wciąż wisi.

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 9:40. Ola dostała w poniedziałek dostęp do bazy `cennik` na tydzień, a dziś `psql` odpowiada jej `FATAL: LDAP authentication failed for user "ola.wisniewska"`. W logu `pg01` stoi `LDAP user "ola.wisniewska" does not exist`, choć konto w AD działa. Tomek wkleja filtr z `pg_hba.conf` do `ldapsearch` i dostaje zero wpisów: we wtorek ktoś przeniósł grupę `GG_Baza_Cennik_RO` do innej jednostki, więc jej pełna nazwa w filtrze przestała pasować. Po poprawce DN w regule i przeładowaniu Ola loguje się o 9:55, a do standardu trafia zasada, że grup użytych w filtrach baz nie przenosimy bez zmiany w GLPI.

Jak to działa

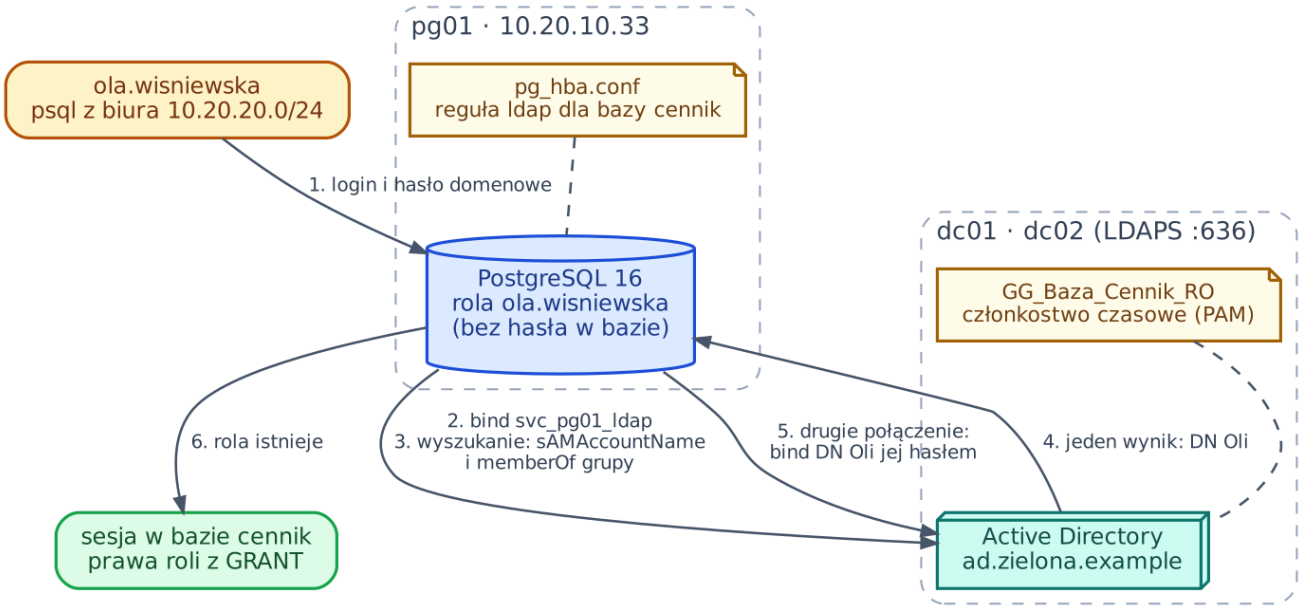
Rozwiń: jak to działa pod spodem

Hasło sprawdza AD, rola zostaje w bazie

Pojęcie: metoda ldap. Metoda uwierzytelniania w `pg_hba.conf`, która działa jak zwykłe hasło, tylko parę login i hasło sprawdza serwer katalogu, a nie PostgreSQL [2]. LDAP służy wyłącznie do sprawdzenia hasła, więc rola o tej samej nazwie musi już istnieć w bazie.

> **Pojęcie: bind.** Zalogowanie się do serwera LDAP: klient podaje pełną nazwę obiektu (DN) i hasło, a serwer odpowiada, czy pasują.

Klient wysyła hasło do PostgreSQL otwartym tekstem, tak jak w metodzie `password`. Dlatego reguła stoi na `hostssl`, a nie `host`: `ldapscheme` szyfruje tylko odcinek od `pg01` do kontrolera, a odcinek od klienta do `pg01` szyfruje wyłącznie SSL samego PostgreSQL [2].



Rysunek 1. Logowanie w trybie search+bind: Ola podaje hasło, pg01 wiązaniem konta svc_pg01_ldap wyszukuje jej konto z filtrem na grupę, a potem drugim połączeniem sprawdza jej hasło w AD (kliknięcie otwiera pełny rozmiar)

Dwa tryby: simple bind i search+bind

	Simple bind	Search+bind
Co robi PostgreSQL	składa DN z przedrostka, loginu i przyrostka i od razu wiąże się jako użytkownik	wiąże się kontem technicznym (albo anonimowo), wyszukuje użytkownika, potem wiąże się jako znaleziony użytkownik jego hasłem
Opcje	`ldapprefix`, `ldapsuffix`	`ldapbasedn`, `ldapbinddn`, `ldapbindpasswd`, `ldapsearchattribute` albo `ldapsearchfilter`

	Simple bind	Search+bind
Konto techniczne	niepotrzebne	potrzebne, bo AD nie pozwala na wyszukiwanie anonimowe
Sprawdzenie grupy	niemożliwe: AD sprawdza tylko hasło	filtr z `memberOf`
Gdy konto jest w innej jednostce	działa, jeśli DN da się złożyć z loginu	działa, wyszukiwanie obejmuje całe poddrzewo

Opcje wspólne dla obu trybów: `ldapserver` (kilka nazw rozdzielonych spacją), `ldapport` (bez niego port domyślny biblioteki), `ldapscheme=ldaps`, `ldaptls=1` (StartTLS) i `ldapurl`. Mieszanie opcji simple bind z opcjami search+bind w jednej regule jest błędem [2].

W AD simple bind działa z loginem w postaci UPN, gdy UPN kont kończy się na `@ad.zielona.example` (sprawdzamy w AD, zanim tak skonfigurujemy):

```
hostssl  NazwaBazy  all  10.20.20.0/24  ldap ldapserver=dc01.ad.zielona.example
ldapscheme=ldaps ldapsuffix="@ad.zielona.example"
```

Tak zalogować się może każdy pracownik, który ma rolę w bazie, bez względu na grupy. W Zielonej używamy search+bind, bo cały pomysł dostępu na wniosek opiera się na grupie.

> **Pojęcie: StartTLS.** Polecenie, które zamienia zwykłe połączenie LDAP na porcie 389 w szyfrowane, już po jego nawiązaniu. LDAPS na porcie 636 szyfruje od pierwszego bajtu. PostgreSQL zna oba: `ldaptls=1` albo `ldapscheme=ldaps`; dokumentacja nazywa LDAPS sposobem niestandardowym, ale AD go obsługuje i reszta bazy (Gitea, Jenkins, Zabbix) też łączy się przez 636 [2][5].

Certyfikat i zaufanie

Kontroler domeny wystawia LDAPS tylko wtedy, gdy ma certyfikat z rozszerzeniem Server Authentication, z pełną nazwą kontrolera w polu podmiotu albo w nazwach alternatywnych, wydany przez urząd, któremu ufają kontroler i klienci [5]. Stąd dwie zasady: w `ldapserver` stoją pełne nazwy `dc01.ad.zielona.example`, a nie adresy IP, i `pg01` musi ufać `CA-Zielona`.

PostgreSQL nie ma własnej opcji z plikiem urzędu dla LDAP. Zaufanie bierze z konfiguracji biblioteki OpenLDAP, czyli z `/etc/openldap/ldap.conf` [4]. Gdy `ldapsearch` z konsoli działa, a PostgreSQL nie, najpierw sprawdzamy, czy usługa `postgresql-16` nie ma w środowisku innego `LDAPCONF` albo `LDAPTLS_CACERT`.

Konto techniczne

`svc_pg01_ldap` służy tylko do wyszukiwania. Nie należy do żadnej grupy poza Domain Users, nie loguje się na komputery, a jego hasło stoi jawnym tekstem w `pg_hba.conf`. Dlatego:

- plik `pg_hba.conf` czyta tylko użytkownik `postgres` (sprawdzamy `ls -l` w katalogu danych),
- kopia katalogu danych w pgBackRest zawiera ten plik, więc repozytorium kopii jest tak samo tajne jak hasło,
- przy zmianie hasła konta podmieniamy je w regule i w Vaultwardenie w jednej zmianie, inaczej po wygaśnięciu starego hasła nikt nie zaloguje się do `cennik`.

Filtr z grupą i grupy zagnieżdżone

`ldapsearchfilter` to filtr LDAP, w którym `$username` zamienia się na login podany przez klienta; `ldapsearchattribute=sAMAccountName` byłby równoważny filtrowi `(sAMAccountName=$username)` [2]. Dokładamy do niego warunek `memberOf`, więc wyszukiwanie zwraca konto tylko członka grupy.

Zwykły warunek `memberOf=DN` sprawdza członkostwo bezpośrednie. Gdy do grupy dostępu dodajemy inne grupy, potrzebna jest reguła dopasowania `LDAP_MATCHING_RULE_IN_CHAIN` (identyfikator 1.2.840.113556.1.4.1941), która przechodzi cały łańcuch grup [6]:

```
ldapsearchfilter="(&(sAMAccountName=$username)(memberOf:1.2.840.113556.1.4.1941:=CN=GG_Baz
a_Cennik_RO,OU=Grupy,DC=ad,DC=zielona,DC=example))"
```

Microsoft ostrzega, że takie zapytania mocniej obciążają kontroler [6]. W Zielonej grupy dostępu do baz zawierają ludzi, nie inne grupy, więc zostajemy przy zwykłym `memberOf`.

Grupy w AD a role w bazie

PostgreSQL nie zakłada ról sam i nie synchronizuje ich z AD: grupa w AD decyduje, kto w ogóle przejdzie logowanie, a rola i jej `GRANT` w bazie decydują, co ta osoba zobaczy. Obie warstwy utrzymujemy razem.

W AD	W PostgreSQL	Kto zmienia
grupa `GG_Baza_Cennik_RO`	rola grupowa `cennik_ro` z `GRANT SELECT`	my, raz, przy nowej bazie
członkostwo osoby w grupie, czasowe albo stałe	rola `"ola.wisniewska"` z `LOGIN IN ROLE cennik_ro`, na stałe	członkostwo: dyżurny z wniosku KU-05; rola: przy pierwszym wniosku danej osoby

W AD	W PostgreSQL	Kto zmienia
grupa zapisu (np. `GG_Baza_Cennik_RW`)	osobna rola `cennik_rw`, nadawana `GRANT cennik_rw TO` na czas wniosku	dyżurny, z `REVOKE` w dniu końca

Zapis wymaga obu kroków, bo prawa w bazie wynikają z roli, a nie z grupy w AD. Filtr reguły musi wtedy przepuszczać członków obu grup, na przykład `(|(memberOf=DN grupy RO)(memberOf=DN grupy RW))` wewnątrz warunku z `sAMAccountName`.

Gdy osób i baz robi się dużo, ręczne zakładanie ról męczy. Istnieją narzędzia, które czytają grupy z LDAP i zakładają albo usuwają role według reguł, na przykład `pg-ldap-sync` (Ruby) i `ldap2pg` (Dalibo). Podajemy je informacyjnie: w Zielonej ich nie wdrażamy, a przed ewentualnym wdrożeniem sprawdzamy w ich dokumentacji obsługiwane wersje PostgreSQL.

Dostęp czasowy z Privileged Access Management

> **Pojęcie: Privileged Access Management (PAM) w AD.** Funkcja lasu Active Directory od poziomu Windows Server 2016, która pozwala dodać konto do grupy z czasem życia; po jego upływie członkostwo samo znika [3]. Skrót PAM ma też drugie znaczenie: Pluggable Authentication Modules w Linuksie, z którymi PostgreSQL ma osobną metodę `pam`. Na tej stronie PAM znaczy zawsze funkcję AD.

Dostęp „na chwilę” do `cennik` to członkostwo czasowe w `GG_Baza_Cennik_RO`, dodawane `Add-ADGroupMember -MemberTimeToLive` według strony Wnioski o usługę, katalog usług i onboarding: od nowego konta do odebrania dostępu. Usunięcie wygasłego członkostwa kontroler ocenia od razu, bez czekania na replikację [3], więc po terminie filtr `memberOf` przestaje przepuszczać tę osobę przy kolejnym logowaniu. Zachowanie w naszej domenie potwierdzamy w dniu końca tym samym `ldapsearch` co w Fiszce: zero wpisów znaczy, że dostęp wygasł.

Trzy rzeczy, których członkostwo czasowe nie załatwia:

1. Sesja otwarta przed terminem trwa do rozłączenia; w dniu końca sprawdzamy `pg_stat_activity` i w razie potrzeby kończymy ją `pg_terminate_backend`.
2. `VALID UNTIL` przy roli dotyczy tylko hasła trzymanego w PostgreSQL, więc przy logowaniu przez LDAP nic nie zmienia; zostaje drogą zapasową dla osób bez konta w domenie.
3. Prawa zapisu nadane `GRANT cennik_rw` nie wygasają same; `REVOKE` jest krokiem zgłoszenia.

Kolejność reguł i brak drugiej szansy

PostgreSQL wybiera pierwszą regułę, która pasuje do typu połączenia, bazy, roli i adresu. Gdy uwierzytelnienie w tej regule się nie uda, kolejnych reguł nie sprawdza [1]. Reguła `ldap` dla `cennik` musi więc stać nad ogólną regułą `scram-sha-256` dla sieci biura, a rola, która ma logować się hasłem w PostgreSQL, nie może należeć do `cennik_ro`, bo trafi w regułę LDAP i tam przepadnie.

Częste błędy

Objaw	Przyczyna	Co robimy
w logu `LDAP user "ola.wisniewska" does not exist`, choć konto w AD działa	wyszukiwanie z filtrem nie zwróciło konta: osoby nie ma w grupie, członkostwo wygasło, grupa zmieniła DN albo `ldapbasedn` nie obejmuje jednostki z kontem	uruchamiamy `ldapsearch` z Fiszki z tym samym filtrem, poprawiamy DN albo dodajemy osobę z wniosku
`could not search LDAP for filter` z diagnostyką `000004DC`	wyszukiwanie bez udanego bindu: najczęściej brak `ldapbinddn`, więc PostgreSQL próbuje anonimowo, a AD nie pozwala na wyszukiwanie anonimowe; możliwe też podążanie biblioteki za odsyłaczami do innych partycji katalogu przy wyszukiwaniu od korzenia domeny	uzupełniamy `ldapbinddn` i `ldapbindpasswd`, sprawdzamy je `ldapsearch -D ... -W`; przy odsyłaczach zawężamy `ldapbasedn` do jednostki z kontami (sprawdzamy w dokumentacji OpenLDAP opcję `REFERRALS` w `ldap.conf`)
logowanie działa z `ldapsearch`, z PostgreSQL nie, w logu błąd połączenia z serwerem LDAP	biblioteka OpenLDAP w usłudze nie ufa certyfikatowi albo w `ldapserver` stoi adres IP	`TLS_CACERT` w `/etc/openldap/ldap.conf`, pełne nazwy kontrolerów, test `openssl s_client`
`FATAL: role "ola.wisniewska" does not exist` po udanym haśle	PostgreSQL nie zakłada ról z AD	`CREATE ROLE "ola.wisniewska" LOGIN IN ROLE cennik_ro` z wniosku
Ola wpisuje `Ola.Wisniewska` i nie wchodzi	AD porównuje loginy bez względu na wielkość liter, PostgreSQL nazwy ról z nią	role zakładamy małymi literami i tak podajemy login w instrukcji dla zespołu
konto Oli zablokowane w AD po porannych próbach aplikacji	złe hasło w pliku połączenia aplikacji, każda próba liczy się do blokady konta w domenie	aplikacje łączą się własnym kontem technicznym w PostgreSQL, nie kontem człowieka
nikt nie loguje się do `cennik`, gdy `dc01` jest w aktualizacji	w `ldapserver` stoi jeden kontroler	dwa kontrolery rozdzielone spacją w cudzysłowie
po zmianie hasła `svc_pg01_ldap` logowanie przestaje działać	stare hasło w `ldapbindpasswd`	nowe hasło w regule, `pg_reload_conf()`, test `psql` w tej samej zmianie

Objaw	Przyczyna	Co robimy
audytor pyta, czemu hasła lecą otwartym tekstem	reguła `host` zamiast `hostssl` przy metodzie `ldap`	`hostssl` i `sslmode=verify-full` po stronie klientów

Ćwiczenia

1. Tomek proponuje uprościć regułę do trybu simple bind z `ldapsuffix="@ad.zielona.example"`, „bo nie trzeba konta technicznego”. Co tracimy?

Odpowiedź

Sprawdzenie grupy. W simple bind AD odpowiada tylko, czy hasło pasuje, więc zaloguje się każdy pracownik z rolą w bazie, także po wygaśnięciu członkostwa czasowego. Dostęp na wniosek KU-05 przestałby działać, bo nic nie łączy logowania z grupą.

1. Wniosek Oli o odczyt `cennik` wygasa w piątek o 16:00. O 16:30 Ola nadal pracuje w otwartym oknie `psql`. Dlaczego i co robimy?

Odpowiedź

PostgreSQL sprawdza grupę tylko przy logowaniu, a sesja otwarta przed terminem trwa do rozłączenia. Dyżurny sprawdza `pg_stat_activity`, kończy sesję `pg_terminate_backend` i zapisuje to w zgłoszeniu. Nowe logowanie już się nie uda, bo filtr `memberOf` nie znajdzie Oli.

1. Ola ma dostać na jeden dzień prawo zapisu w `cennik`. Dyżurny dodaje ją do grupy zapisu w AD z `-MemberTimeToLive`. Czy to wystarczy?

Odpowiedź

Nie. Grupa w AD tylko otwiera logowanie; prawa w bazie wynikają z roli. Potrzebne jest jeszcze `GRANT cennik_rw TO "ola.wisniewska"`, a w dniu końca `REVOKE`, bo nadanie w PostgreSQL samo nie wygasa. Filtr reguły musi też przepuszczać członków grupy zapisu.

1. Po przeniesieniu konta Oli do jednostki `OU=Programisci,DC=ad,DC=zielona,DC=example` logowanie przestało działać, a filtr w `ldapsearch` z bazą `DC=ad,DC=zielona,DC=example` znajduje konto. Gdzie szukamy błędu?

Odpowiedź

W `ldapbasedn` reguły: jeżeli ktoś zawęził go do `OU=Uzytkownicy,...`, wyszukiwanie PostgreSQL nie obejmuje nowej jednostki, choć `ldapsearch` z szerszą bazą konto widzi. Porównujemy opcje z `pg_hba.conf` z poleceniem testowym znak po znaku, najlepiej kopiując je z pliku.

Powiązane: Uwierzytelnianie i bezpieczeństwo połączeń · postgresql.conf i pg_hba.conf · Użytkownicy, role i uprawnienia · Sesje, długie zapytania, kasowanie kwerendy · Wnioski o usługę, katalog usług i onboarding: od nowego konta do odebrania dostępu · Logowanie kontem z Active Directory

Źródła (stan na 19.10.2026):

1. postgresql.org/docs/16: The pg_hba.conf File, łączenie linii ukośnikiem wstecznym, `hostssl`, znak `+` przy roli, brak przejścia do kolejnej reguły po nieudanym uwierzytelnieniu, przeładowanie; pg_hba_file_rules, widok domyślnie tylko dla superużytkownika.
2. postgresql.org/docs/16: LDAP Authentication, tryby simple bind i search+bind, opcje `ldapserver`, `ldapport`, `ldapscheme`, `ldaptls`, `ldapprefix`, `ldapsuffix`, `ldapbasedn`, `ldapbinddn`, `ldapbindpasswd`, `ldapsearchattribute`, `ldapsearchfilter`, `ldapurl`, zakaz mieszania trybów, rola musi istnieć w bazie, szyfrowanie tylko odcinka do serwera LDAP.
3. learn.microsoft.com: What's new in Active Directory Domain Services in Windows Server 2016, członkostwo z czasem życia (expiring links) dla `member` i `memberOf`; Privileged Access Management for Active Directory Domain Services, wygasłe członkostwo oceniane od razu na każdym kontrolerze.
4. man7.org: ldap.conf(5), `TLS_CACERT`, `TLS_REQCERT` z wartością domyślną `demand`, zmienne `LDAPCONF` i `LDAP<opcja>`.
5. learn.microsoft.com: Enable LDAP over SSL with a third-party certification authority, wymagania certyfikatu LDAPS kontrolera i port 636.
6. learn.microsoft.com: Search Filter Syntax, `LDAP_MATCHING_RULE_IN_CHAIN` (1.2.840.113556.1.4.1941) i uwaga o obciążeniu.
7. Komunikaty `LDAP user "..." does not exist`, `LDAP authentication failed for user` i `could not search LDAP for filter` z `000004DC`: wątki na liście pgsql-general (A vexing problem with LDAP, Test connection user with ldaps server). Narzędzia `pg-ldap-sync` i `ldap2pg`: tylko wzmianka, wersji nie sprawdzaliśmy.