Logi i diagnostyka Nginx

Poziom: średni **Dotyczy: Nginx 1.18+** **Czas: ~20 min**

Zdecydowana większość problemów z Nginksem daje się rozpoznać z samej treści logu — ta strona pokazuje, gdzie szukać i co znaczą najczęstsze kody błędów.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
HTTP	Hypertext Transfer Protocol: protokół stron WWW
JWT	JSON Web Token: podpisany token z danymi, tu przenoszący tagi stanu urządzenia z FortiClient do FortiOS
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

TYLKO ODCZYT

Sprawdzenie logu dostępu na żywo: tail access_log

```
sudo tail -f /var/log/nginx/access.log
```

> **Pojęcie: access_log.** Log zapisujący jedną linię na każde obsłużone żądanie HTTP — adres klienta, znacznik czasu, metodę i ścieżkę, kod odpowiedzi, rozmiar odpowiedzi. Domyślna lokalizacja to `/var/log/nginx/access.log`, ustawiana dyrektywą `access_log` w kontekście `http` albo `server`.

Przykład:

```
10.20.20.20 - - [22/Sep/2026:14:03:11 +0200] "GET /app/status HTTP/1.1" 200 512 "-"
"curl/8.4.0"
10.20.20.145 - - [22/Sep/2026:14:03:14 +0200] "POST /app/upload HTTP/1.1" 413 0 "-"
```

```
"Mozilla/5.0"
```

HTTP = Hypertext Transfer Protocol: protokół stron WWW · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Sprawdzenie logu błędów na żywo: tail error_log

```
sudo tail -f /var/log/nginx/error.log
```

> **Pojęcie: error_log.** Log zdarzeń wewnętrznych procesu — błędów połączenia z backendem, przekroczonych timeoutów, problemów ze składnią konfiguracji przy starcie. W przeciwieństwie do `access_log` nie zapisuje jednej linii na każde żądanie, tylko na sytuacje wymagające uwagi.

Przykład:

```
2026/09/22 14:05:02 [error] 1044#1044: *281 connect() failed (111: Connection refused)
while connecting to upstream, client: 10.20.20.20, server: NazwaDomeny, request: "GET
/app/status HTTP/1.1", upstream: "http://10.20.10.41:8080/status", host: "NazwaDomeny"
```

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Ustawienie poziomu szczegółowości logu błędów: error_log z poziomem

```
error_log /var/log/nginx/error.log warn;
```

> **Pojęcie: poziom logowania.** Nginx obsługuje poziomy `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert`, `emerg`, w kolejności rosnącej wagi — wskazany poziom zapisuje zdarzenia tego poziomu i wszystkich poważniejszych, ale nie łagodniejszych. `debug` wymaga specjalnie skompilowanej binarki (`--with-debug`) i generuje ogromną ilość danych, więc włączamy go tylko na czas diagnozowania konkretnego problemu, nigdy na stałe w produkcji.

TYLKO ODCZYT

Własny format logu z czasem odpowiedzi upstreamu: log_format

```
log_format upstream_timing '$remote_addr - $status "$request" '
                           'rt=$request_time uct="$upstream_connect_time" '
                           'urt="$upstream_response_time"';

access_log /var/log/nginx/access.log upstream_timing;
```

> **Pojęcie: log_format.** Definicja własnego układu kolumn logu dostępu, przydatna, gdy domyślny format `combined` nie pokazuje najważniejszej informacji przy diagnozowaniu wolnych żądań — czasu, jaki zajęło samo połączenie się z backendem (`$upstream_connect_time`) i czasu, jaki backend potrzebował na odpowiedź (`$upstream_response_time`), osobno od całkowitego czasu obsługi żądania (`$request_time`).

Przykład:

```
10.20.20.20 - 200 "GET /app/reports HTTP/1.1" rt=2.341 uct="0.002" urt="2.338"
```

Wynik pokazuje, że z 2,3 sekundy całkowitego czasu żądania niemal cały czas poszedł na oczekiwanie na odpowiedź backendu (`urt`), a nie na samo Nginksie ani na nawiązanie połączenia (`uct`) — problem wydajnościowy leży po stronie aplikacji, nie proxy.

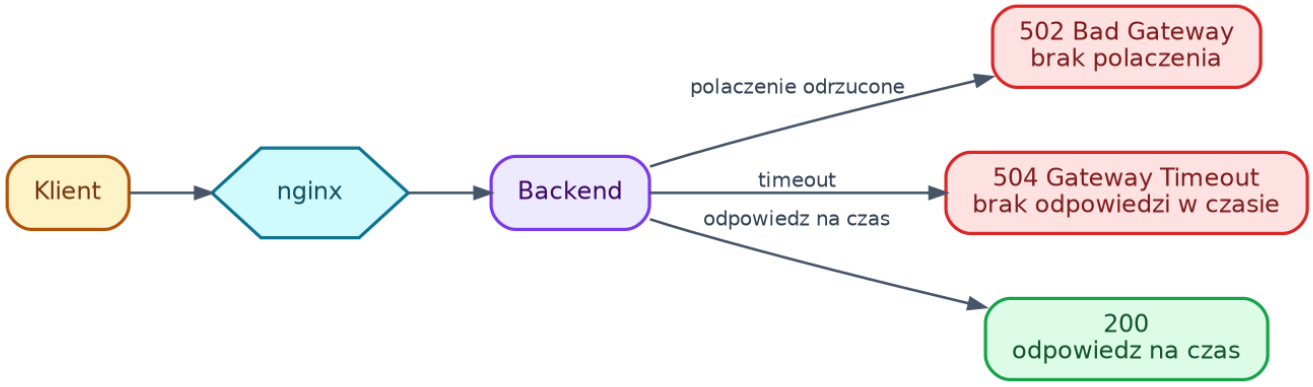
Z życia wzięte

Z ŻYCIA WZIĘTE

W ostatni dzień zamknięcia miesiąca pani Halina zgłasza o 20:00, że Optima przez przeglądarkę pokazuje błąd bramy zamiast raportu. Tomek sprawdza `error.log` na `docker01` i widzi `connect() failed (111: Connection refused)` dla backendu aplikacji księgowej, nie samego nginksa. Okazuje się, że usługa aplikacji padła pod obciążeniem końca miesiąca, więc restart backendu, nie zmiana konfiguracji proxy, przywraca dostęp.

Najczęstsze kody błędów i co znaczą

> **Pojęcie: kod odpowiedzi HTTP w kontekście reverse proxy.** Część kodów, które widzi klient, pochodzi bezpośrednio od aplikacji backendu i Nginx tylko je przekazuje dalej bez zmian; inne generuje sam Nginx, gdy backend jest w ogóle niedostępny albo odpowiada w sposób, którego proxy nie potrafi obsłużyć. Rozróżnienie tych dwóch źródeł jest pierwszym krokiem diagnozy.



Nginx zwraca 502, gdy backend odrzuca połączenie, a 504, gdy backend połączenie przyjął, ale nie odpowiedział w wyznaczonym czasie (kliknięcie otwiera pełny rozmiar)

Kod	Znaczenie	Typowa przyczyna
502 Bad Gateway	Nginx nawiązał albo próbował nawiązać połączenie z backendem, ale dostał odpowiedź, której nie da się zinterpretować, albo backend odmówił połączenia	aplikacja backendu nie działa, padła, albo `proxy_pass` wskazuje zły port
504 Gateway Timeout	Nginx połączył się z backendem, ale backend nie odpowiedział w czasie ustawionym w `proxy_read_timeout`	aplikacja wisi, wykonuje bardzo długie zapytanie do bazy, albo jest przeciążona
413 Request Entity Too Large	ciało żądania klienta przekroczyło limit	domyślny `client_max_body_size 1m` zbyt mały dla wgrywanego pliku, opisany na stronie Reverse proxy i równoważenie obciążenia

Kod	Znaczenie	Typowa przyczyna
403 Forbidden	Nginx odmówił dostępu do zasobu	najczęściej brak uprawnień systemowych do pliku albo katalogu wskazanego przez `root` / `alias` — user `nginx` / `www-data` nie ma prawa odczytu
upstream sent too big header (w logu, klient widzi 502)	odpowiedź backendu (zwykle nagłówki, np. duże ciasteczko albo długi token w nagłówku autoryzacji) nie zmieściła się w domyślnym buforze	zbyt małe `proxy_buffer_size` / `proxy_buffers` względem realnego rozmiaru nagłówków aplikacji

Diagnoza błędu 403 z uprawnieniami katalogu

```
sudo -u nginx cat /var/www/example/index.html
```

> **Pojęcie: user procesu roboczego.** Procesy robocze Nginksa działają jako user ustawiony dyrektywą `user` w `nginx.conf` (zwykle `nginx` na RHEL/Rocky, `www-data` na Debianie/Ubuntu), nie jako root — plik albo katalog musi być dla tego usera przynajmniej do odczytu, a każdy katalog na ścieżce do pliku musi mieć dla niego prawo wykonania (`x`), inaczej Nginx nie ma jak w ogóle wejść do środka, nawet jeśli sam plik ma poprawne uprawnienia.

Przykład błędu w logu przy braku uprawnień:

```
2026/09/22 14:12:07 [error] 1044#1044: *290 open() "/var/www/example/index.html" failed (13:
Permission denied), client: 10.20.20.20, server: NazwaDomeny, request: "GET / HTTP/1.1"
```

Rozwiązanie upstream sent too big header

```
proxy_buffer_size 16k;
proxy_buffers 4 16k;
```

Podnosimy rozmiar bufora na nagłówki odpowiedzi backendu — typowo potrzebne przy aplikacjach zwracających duże tokeny JWT albo wiele nagłówków `Set-Cookie` naraz.

Rotacja logów

> **Pojęcie: rotacja logów.** Mechanizm, który po osiągnięciu określonego rozmiaru albo wieku przenosi bieżący plik logu na bok (zwykle z dopiskiem daty albo numeru) i tworzy w jego miejsce nowy, pusty — bez tego pliki logów rosłyby bez ograniczeń aż do zapełnienia dysku. Na dystrybucjach korzystających z `logrotate` konfiguracja leży w `/etc/logrotate.d/nginx` i jest instalowana razem z pakietem.

Uważaj: samo przeniesienie pliku logu na bok przez `logrotate` nie wystarcza — Nginx trzyma otwarty deskryptor do starego pliku pod jego starą nazwą i dalej do niego pisze, dopóki nie dostanie sygnału do ponownego otwarcia plików logów. Standardowa konfiguracja `logrotate` dla Nginksa robi to automatycznie po rotacji:

```
sudo nginx -s reopen
```

Limity połączeń na poziomie systemu

Limit deskryptorów plików procesu głównego: worker_rlimit_nofile

```
worker_rlimit_nofile 65535;
```

> **Pojęcie: deskryptor plików.** Każde otwarte połączenie sieciowe (do klienta albo do backendu) i każdy otwarty plik zużywa jeden deskryptor procesu — limit domyślny systemu operacyjnego dla pojedynczego procesu bywa zbyt niski dla serwera obsługującego dużą liczbę jednoczesnych połączeń. `worker_rlimit_nofile` podnosi ten limit dla procesów roboczych Nginksa niezależnie od domyślnego limitu systemowego usera, podobnie jak `LimitNOFILE` w jednostce systemd, opisane na stronie Apache Solr: utrzymanie na przykładzie usługi Solr.

Sprawdzenie aktualnego limitu działającego procesu:

```
cat /proc/$(pgrep -o nginx)/limits | grep 'open files'
```

Przykład:

```
Max open files            65535                65535                files
```

Częste błędy

Interpretacja 502 jako problemu samego Nginksa. 502 niemal zawsze oznacza, że winowajcą jest backend albo sieć między Nginksem a backendem, nie sama konfiguracja Nginksa — pierwszym krokiem diagnozy jest sprawdzenie, czy backend w ogóle żyje i nasłuchuje na porcie wskazanym w `proxy_pass`, dopiero potem szukanie błędu w konfiguracji proxy.

Zbyt krótki proxy_read_timeout dla wolnych operacji. Endpoint generujący raport albo eksport danych regularnie kończy się 504, mimo że backend faktycznie by odpowiedział, gdyby dać mu więcej czasu — rozwiązaniem jest podniesienie `proxy_read_timeout` konkretnie dla tego `location`, a nie globalnie dla całego serwera.

Log rośnie mimo skonfigurowanego logrotate. Plik przeniesiony przez `logrotate` nadal rośnie pod starą nazwą (`access.log.1` zamiast nowego `access.log`) — brakuje kroku `nginx -s reopen` po rotacji, standardowo wywoływanego automatycznie przez skrypt `postrotate` w konfiguracji dystrybucyjnej, ale wartego ręcznego sprawdzenia po migracji albo customowej instalacji.

□□ Ćwiczenia

> Ćwiczenia zakładają lokalny Nginx z dowolnym backendem testowym, który można celowo zatrzymać.

Ćwiczenie 1 — wywołanie i rozpoznanie błędu 502

```
sudo systemctl stop docker  # albo inny sposób zatrzymania backendu
curl -i http://localhost/app/
```

Zatrzymujemy backend, wysyłamy żądanie przez Nginksa i sprawdzamy zarówno odpowiedź dla klienta, jak i wpis w `error.log`.

Odpowiedź

Klient dostaje `502 Bad Gateway`, a w `error.log` pojawia się wpis `connect() failed (111: Connection refused)` z adresem i portem backendu, do którego Nginx próbował się połączyć.

Pytania kontrolne

1. Dlaczego 504 nie zawsze oznacza awarię backendu, w odróżnieniu od 502?

Odpowiedź

502 oznacza, że backend nie odpowiedział wcale albo odpowiedział w sposób niezrozumiały dla Nginksa (np. odmówił połączenia) — 504 oznacza, że backend żyje i przyjął połączenie, ale nie zdążył odpowiedzieć w ustalonym czasie, co może oznaczać zarówno realną awarię, jak i po prostu bardzo wolną, ale poprawną operację.

Powiązane: Reverse proxy i równoważenie obciążenia · Architektura Nginx: procesy, konteksty, pliki konfiguracyjne · Apache Solr: utrzymanie

Źródła: dokumentacja nginx.org, rozdziały „ngx_http_log_module" i „Debugging Log" · `man 8 logrotate`

Revision #37
Created 2026-09-25 07:40:16 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:16 UTC by Tomasz Konieczny