Wymiana plików

Serwer SFTP i Apache Solr.

- Serwer SFTP
- Apache Solr: utrzymanie
- Apache Solr w Dockerze: instalacja, rdzeń i pierwsze wyszukiwanie

Serwer SFTP

Typ: procedura **Dotyczy: OpenSSH na RHEL/Rocky i Debian/Ubuntu** **Czas: ~30 min**

Efekt: nowy użytkownik zamknięty w swoim katalogu (chroot), logujący się wyłącznie po SFTP kluczem SSH, bez dostępu do powłoki i bez możliwości wyjścia poza własny katalog.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
FTP	File Transfer Protocol: stary protokół przesyłania plików, bez szyfrowania
FTPS	FTP Secure: protokół FTP z szyfrowaniem TLS
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SFTP	SSH File Transfer Protocol: bezpieczne przesyłanie plików przez SSH
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Czym jest SFTP

Pojęcie: SFTP (SSH File Transfer Protocol). Protokół przesyłania plików działający w całości wewnątrz jednej sesji SSH, na tym samym porcie 22 co zwykłe logowanie terminalowe — w odróżnieniu od FTP, nie otwiera osobnych, dodatkowych połączeń na inne porty do przesyłania danych, więc jest prostszy do przepuszczenia przez firewall i domyślnie w pełni szyfrowany, bo dziedziczy szyfrowanie z samego SSH.

Pojęcie: FTPS. FTP rozszerzone o TLS — inny protokół niż SFTP, mimo podobnej nazwy i podobnego zastosowania. FTPS korzysta z oddzielnych połączeń kontrolnych i danych (tak jak zwykłe FTP), więc wymaga otwarcia szerszego zakresu portów na firewallu, a jego bezpieczeństwo zależy od poprawnej konfiguracji certyfikatu TLS, zupełnie osobno od kluczy SSH. Konta konfigurowane poniższą procedurą korzystają wyłącznie z SFTP — nie z FTPS.

Zanim zaczniemy

> **Z ŻYCIA WZIĘTE**
>
> O 8:15 dział księgowości zgłasza Tomkowi, że kontrahent z zewnątrz od rana nie może wgrać pliku na `app01`, bo dostaje `Connection closed` zaraz po podaniu hasła. Tomek sprawdza `ls -ld` na katalogu chroot i widzi, że ktoś wieczorem poprzedniego dnia odpalił na nim `chmod 775` przy porządkowaniu uprawnień, bo katalog wyglądał „za bardzo zamknięty". Po przywróceniu `root:root` i `755` logowanie od razu wraca do działania.

Wymaganie	Jak sprawdzić
Serwer OpenSSH zainstalowany i działający	`systemctl status sshd.service`
Katalog nadrzędny dla kont SFTP istnieje na dysku (np. `/ftp`)	`ls -ld /ftp`
Grupa systemowa dla kont SFTP istnieje	`getent group sftpusers`
Klucz publiczny użytkownika dostarczony z wyprzedzeniem	plik `.pub` od użytkownika, nie generujemy go za niego

Uważaj: wymagania własnościowe katalogu chroot są bardzo sztywne i to jest najczęstszy powód, dla którego logowanie nie działa mimo poprawnej reszty konfiguracji — katalog wskazany w `ChrootDirectory` oraz każdy katalog nadrzędny na ścieżce do niego musi należeć do roota (`root:root`) i nie może mieć prawa zapisu dla grupy ani dla innych. OpenSSH odmawia użycia chroot, jeśli choć jeden katalog na tej ścieżce jest zapisywalny przez kogokolwiek poza rootem — traktuje to jako lukę bezpieczeństwa, bo użytkownik zamknięty w chroot mógłby inaczej podmienić zawartość katalogu, z którego korzysta sam proces SSH.

Przebieg

1. Utworzenie grupy dla kont SFTP

```
sudo groupadd sftpusers
```

Jedna wspólna grupa systemowa, do której będą należeć wszyscy użytkownicy objęci regułami chroot z kroku 5 — blok `Match Group` w `sshd_config` odwołuje się do niej po nazwie, więc dodanie kolejnego użytkownika do tej grupy automatycznie obejmuje go tymi samymi regułami, bez dodatkowej zmiany w `sshd_config`.

Weryfikujemy:

```
getent group sftpusers
```

Przykład:

```
sftpusers:x:1050:
```

2. Utworzenie konta użytkownika bez powłoki

```
sudo useradd -d /ftp/nazwa_usera -g sftpusers -s /usr/sbin/nologin nazwa_usera
```

> (gdzie `nazwa_usera` to nazwa konta systemowego tworzonego dla klienta albo integracji korzystającej z SFTP)

Powłoka `/usr/sbin/nologin` uniemożliwia temu kontu zwykłe logowanie terminalowe przez SSH — w połączeniu z `ForceCommand internal-sftp` z kroku 5 i tak wymusilibyśmy sesję SFTP, ale ustawienie `nologin` już na poziomie samego konta jest dodatkową warstwą zabezpieczenia, niezależną od konfiguracji SSH.

3. Ustawienie hasła konta (opcjonalnie, przy braku logowania kluczem)

```
sudo passwd nazwa_usera
```

Wymagane tylko wtedy, gdy to konto ma się logować hasłem zamiast kluczem — w docelowej konfiguracji z krokiem 7 (klucz zamiast hasła) ten krok pomijamy albo blokujemy hasło komendą `passwd -l nazwa_usera` po dodaniu klucza.

Przykład:

```
New password:
Retype new password:
passwd: password updated successfully
```

4. Utworzenie struktury katalogu chroot

```
sudo mkdir -p /ftp/nazwa_usera/dane
sudo chown root:root /ftp/nazwa_usera
sudo chmod 755 /ftp/nazwa_usera
sudo chown nazwa_usera:sftpusers /ftp/nazwa_usera/dane
```

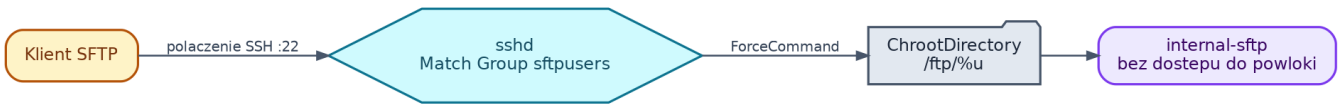
Katalog główny użytkownika (`/ftp/nazwa_usera`) musi zostać własnością roota, zgodnie z wymaganiem z sekcji „Zanim zaczniemy" — dopiero podkatalog wewnątrz niego (`dane`) należy do samego użytkownika i to w nim faktycznie ląduje przesyłane przez niego pliki. Próba zapisu bezpośrednio do katalogu głównego chroot kończy się odmową uprawnień, co jest zamierzonym zachowaniem, nie usterką.

5. Dodanie bloku Match Group w sshd_config

```
sudo tee -a /etc/ssh/sshd_config <<'EOF'

Match Group sftpusers
    ChrootDirectory /ftp/%u
    ForceCommand internal-sftp
    AllowTcpForwarding no
    X11Forwarding no
EOF
```

> **Pojęcie: Match Group.** Blok w `sshd_config`, którego dyrektywy obowiązują tylko dla połączeń użytkowników należących do wskazanej grupy — wszystko poza tym blokiem działa dla nich tak samo jak dla reszty serwera, ale wewnątrz bloku nadpisujemy kluczowe ustawienia specyficzne dla kont SFTP.



Połączenie SFTP przechodzi kolejno przez sshd, katalog chroot wskazany przez ForceCommand i wbudowaną implementację internal-sftp (kliknięcie otwiera pełny rozmiar)

`%u` w ścieżce `ChrootDirectory` rozwija się automatycznie do nazwy loginu łączącego się użytkownika, więc jeden wpis w konfiguracji obsługuje dowolną liczbę kont bez dopisywania

osobnej reguły na każde z nich.

> **Pojęcie: ForceCommand internal-sftp.** Niezależnie od tego, jaką komendę użytkownik próbowałby wywołać przy połączeniu, serwer ignoruje ją i zawsze uruchamia wbudowaną w OpenSSH implementację SFTP (`internal-sftp`), działającą w pamięci bez odwołania do zewnętrznego binarnego pliku wykonywalnego wewnątrz chroot — dzięki temu nie musimy w ogóle kopiować żadnych binariów do środowiska chroot, jak byłoby to konieczne przy klasycznym `/usr/lib/openssh/sftp-server`.

`AllowTcpForwarding no` blokuje wykorzystanie tego konta jako tunelu SSH do innych hostów w sieci — bez tego ograniczenia konto ograniczone do SFTP nadal mogłoby posłużyć jako punkt przeskoku sieciowego, mimo braku dostępu do powłoki.

6. Sprawdzenie składni i przeładowanie sshd

```
sudo sshd -t
```

Sprawdzamy składnię konfiguracji przed przeładowaniem — błąd w `sshd_config` sprawdzony dopiero po restarcie usługi potrafi odciąć wszystkie sesje SSH do serwera, łącznie z sesją administratora.

```
sudo systemctl restart sshd.service
```

7. Konfiguracja logowania kluczem zamiast hasłem

```
sudo mkdir -p /ftp/nazwa_usera/dane/.ssh
```

Katalog `.ssh` umieszczamy wewnątrz podkatalogu należącego do użytkownika (`dane`), nie w katalogu głównym chroot — z tych samych powodów własnościowych co w kroku 4.

Wgrywamy dostarczony wcześniej klucz publiczny użytkownika do pliku `authorized_keys`:

```
sudo tee /ftp/nazwa_usera/dane/.ssh/authorized_keys < SciezkaDoPliku
```

Ustawiamy wymagane, restrykcyjne uprawnienia:

```
sudo chown -R nazwa_usera:sftpusers /ftp/nazwa_usera/dane/.ssh
sudo chmod 700 /ftp/nazwa_usera/dane/.ssh
sudo chmod 600 /ftp/nazwa_usera/dane/.ssh/authorized_keys
```

Uważaj: OpenSSH odmawia użycia pliku `authorized_keys`, jeśli katalog `.ssh` albo sam plik mają zbyt szerokie uprawnienia (np. zapis dla grupy) — te same zasady bezpieczeństwa, co przy zwykłym logowaniu SSH kluczem, obowiązują też wewnątrz chroot.

Wskazujemy niestandardową ścieżkę pliku kluczy w bloku `Match Group`, jeśli różni się od domyślnej lokalizacji względem katalogu domowego:

```
sudo sed -i '/Match Group sftpusers/a\    AuthorizedKeysFile
/ftp/%u/dane/.ssh/authorized_keys' /etc/ssh/sshd_config
sudo sshd -t && sudo systemctl restart sshd.service
```

Weryfikacja

Sprawdzenie logowania kluczem

```
sftp -i SciezkaDoKlucza -P 22 nazwa_usera@NazwaSerwera
```

Przykład udanego połączenia:

```
Connected to NazwaSerwera.
sftp> pwd
Remote working directory: /
sftp> ls
dane
sftp> cd dane
sftp> put plik_testowy.txt
Uploading plik_testowy.txt to /dane/plik_testowy.txt
plik_testowy.txt                              100%   12    0.1KB/s   00:00
```

Katalog widziany po zalogowaniu to `/` — to jest właściwe zachowanie chroot: użytkownik widzi katalog `/ftp/nazwa_usera` jako swój własny korzeń systemu plików i nie ma żadnej możliwości wyjścia poza niego, nawet komendą `cd ..`.

Sprawdzenie uprawnień katalogu chroot

```
ls -ld /ftp/nazwa_usera
```

Przykład poprawnego wyniku:

```
drwxr-xr-x. 3 root root 4096 Sep 24 10:15 /ftp/nazwa_usera
```

Właściciel `root:root` i brak prawa zapisu dla grupy ani innych (`w` obecne tylko w pierwszej trójce uprawnień) potwierdzają zgodność z wymaganiem z sekcji „Zanim zaczniemy".

Wycofanie zmian

Usunięcie konta i jego danych, gdy dostęp SFTP nie jest już potrzebny:

```
sudo userdel nazwa_usera
```

```
sudo rm -rf /ftp/nazwa_usera
```

Usunięcie bloku `Match Group` z `sshd_config` ma sens dopiero, gdy usuwamy ostatnie konto należące do grupy `sftpusers` — dopóki istnieje choć jedno takie konto, blok zostaje.

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa
`Connection closed` zaraz po próbie logowania	katalog `ChrootDirectory` albo katalog nadrzędny ma złego właściciela albo zbyt szerokie uprawnienia	`ls -ld` na każdym katalogu ścieżki aż do `/`, poprawa na `root:root` i `755` lub węziej
`Permission denied (publickey)` mimo poprawnego klucza	zły właściciel albo uprawnienia `.ssh`/`authorized_keys`, albo klucz w niewłaściwej lokalizacji	sprawdzenie `chmod 700`/`600` i ścieżki wskazanej w `AuthorizedKeysFile`

Objaw	Przyczyna	Naprawa
Użytkownik widzi więcej niż powinien, w tym pliki systemowe	brak albo błędny blok `Match Group`, `ChrootDirectory` nie zadziałał	`sudo sshd -T

Objaw	Przyczyna	Naprawa
Logowanie działa, ale użytkownik trafia do powłoki zamiast sesji SFTP	brak `ForceCommand internal-sftp` w bloku `Match Group`	dodanie dyrektywy i przeładowanie `sshd`
`sshd -t` zgłasza błąd składni po edycji	literówka w nazwie dyrektywy albo zły poziom wcięcia bloku `Match`	poprawa pliku, ponowne `sshd -t` przed jakimkolwiek restartem usługi

Powiązane: Apache Solr: utrzymanie · Certyfikaty SSL: wystawienie, konwersja, weryfikacja

Źródła: `man 5 sshd_config` · `man 8 sshd` · dokumentacja OpenSSH, rozdział o `ChrootDirectory` i `internal-sftp`

Apache Solr: utrzymanie

Poziom: średni **Dotyczy: Apache Solr 8.x/9.x na systemd** **Czas: ~20 min**

Solr rzadko wymaga zmian w codziennej eksploatacji, ale dwie rzeczy wracają regularnie: limit otwartych plików usługi i zabezpieczenie dostępu do API basic authem — obie opisujemy tu wprost, z realnym przykładem override'u systemd.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
HTTP	Hypertext Transfer Protocol: protokół stron WWW
JVM	Java Virtual Machine: maszyna wirtualna, w której działają programy w Javie
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
SolrCloud	tryb Solra jako klastra z wbudowanym albo zewnętrznym ZooKeeperem, w którym dane trzymamy w kolekcjach

Fiszka

TYLKO ODCZYT

Sprawdzenie aktualnego limitu działającej usługi: systemctl show

```
sudo systemctl show solr.service -p LimitNOFILE
```

Wartość `15000` to typowy limit domyślny, jaki instalator Solr ustawia samodzielnie bez dodatkowej konfiguracji.

Przykład:

```
LimitNOFILE=15000
```

systemctl = system control: sterowanie usługami systemd

□□ TYLKO ODCZYT

Utworzenie override'u limitu: systemctl edit

```
sudo systemctl edit solr.service
```

□□ **systemctl** = system control: sterowanie usługami systemd

□□ PLIK KONFIGURACYJNY

Zawartość override'u limitów: [Service] LimitNOFILE

```
[Service]
LimitNOFILE=70000
LimitNPROC=65000
LimitSIGPENDING=65000
```

`LimitNPROC` ogranicza analogicznie liczbę procesów/wątków, a `LimitSIGPENDING` liczbę oczekujących sygnałów — przy usłudze typu Solr, mocno wielowątkowej, warto podnosić wszystkie trzy limity razem, nie tylko `LimitNOFILE`.

□□ ZMIENIA SYSTEM

Przeładowanie konfiguracji systemd i restart usługi: daemon-reload

```
sudo systemctl daemon-reload
sudo systemctl restart solr.service
```

Uważaj: sam zapis pliku override nie wystarcza — `daemon-reload` każe systemd ponownie wczytać definicje jednostek z dysku, a restart usługi jest konieczny, bo `LimitNOFILE` jest odczytywany tylko raz, w momencie startu procesu.

□□ **systemctl** = system control: sterowanie usługami systemd

□□ TYLKO ODCZYT

Weryfikacja zastosowanego limitu: systemctl show + /proc/limits

```
sudo systemctl show solr.service -p LimitNOFILE
cat /proc/$(pgrep -f solr.jar)/limits | grep 'open files'
```

Druga komenda sprawdza limit faktycznie zastosowany do działającego procesu, niezależnie od tego, co deklaruje sama definicja jednostki — dobra praktyka po każdej zmianie tego typu parametru, bo różnica między nimi ujawniłaby błąd w kolejności `daemon-reload` i restartu.

Przykład:

```
LimitNOFILE=70000
Max open files            70000                70000                files
```

systemctl = system control: sterowanie usługami systemd · **cat** = concatenate: sklej i wypisz pliki

ZMIENIA SYSTEM

Wyłączenie basic auth: mv security.json

```
sudo mv /solr/data/security.json /solr/data/security.json.disabled
```

Solr przy starcie nie znajduje pliku o oczekiwanej nazwie i traktuje instancję jako nieuwierzytelnioną — dane konfiguracji użytkowników i haseł zostają zachowane w przemianowanym pliku, gotowe do przywrócenia bez ponownego wpisywania hashy.

mv = move: przenieś albo zmień nazwę

ZMIENIA SYSTEM

Włączenie basic auth z powrotem: mv security.json.disabled

```
sudo mv /solr/data/security.json.disabled /solr/data/security.json
```

Sama zmiana nazwy pliku z powrotem na `security.json` przywraca poprzednią konfigurację uwierzytelniania dokładnie taką, jaka była przed wyłączeniem.

mv = move: przenieś albo zmień nazwę

ZMIENIA SYSTEM

Restart usługi po zmianie basic auth: systemctl restart

```
sudo systemctl restart solr.service
```

systemctl = system control: sterowanie usługami systemd

ZMIENIA SYSTEM

Zmiana hasła przez API bez restartu usługi: curl set-user

```
curl --user NazwaUzytkownika http://localhost:8983/solr/admin/authentication -H
'Content-type:application/json' -d '{"set-user": {"NazwaUzytkownika":"NoweHaslo"}}'
```

Komenda prosi o aktualne hasło użytkownika, którego dotyczy zmiana (parametr `--user` bez podanego hasła w linii komendy — `curl` zapyta o nie interaktywnie), a nowe hasło przekazuje w treści żądania — Solr sam wylicza nowy hash i zapisuje go do `security.json`, bez potrzeby restartu usługi.

Przykład:

```
Enter host password for user 'solr':
{
  "responseHeader":{
    "status":0,
    "QTime":45}}
```

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Sprawdzenie uprawnień pliku security.json: ls -l

```
ls -l /solr/data/security.json
```

Uważaj: błąd `access denied` przy próbie zmiany hasła albo przy zwykłym zapytaniu do API po włączeniu basic auth często nie oznacza złego hasła, tylko złe uprawnienia systemowe do pliku `security.json` — plik musi być czytelny dla usera i grupy, na których działa proces Solr.

Przykład poprawnych uprawnień:

```
-rw-r----- 1 solr solr 512 Sep 24 11:02 /solr/data/security.json
```

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **ls** = list: wypisz zawartość katalogu

TYLKO ODCZYT

Status usługi: systemctl status

```
sudo systemctl status solr.service
```

systemctl = system control: sterowanie usługami systemd

TYLKO ODCZYT

Logi usługi: journalctl

```
sudo journalctl -u solr.service -n 100 --no-pager
```

Dokładnie taki komunikat jest jednym z sygnałów, że warto sprawdzić bieżący `LimitNOFILE`, opisany na początku tej strony.

Przykład wpisu przy problemie z limitem plików:

```
Sep 24 10:58:03 NazwaSerwera solr[2841]: WARN  [Thread-12] TransactionLog Too many open files
```

journalctl = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Miejsce zajmowane przez indeks na dysku: du -sh

```
du -sh /solr/data/*
```

Przykład:

```
4.2G    /solr/data/NazwaRdzenia
512K    /solr/data/security.json
```

du = disk usage: ile miejsca zajmuje katalog

Z życia wzięte

Z ŻYCIA WZIĘTE

O 16:40 aplikacja zgłasza masowe błędy 401 przy wyszukiwaniu na `app01`, chociaż nikt tego dnia nie zmieniał haseł. Tomek sprawdza `journalctl` i widzi, że wcześniej ktoś ręcznie edytował `security.json`, żeby dopisać nowego użytkownika, i literówka w JSON-ie zepsuła cały plik. Przywraca kopię pliku sprzed edycji i restartuje `solr.service`, po czym umawia się z zespołem, że kolejne konta dodają wyłącznie przez API `set-user`, nie ręczną edycją.

Czym jest Solr

Pojęcie: Apache Solr. Silnik wyszukiwania pełnotekstowego zbudowany na bibliotece Lucene, udostępniający dane przez REST API po HTTP — używamy go jako komponentu wspomagającego wyszukiwanie w aplikacjach, które przechowują właściwe dane gdzie indziej (np. w relacyjnej bazie danych), a do Solr wysyłają tylko kopię potrzebną do indeksowania i szybkiego przeszukiwania.

Pojęcie: rdzeń (core) i kolekcja. Rdzeń to pojedyncza, samodzielna instancja indeksu wewnątrz Solr, ze swoim własnym schematem pól i własnymi danymi — odpowiednik jednej bazy danych w silniku relacyjnym. Kolekcja to pojęcie z trybu rozproszonego Solr (SolrCloud), gdzie jedna logiczna kolekcja może być fizycznie podzielona na kilka rdzeni (shardów) rozłożonych na różnych węzłach — w prostszej instalacji jednowęzłowej, jaką najczęściej utrzymujemy, jedna kolekcja odpowiada dokładnie jednemu rdzeniowi i rozróżnienie to nie ma praktycznego znaczenia.

Limit otwartych plików dla usługi solr.service

Pojęcie: LimitNOFILE. Parametr jednostki systemd ograniczający liczbę deskryptorów plików (w tym połączeń sieciowych), jakie proces usługi może mieć otwarte jednocześnie — Solr, jako proces Javy obsługujący jednocześnie wiele segmentów indeksu na dysku i wiele połączeń HTTP, potrafi przy większym ruchu i większym indeksie łatwo wyczerpać zbyt niski limit domyślny systemu.

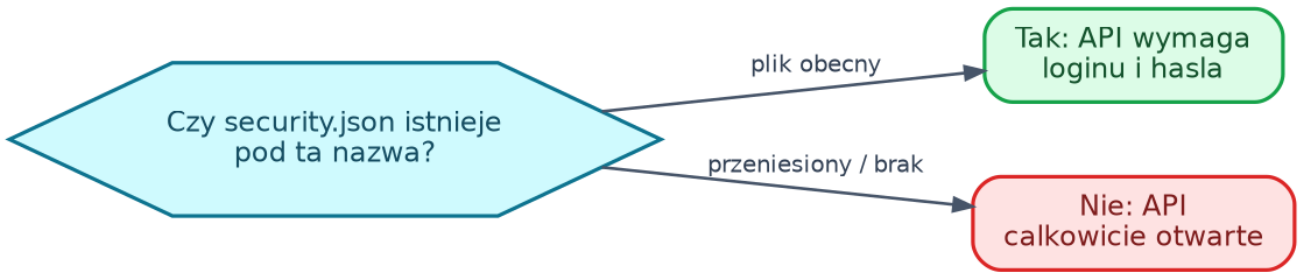
Uważaj: limit ustawiony zbyt wysoko (rzędu miliona) bywa równie problematyczny co zbyt niski — w praktyce obserwowaliśmy, że tak wysoki limit sam w sobie powodował problemy wydajnościowe aplikacji, prawdopodobnie przez sposób, w jaki JVM rezerwuje wewnętrzne struktury proporcjonalnie do zadeklarowanego limitu. Wartości rzędu kilkudziesięciu tysięcy sprawdzają się w praktyce lepiej niż skrajnie wysokie.

Utworzenie override'u limitu: systemctl edit

> **Pojęcie: override systemd.** Zamiast edytować bezpośrednio plik jednostki dostarczony przez pakiet (co zostałoby nadpisane przy kolejnej aktualizacji), `systemctl edit` tworzy osobny plik nakładki w `/etc/systemd/system/solr.service.d/override.conf`, którego zawartość systemd łączy z oryginalną jednostką przy starcie — to jest właściwy, przetrwały aktualizacje sposób zmiany parametrów usługi zainstalowanej z pakietu.

Włączanie i wyłączanie basic auth

> **Pojęcie: basic auth w Solr.** Solr przechowuje konfigurację uwierzytelniania w pliku `security.json` w katalogu danych instancji — obecność tego pliku pod właściwą nazwą włącza wymóg podania loginu i hasła przy każdym zapytaniu do API, jego brak (albo inna nazwa pliku) pozostawia API całkowicie otwarte.



Obecność pliku security.json pod właściwą nazwą decyduje o tym, czy API Solr wymaga uwierzytelnienia (kliknięcie otwiera pełny rozmiar)

Restart usługi po zmianie basic auth

> **Pojęcie: security.json.** Solr nie obserwuje tego pliku na żywo — każda zmiana jego obecności albo zawartości (poza zmianą hasła przez API, opisaną niżej) wymaga restartu usługi, żeby zaczęła obowiązywać.

Zmiana hasła użytkownika SolR

> **Pojęcie: hash sha256 z solą w security.json.** Hasła w pliku `security.json` nigdy nie są przechowywane jawnym tekstem — Solr zapisuje je jako hash SHA-256 połączony z losową solą (`salt`), unikalną dla każdego użytkownika, zabezpieczającą przed atakiem ze wcześniej wyliczonymi tablicami hashy (rainbow tables). Struktura wpisu w pliku wygląda tak:

```
{
  "authentication": {
    "class": "solr.BasicAuthPlugin",
    "credentials": {
      "NazwaUzytkownika": "5v3Z+z7z1eT0hR2xQwK9L3z8n1F4vXyZbA2kQhWn3xI=
Yh8pQz2xN1kL7vB3mR9tS4wJ6cX0eA5dF2gH8jK1lM3n"
    }
  }
}
```

Pierwsza część wartości (przed spacją) to hash, druga to sól — ręczna edycja tego pola bez znajomości dokładnego algorytmu solenia praktycznie zawsze kończy się niedziałającym uwierzytelnianiem, dlatego zmianę hasła wykonujemy przez API, a nie ręczną edycją pliku.

Częste błędy

Zmiana LimitNOFILE bez daemon-reload. `systemctl restart` samo z siebie nie wystarcza, jeśli plik override został dopiero co utworzony — systemd musi najpierw wczytać nową definicję jednostki komendą `daemon-reload`, inaczej restart uruchamia usługę wciąż ze starym limitem.

Ustawienie zbyt wysokiego limitu w oczekiwaniu na lepszą wydajność. Skrajnie wysoki `LimitNOFILE` (rzędu miliona) bywa źródłem tych samych problemów wydajnościowych, którym miał

zapobiec — wartości pośrednie, rzędu kilkudziesięciu tysięcy, sprawdzają się lepiej w praktyce.

Access denied po zmianie hasła przez API. Zwykle nie chodzi o samo hasło, tylko o uprawnienia systemowe pliku `security.json` — sprawdzamy właściciela i grupę pliku, zanim zaczniemy podejrzewać logikę uwierzytelniania.

□□ Ćwiczenia

> Ćwiczenia zakładają lokalną instalację Apache Solr uruchomioną jako usługa systemd.

Ćwiczenie 1 — pełny cykl zmiany limitu

```
sudo systemctl show solr.service -p LimitNOFILE
sudo systemctl edit solr.service
sudo systemctl daemon-reload
sudo systemctl restart solr.service
sudo systemctl show solr.service -p LimitNOFILE
```

Sprawdzamy limit przed zmianą, dodajemy override podnoszący `LimitNOFILE`, przeładowujemy i restartujemy, a na końcu potwierdzamy nową wartość.

Odpowiedź

Pierwsze wywołanie `systemctl show` powinno pokazać wartość domyślną (np. `15000`), a ostatnie — wartość ustawioną w override (np. `70000`). Jeśli wartość się nie zmieniła, najczęstszą przyczyną jest pominięty `daemon-reload` przed restartem.

Powiązane: Serwer SFTP · Logi i diagnostyka

Źródła: `man 5 systemd.exec` (sekcja `LimitNOFILE`) · dokumentacja Apache Solr, rozdział „Basic Authentication Plugin"

Apache Solr w Dockerze: instalacja, rdzeń i pierwsze wyszukiwanie

Typ: procedura

Dotyczy: Apache Solr 10.0.0, obraz solr:10.0.0 (sprawdzono na Docker Hub i w dokumentacji projektu 3.10.2026)

Czas: ~30 min

Efekt: działający kontener Solr na `docker01`, z jednym rdzeniem `dokumenty`, kilkoma załadowanymi dokumentami i pierwszym udanym wyszukiwaniem. Utrzymanie zainstalowanego Solra (limity plików, hasła) opisujemy w Apache Solr: utrzymanie, tu nie dublujemy.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
CNET	CNET Networks: amerykańska firma medialna, w której w 2004 roku powstał Solr
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
SolrCloud	tryb Solra jako klastra z wbudowanym albo zewnętrznym ZooKeeperem, w którym dane trzymamy w kolekcjach
ZooKeeper	Apache ZooKeeper: osobny program, który w klastrze trzyma wspólną konfigurację i pilnuje, kto jest kim

Zanim zaczniemy

Z ŻYCIA WZIĘTE

> Wtorek, 9:30. Marek ze Swarzędzu skarży się, że wyszukiwarka instrukcji magazynowych w aplikacji Oli szuka ponad dziesięć sekund, bo baza na `pg01` przegląda każdy wiersz tekstu po kolei. Ola prosi o prawdziwy silnik wyszukiwania. Tomek stawia Solra w kontenerze na `docker01`, zakłada rdzeń `dokumenty`, a po kilkunastu minutach ta sama fraza wraca w kilka milisekund. Dane zostają w bazie, Solr dostaje tylko ich kopię do szukania.

Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo na maszynie z zainstalowanym Dockerem i wtyczką Compose. Nazwy, hasła i adresy podmieniamy na własne.

Skąd się wziął Solr

Doug Cutting napisał Lucene w 1999 roku, w Javie, jako bibliotekę do szybkiego przeszukiwania tekstu. W 2001 roku trafiła do Apache, a w 2005 stała się samodzielnym projektem. Nazwa Lucene to drugie imię żony Douga, które nosiła też jej babcia.

Lucene jest biblioteką, nie programem: żeby z niej skorzystać, trzeba napisać aplikację. W 2004 roku Yonik Seeley, pracujący w firmie CNET Networks, zrobił taką aplikację na własny użytek firmy, jako wyszukiwarkę dla serwisów CNET. Nazwał ją Solr i udostępnił Lucene przez zwykłe zapytania HTTP, więc korzystać z niej mógł każdy język programowania.

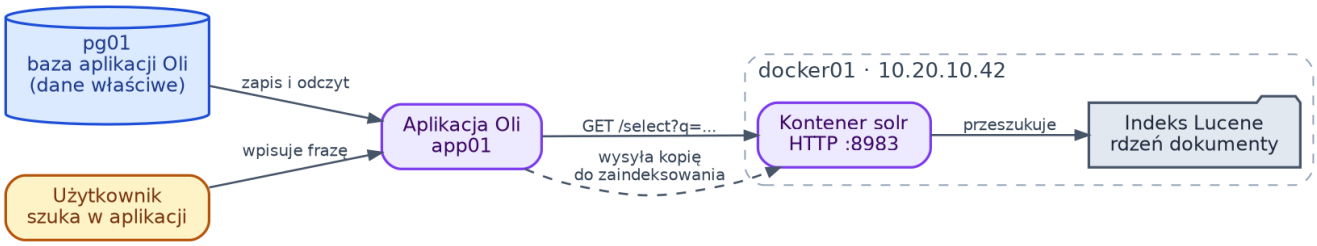
W styczniu 2006 roku CNET przekazał kod Fundacji Apache. W 2007 roku Solr dostał status samodzielnego projektu głównego, w 2010 roku zespoły Lucene i Solr połączyły siły, a w lutym 2021 roku Solr znów został osobnym projektem Apache. Wersja 10.0.0 ukazała się 3 marca 2026 roku.

Co instalujemy

Pojęcie: Apache Solr. Serwer wyszukiwania pełnotekstowego zbudowany na Lucene, z którym rozmawiamy przez HTTP. Wysyłamy do niego dokumenty, a potem pytamy o słowa, które w nich występują, i dostajemy odpowiedź w kilka milisekund.

Pojęcie: indeks odwrócony. Struktura, w której dla każdego słowa pamiętamy listę dokumentów, w których ono występuje, jak skorowidz na końcu książki. Dzięki niej szukanie słowa nie wymaga czytania wszystkich dokumentów, tylko zajrzenia do jednego wpisu.

Pojęcie: rdzeń (core) i kolekcja. Rdzeń to jeden osobny indeks ze swoim zestawem pól, odpowiednik jednej bazy w bazie relacyjnej. Kolekcja to ta sama rzecz w trybie klastra SolrCloud, gdzie jedna kolekcja może być podzielona na kilka rdzeni na różnych serwerach.



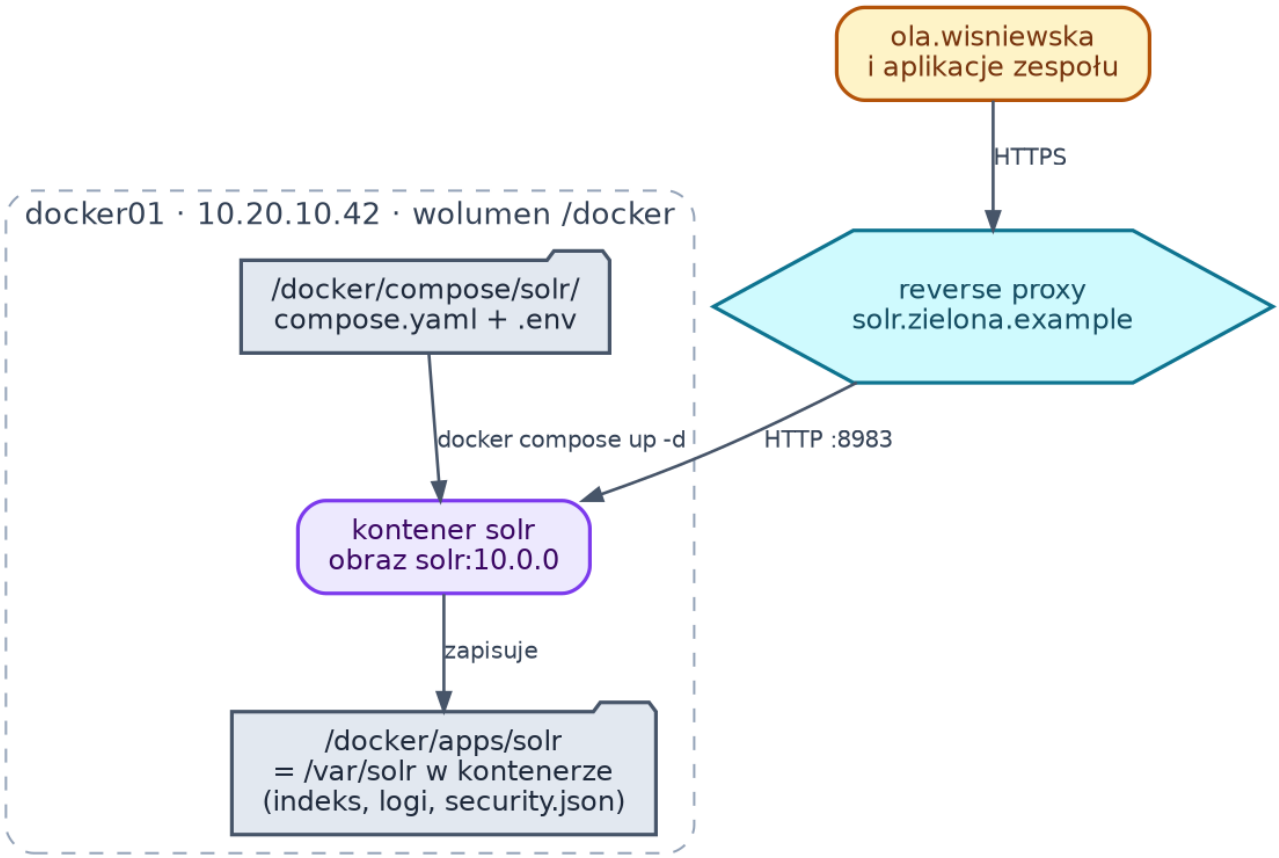
Solr w architekturze Zielonej: aplikacja Oli zapisuje dane w pg01, a kopię do szukania wysyła do Solra na docker01 (kliknięcie otwiera pełny rozmiar)

Użytkownik wpisuje frazę w aplikacji, aplikacja pyta Solra przez HTTP i dostaje listę identyfikatorów. Prawdziwe dane zostają w `pg01`, bo indeks można w każdej chwili odbudować z bazy, a z indeksu bazy nie odbudujemy.

> **Pojęcie: tryb SolrCloud i tryb user-managed.** Od wersji 10 polecenie `bin/solr start` domyślnie uruchamia Solra jako SolrCloud, z wbudowanym ZooKeeperem (osobnym programem, który w klastrze trzyma konfigurację) i z kolekcjami. Tryb user-managed, uruchamiany przełącznikiem `--user-managed`, jest klasycznym, samodzielnym Solrem z rdzeniami. Sposób, w jaki obraz Dockera 10.0.0 wybiera tryb, sprawdzamy po pierwszym starcie w poniższym kroku, bo od tego zależy kilka poleceń (tworzenie, kopia).

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen `vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał systemu.

Wymaganie	Uwaga
Docker Engine i wtyczka Compose	sprawdzamy `docker compose version`
Wolny port `8983` na hoście	domyślny port Solra
Około 2 GB wolnej pamięci dla kontenera	w pliku Compose ustawiamy stertę Javy na `1g`
Katalog `/docker/apps/solr` należący do użytkownika o numerze 8983	tym numerem pracuje proces Solra w kontenerze, patrz krok 1



Układ plików i ruchu: compose.yaml na docker01, kontener solr, katalog danych /docker/apps/solr jako /var/solr i reverse proxy przed Solrem (kliknięcie otwiera pełny rozmiar)

Obraz przypinamy do konkretnej wersji (`10.0.0`), nigdy do `latest`, a aktualną wersję sprawdzamy na Docker Hub przed każdą instalacją. Plik Compose nie zawiera haseł, więc osobny plik `.env` nie jest potrzebny: hasło użytkownika Solra ustawiamy poleceniem w kroku 11.

> Po instalacji Solr odpowiada każdemu, kto dotrze do portu `8983`, bez pytania o hasło. Zanim udostępnimy go aplikacji z `app01`, włączamy uwierzytelnianie (krok 11) i stawiamy przed nim reverse proxy z HTTPS, jak w Reverse proxy i równoważenie obciążenia.

Przebieg

1. Utworzenie katalogów i nadanie właściciela

```
sudo mkdir -p /docker/apps/solr /docker/compose/solr
sudo chown 8983:8983 /docker/apps/solr
```

Katalog danych musi należeć do użytkownika, którym pracuje Solr w kontenerze, inaczej kontener nie zapisze indeksu. Numer 8983 sprawdzamy poleceniem:

```
docker run --rm --entrypoint id solr:10.0.0
```

Przykład:

```
uid=8983(solr) gid=8983(solr) groups=8983(solr)
```

2. Plik Compose: compose.yaml

Plik `/docker/compose/solr/compose.yaml`:

```
services:
  solr:
    image: solr:10.0.0
    container_name: solr
    restart: always
    ports:
      - "8983:8983"
    environment:
      - SOLR_HEAP=1g
    volumes:
      - /docker/apps/solr:/var/solr
    ulimits:
      nofile:
        soft: 65000
        hard: 65000
```

Katalog `/docker/apps/solr` na hoście to `/var/solr` w kontenerze: tam leży indeks, logi i konfiguracja. Sekcja `ulimits` robi to samo co `LimitNOFILE` w systemd, o którym piszemy w Apache Solr: utrzymanie.

3. Pierwsze uruchomienie: docker compose up

```
cd /docker/compose/solr
docker compose up -d
docker compose logs -f solr
```

Pobranie obrazu zajmuje chwilę. Log przeglądamy, aż pojawi się informacja, że Solr nasłuchuje, i przerywamy go klawiszami Ctrl+C.

4. Sprawdzenie trybu pracy: admin/info/system

```
curl -s "http://localhost:8983/solr/admin/info/system?wt=json" | grep -o '"mode":"[a-z]*"'
```

Wartość `solrcloud` oznacza tryb klastra (kolekcje, ZooKeeper), a `std` tryb samodzielny (rdzenie). Zapamiętujemy wynik, bo wraca w kroku 5 i w kopii zapasowej.

Przykład:

```
"mode":"solrcloud"
```

5. Utworzenie rdzenia albo kolekcji: solr create

```
docker exec solr solr create -c dokumenty
```

W trybie samodzielnym polecenie tworzy rdzeń, a w SolrCloud kolekcję, w obu przypadkach o nazwie `dokumenty`.

Przykład (tryb samodzielny):

```
Created new core 'dokumenty'
```

Przykład (SolrCloud):

```
Created collection 'dokumenty' with 1 shard(s), 1 replica(s) with config-set 'dokumenty'
```

6. Zdefiniowanie pól: Schema API

```
curl -X POST -H 'Content-type:application/json' "http://localhost:8983/solr/dokumenty/schema" -d '{"add-field":[{"name":"tytul","type":"text_general","stored":true},{"name":"tresc","type":"text_general","stored":true},{"name":"dzial","type":"string","stored":true}]}'
```

Mówimy Solrowi, jakie pola mają nasze dokumenty. Pola `tytul` i `tresc` są tekstem dzielonym na słowa (`text_general`), a `dzial` jednym niepodzielnym napisem (`string`), po którym będziemy filtrować.

Przykład:

```
{
  "responseHeader":{
    "status":0,
    "QTime":87}}
```

7. Przygotowanie przykładowych dokumentów: dokumenty.json

Plik `/docker/compose/solr/dokumenty.json`:

```
[
  {"id": "1", "tytul": "Regulamin magazynu w Swarzędzu", "tresc": "Zmiana zaczyna się o 6:00. Każdy pracownik loguje skaner przed wydaniem towaru.", "dzial": "magazyn"},
  {"id": "2", "tytul": "Instrukcja logowania skanera", "tresc": "Skaner loguje się do systemu WMS. Gdy skaner nie widzi serwera, zgłaszamy to kierownikowi zmiany.", "dzial": "magazyn"},
  {"id": "3", "tytul": "Procedura zamknięcia miesiąca", "tresc": "Księgowość kończy ewidencję faktur do trzeciego dnia roboczego i wysyła raport do prezesa.", "dzial": "ksiegowosc"},
  {"id": "4", "tytul": "Zasady pracy zdalnej handlowców", "tresc": "Handlowiec łączy się z siecią firmy przez VPN. Faktury drukujemy w centrali.", "dzial": "handel"}
]
```

Każdy dokument ma unikalne `id`. Wysłanie dokumentu z już istniejącym `id` zastępuje stary, więc ładowanie można bezpiecznie powtarzać.

8. Załadowanie dokumentów do Solra: curl update

```
curl -X POST -H 'Content-type:application/json' --data-binary @dokumenty.json
"http://localhost:8983/solr/dokumenty/update?commit=true"
```

Parametr `commit=true` każe Solrowi od razu zatwierdzić zmiany, żeby dokumenty były widoczne w wynikach. Przy dużych ładowaniach commit robimy raz na końcu, nie przy każdej porcji.

Przykład:

```
{
  "responseHeader":{
    "status":0,
    "QTime":112}}
```

Dokumenty można też załadować narzędziem dołączonym do Solra, które samo rozpoznaje typ pliku:

```
docker exec solr solr post -c dokumenty /var/solr/dokumenty.json
```

Ta wersja wymaga, żeby plik leżał wewnątrz kontenera, więc najpierw kopiujemy go do `/docker/apps/solr` na hoście. Opis opcji narzędzia `post` sprawdzamy w dokumentacji Solra dla naszej wersji.

9. Pierwsze wyszukiwanie: select

```
curl "http://localhost:8983/solr/dokumenty/select?q=tresc:skaner&fl=id,tytul&rows=5"
```

(gdzie `q` to zapytanie, `fl` lista pól w odpowiedzi, a `rows` maksymalna liczba wyników)

Przykład:

```
{
  "responseHeader":{
    "status":0,
    "QTime":4,
    "params":{
      "q":"tresc:skaner",
      "fl":"id,tytul",
      "rows":"5"}},
  "response":{"numFound":2,"start":0,"numFoundExact":true,"docs":[
      {
        "id":"1",
```

```
        "tytul":"Regulamin magazynu w Swarzędzu"},
      {
        "id":"2",
        "tytul":"Instrukcja logowania skanera"}]
  }}
```

Oba dokumenty ze słowem „skaner" w treści wracają w kilka milisekund. Wielkość liter nie ma znaczenia, bo Solr normalizuje tekst przy indeksowaniu.

10. Zawężenie wyników do jednego działu: fq

```
curl "http://localhost:8983/solr/dokumenty/select?q=*:*&fq=dzial:handel&fl=id,tytul"
```

Parametr `fq` to filtr, który zawęża wyniki, ale nie wpływa na ich kolejność. Zwraca tu jeden dokument, numer 4. Filtry po niepodzielnych polach, jak `dzial`, są szybkie, bo Solr zapamiętuje ich wyniki.

11. Włączenie uwierzytelniania Basic Auth: solr auth enable

```
docker exec solr solr auth enable --type basicAuth --credentials nazwa_usera:HasloDoSolra
```

Od tej chwili każde zapytanie bez hasła kończy się odpowiedzią `401`, więc do kolejnych poleceń `curl` dodajemy `--user nazwa_usera`. Jeśli polecenie zgłosi brak adresu ZooKeepera albo pliku `solr.in.sh`, sprawdzamy w dokumentacji rozdział Basic Authentication Plugin, jakich opcji wymaga nasz tryb. Przełączanie, zmianę hasła przez API i diagnozę błędu `access denied` opisujemy w Apache Solr: utrzymanie.

Sprawdzamy, że zamek działa:

```
curl -i "http://localhost:8983/solr/dokumenty/select?q=*:*"
```

Przykład:

```
HTTP/1.1 401 Unauthorized
```

12. Katalog na kopie zapasowe

```
sudo install -d -o 8983 -g 8983 /docker/apps/solr/data/kopie
```

Kopie z API zapisujemy pod `/var/solr/data/kopie` w kontenerze, czyli w katalogu domowym Solra. Solr domyślnie pozwala pisać i czytać tylko pod swoim katalogiem domowym, więc ścieżka spoza niego kończy się błędem. Z hosta ten katalog zabiera potem NetWorker na `bkp01`.

13. Kopia rdzenia przez API replikacji: command=backup

Dla trybu samodzielnego (`"mode":"std"`):

```
curl --user nazwa_usera
"http://localhost:8983/solr/dokumenty/replication?command=backup&location=/var/solr/data/kopie
&name=dokumenty-2026-10-03&numberToKeep=7"
```

(gdzie `name` to nazwa kopii, a `numberToKeep` liczba kopii, które zostawiamy, starsze znikają)

Przebieg kopii sprawdzamy poleceniem:

```
curl --user nazwa_usera
"http://localhost:8983/solr/dokumenty/replication?command=details&wt=json"
```

W odpowiedzi szukamy w sekcji `backup` statusu `success`.

14. Kopia kolekcji w SolrCloud: action=BACKUP

Dla trybu klastra (`"mode":"solrcloud"`) kopię robi Collections API, a nie replikacja:

```
curl --user nazwa_usera
"http://localhost:8983/solr/admin/collections?action=BACKUP&name=dokumenty-2026-10-
03&collection=dokumenty&location=/var/solr/data/kopie"
```

Każdy katalog kopii może zawierać kopię tylko jednej kolekcji, więc dla kolejnych kolekcji zakładamy osobne podkatalogi.

15. Znacznik przed ryzykowną zmianą: CREATESNAPSHOT

```
curl --user nazwa_usera
"http://localhost:8983/solr/admin/cores?action=CREATESNAPSHOT&core=dokumenty&commitName=przed-
aktualizacja"
```

Snapshot to nazwany znacznik na konkretnym stanie indeksu, a nie osobna kopia plików: chroni ten stan przed usunięciem przy scalaniu segmentów. Samą kopię robimy z niego potem poleceniem `command=backup` z dodanym `commitName=przed-aktualizacja`. W SolrCloud analogiczne polecenie wywołujemy w Collections API.

16. Odtworzenie rdzenia z kopii: command=restore

```
curl --user nazwa_usera
"http://localhost:8983/solr/dokumenty/replication?command=restore&name=dokumenty-2026-10-
03&location=/var/solr/data/kopie"
```

Postęp sprawdzamy poleceniem `command=restorestatus`, a wynik `success` oznacza koniec. Odtwarzamy tylko po zatrzymaniu aplikacji, która pisze do indeksu, bo w trakcie odtwarzania Solr odpowiada niespójnie.

17. Aktualizacja do nowszej wersji: docker compose pull

Najpierw robimy kopię (kroki 13 albo 14), potem zmieniamy numer wersji w `compose.yaml` i wykonujemy:

```
cd /docker/compose/solr
docker compose pull
docker compose up -d
```

Przy aktualizacji w obrębie tej samej wersji głównej (np. 10.0.0 do 10.0.1) zmiana jest zwykle bezbolesna. Przy skoku między wersjami głównymi czytamy najpierw Upgrade Notes: Lucene czyta indeksy z poprzedniej wersji głównej, ale nie z dwóch wstecz, więc stary indeks trzeba wtedy

zbudować od nowa z bazy.

Weryfikacja

Sprawdzamy, że kontener działa:

```
docker compose ps
```

Przykład:

```
NAME      IMAGE          STATUS          PORTS
solr      solr:10.0.0    Up 5 minutes    0.0.0.0:8983->8983/tcp
```

Sprawdzamy, że rdzeń odpowiada (przed włączeniem uwierzytelniania, albo z `--user nazwa_usera` po nim):

```
curl -s "http://localhost:8983/solr/dokumenty/admin/ping"
```

Przykład:

```
{
  "responseHeader":{
    "status":0,
    "QTime":1},
  "status":"OK"}
```

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: panel administracyjny Solra pod `http://docker01:8983/solr/`, wybrany rdzeń `dokumenty` z lewego menu, zakładka „Query" z wykonanym zapytaniem `q=tresc:skaner` i widocznym wynikiem `numFound: 2`.

Wycofanie zmian

Zatrzymanie i usunięcie kontenera, bez kasowania indeksu:

```
cd /docker/compose/solr
docker compose down
```

Pełne usunięcie razem z danymi, tylko gdy naprawdę zaczynamy od zera:

```
docker compose down
sudo rm -rf /docker/apps/solr
```

Powrót do poprzedniej wersji po nieudanej aktualizacji wymaga kopii: przywracamy stary numer wersji w `compose.yaml` i odtwarzamy rdzeń z kopii z kroku 13 albo 14.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
Kontener restartuje się, w logu `Permission denied` na `/var/solr`	katalog `/docker/apps/solr` nie należy do 8983 albo blokuje go SELinux	`sudo chown 8983:8983 /docker/apps/solr`, sprawdzamy `getenforce`, w razie potrzeby dopisujemy `:z` do wolumenu
`curl: (7) Connection refused` zaraz po `up -d`	Solr startuje kilkanaście sekund	`docker compose logs -f solr`, ponawiamy po komunikacie o nasłuchu
`unknown field 'tytul'` przy ładowaniu	pola nie zostały zdefiniowane	wykonujemy krok 6 przed krokiem 8
`401 Unauthorized` w każdym poleceniu	włączone uwierzytelnianie	dopisujemy `--user nazwa_usera`
`solr create` zgłasza, że rdzeń już istnieje	rdzeń o tej nazwie jest od poprzedniej próby	wybieramy inną nazwę albo usuwamy stary rdzeń
Kopia przez replikację nie działa w SolrCloud	tryb klastra używa Collections API	używamy kroku 14 zamiast 13
Błąd o niedozwolonej ścieżce przy kopii	`location` poza katalogiem domowym Solra	zapisujemy pod `/var/solr/data/kopie`
`Too many open files` w logu Solra	zbyt niski limit plików kontenera	podnosimy `nofile` w sekcji `ulimits` w `compose.yaml`

Objaw	Przyczyna	Co robimy
Port `8983` zajęty	inny proces na hoście	`ss -tlnp \

□□ Ćwiczenia

Ćwiczenie 1: dodanie dokumentu

Dopisujemy do `dokumenty.json` piąty dokument o dziale `handel`, ładujemy go ponownie i sprawdzamy, że `fq=dzial:handel` zwraca teraz dwa wyniki.

Odpowiedź

Dopisujemy obiekt z `"id": "5"` i `"dzial": "handel"`, a potem powtarzamy krok 8. Dokumenty o istniejących numerach zostają zastąpione identycznymi, a nowy pojawia się obok, więc zapytanie z krokiem 10 zwraca dokumenty 4 i 5.

Ćwiczenie 2: tryb pracy

Po co sprawdzamy tryb Solra w kroku 4, zanim zaczniemy robić kopie zapasowe?

Odpowiedź

Bo w trybie samodzielnym kopię rdzenia robimy przez API replikacji, a w SolrCloud kopię kolekcji przez Collections API. Użycie niewłaściwego polecenia kończy się błędem albo kopią, której nie da się odtworzyć.

Ćwiczenie 3: gdzie są dane

Ola pyta, czy może przestać trzymać treść instrukcji w bazie `pg01`, skoro wszystko jest już w Solrze. Co odpowiadamy?

Odpowiedź

Nie. Solr jest kopią do szukania, a nie miejscem na jedyny egzemplarz danych: indeks można odbudować z bazy, ale bazy z indeksu już nie. Prawdziwe dane zostają w `pg01`, a Solr dostaje ich kopię.

Powiązane: Apache Solr: utrzymanie · Generowanie certyfikatów: OpenSSL i urząd certyfikacji Windows · Reverse proxy i równoważenie obciążenia · Instalacja Gitea w Dockerze

Źródła: solr.apache.org/guide, rozdziały „Solr in Docker", „Solr Control Script Reference", „Basic Authentication Plugin", „Backup and Restore", „Collection Management", „Major Changes in Solr 10" · hub.docker.com/_/solr (tagi, sprawdzono 3.10.2026) · en.wikipedia.org, hasła „Apache Lucene" i „Apache Solr"