Hasła i sekrety poza repozytorium: .env, Docker secrets i Vaultwarden

Poziom: referencyjny **Dotyczy: wszystkie aplikacje na docker01** **Czas: ~25 min**

Plik `.env` obok każdego `compose.yaml` już pojawił się na każdej stronie instalacyjnej w tej książce. Ta strona zbiera w jednym miejscu, dlaczego hasła stoją akurat tam, czym różni się od tego mechanizm `secrets` w Compose, i gdzie w Zielonej trzymamy hasła do samych tych plików.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy

Fiszka

TYLKO ODCZYT

Sprawdzenie praw pliku z hasłami: ls -l

```
ls -l /docker/compose/NazwaAplikacji/.env
```

Sprawdzamy, że plik ma prawa `600`, zanim uznamy wdrożenie nowej aplikacji za skończone.

Przykład:

```
docker01:~$ ls -l /docker/compose/netbox/.env
-rw-------. 1 root root 214 wrz 28 09:12 /docker/compose/netbox/.env
```

ls = list: wypisz zawartość katalogu

ZMIENIA SYSTEM

Ograniczenie praw do pliku z hasłami: chmod 600

```
chmod 600 /docker/compose/NazwaAplikacji/.env
```

Odczyt i zapis wyłącznie dla właściciela pliku, opisane już przy konwencji Dockera w Zielonej na stronie Jak trzymamy aplikacje w Dockerze: konwencje Zielonej.

chmod = change mode: zmień prawa dostępu

TYLKO ODCZYT

Sprawdzenie, czy .env trafił przypadkiem do repozytorium: git status

```
git status --ignored
```

Sprawdzamy przed pierwszym commitem nowego katalogu Compose, że `.env` znalazł się w sekcji „Ignored files”, nie w „Changes to be committed”.

Przykład:

```
Ignored files:
  compose/netbox/.env
```

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

TYLKO ODCZYT

Wyszukanie sekretu w historii repozytorium: git log -p -- .env

```
git log -p --all -- '**/.env'
```

Uruchamiamy tę komendę na całym repozytorium konfiguracji po każdym podejrzeniu, że `.env` mógł trafić do commita, zanim ktoś dodał go do `.gitignore`. Pusty wynik potwierdza, że plik nigdy nie był w historii, nie tylko że nie jest w niej teraz.

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

Z życia wzięte

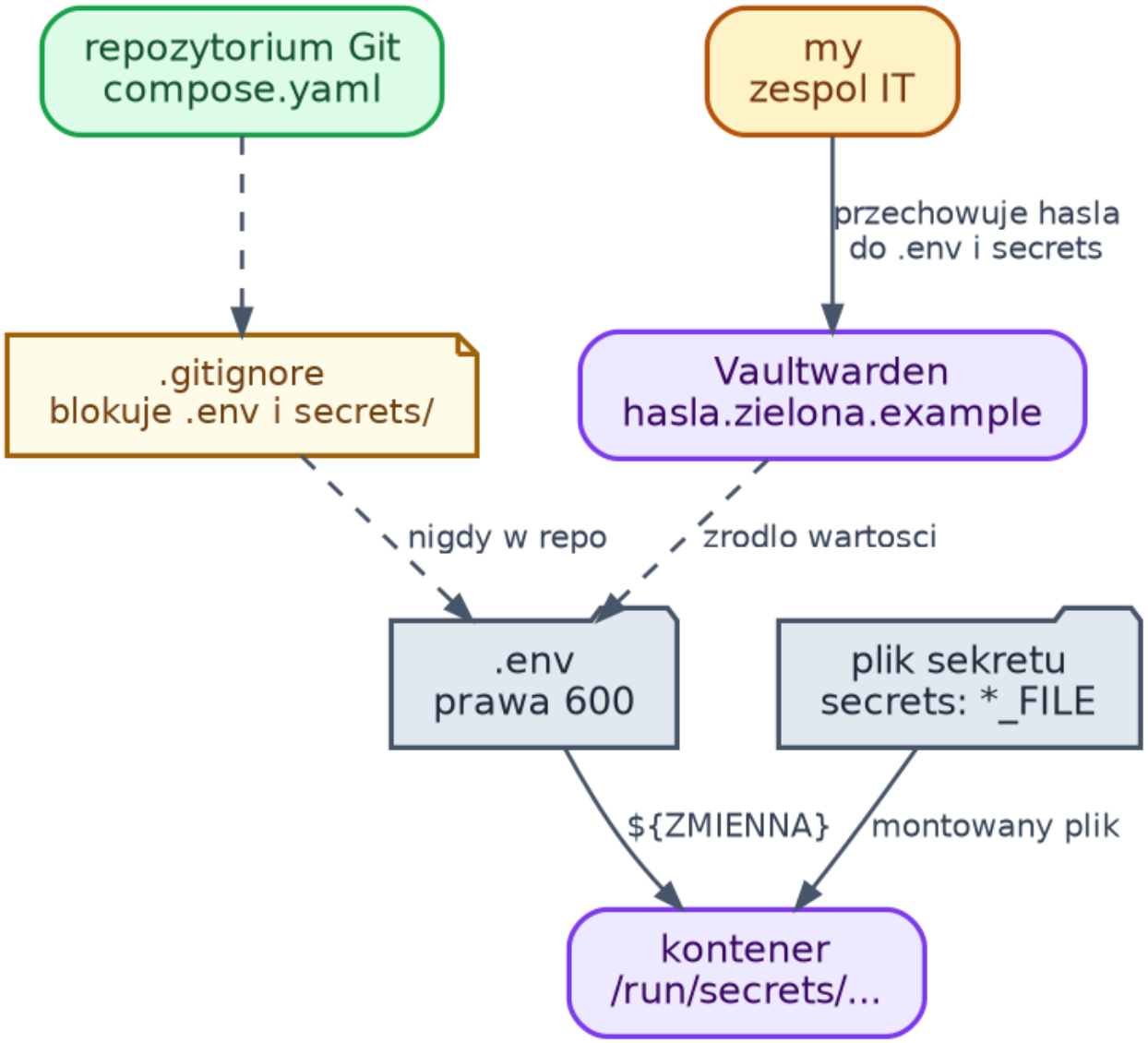
Z ŻYCIA WZIĘTE

Piątek, 14:40. Ola commituje nowy `compose.yaml` swojego zespołu razem z sąsiadującym plikiem `.env`, bo `git add .` w pustym repozytorium wzięło oba naraz, a `.gitignore` jeszcze nie istniał. Push trafia do wspólnego repozytorium konfiguracji na `git.zielona.example`, widocznego dla całego zespołu programistów. Dwadzieścia minut później Ola sama to zauważa przy przeglądaniu historii commitów i zgłasza to nam, zamiast czekać, aż ktoś inny to znajdzie.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: sekret. Każda wartość, która daje dostęp do czegoś: hasło do bazy danych, token API, klucz prywatny certyfikatu. W odróżnieniu od zwykłej konfiguracji (numer portu, nazwa hosta), sekret nigdy nie powinien trafić tam, gdzie trafia zwykła konfiguracja: do repozytorium Git, do logów, do zmiennych środowiskowych widocznych każdemu procesowi na hoście przez `/proc`.



Compose.yaml w repozytorium, .env i plik sekretu poza nim, Vaultwarden jako miejsce, gdzie zespół IT trzyma hasla do obu (kliknięcie otwiera pełny rozmiar)

Dlaczego nie w compose.yaml i nie w Gicie

Plik `compose.yaml` w Zielonej trafia do repozytorium konfiguracji: ma historię zmian, każdy w zespole widzi, co się w nim zmieniło i kiedy, a nowa osoba w zespole odtwarza całe środowisko przez `git clone`. Hasło wpisane wprost w `compose.yaml` dziedziczy dokładnie te same właściwości, tylko dla czegoś, co ma być tajne: zostaje w historii commitów na zawsze, nawet

po usunięciu z najnowszej wersji pliku, i widzi je każdy, kto ma dostęp do repozytorium, niezależnie od tego, czy faktycznie potrzebuje znać to konkretne hasło.

Rozdzielenie na `compose.yaml` (do repozytorium) i `.env` (poza nim) rozwiązuje to jednym podziałem: przepis na uruchomienie aplikacji jest jawny i wspólny dla całego zespołu, a wartości sekretów zostają lokalnie na hoście, gdzie aplikacja faktycznie działa.

Plik .env i podstawienie zmiennych

```
services:
  server:
    image: netboxcommunity/netbox:v4.1.4
    environment:
      - DB_PASSWORD=${DB_PASSWORD}
```

```
DB_PASSWORD=HasloBazyNetboksa
```

Compose podstawia wartość zmiennej `DB_PASSWORD` z pliku `.env` leżącego w tym samym katalogu co `compose.yaml`, automatycznie, bez dodatkowej flagi. Sam plik `compose.yaml`, bez wartości sekretu w środku, może bez obaw trafić do repozytorium: ujawnia strukturę konfiguracji (że aplikacja potrzebuje hasła do bazy), ale nie samą wartość.

> ⚠ Zmienna podstawiona przez `.env` trafia do środowiska procesu wewnątrz kontenera, widoczna każdemu, kto ma dostęp do `docker inspect` na tym kontenerze albo do odczytu `/proc/PID/environ` na hoście. To wystarczające zabezpieczenie przed przypadkowym trafieniem hasła do Gita, ale nie przed każdym, kto ma dostęp do samego `docker01`.

Docker secrets: plik zamiast zmiennej środowiskowej

> **Pojęcie: Docker secrets.** Mechanizm Compose, w którym sekret trafia do kontenera jako plik pod `/run/secrets/<nazwa>`, zamiast jako zmienna środowiskowa. Działa też bez trybu Swarm, wystarczy zwykły `docker compose up`.

```
services:
  server:
    image: netboxcommunity/netbox:v4.1.4
    secrets:
      - db_password

secrets:
  db_password:
    file: ./sekrety/db_password.txt
```

```
mkdir -p /docker/compose/netbox/sekrety
echo -n "HasloBazyNetboksa" > /docker/compose/netbox/sekrety/db_password.txt
chmod 600 /docker/compose/netbox/sekrety/db_password.txt
```

Plik `db_password.txt` trafia do repozytorium tak samo pominięty przez `.gitignore` jak `.env`. Różnica wobec zwykłej zmiennej środowiskowej jest praktyczna: sekret jako plik nie pojawia się w wyniku `docker inspect` kontenera i nie jest widoczny w liście zmiennych środowiskowych procesu, tylko w treści jednego konkretnego pliku, do którego dostęp da się ograniczyć prawami systemu plików wewnątrz kontenera.

> **Pojęcie: konwencja `*_FILE`.** Część oficjalnych obrazów (na przykład PostgreSQL, MariaDB) obsługuje dodatkową zmienną środowiskową z przyrostkiem `_FILE`, wskazującą ścieżkę do pliku z sekretem, zamiast samej wartości:

```
environment:
  - POSTGRES_PASSWORD_FILE=/run/secrets/db_password
secrets:
  - db_password
```

To nie jest zachowanie samego Dockera, tylko skryptu startowego konkretnego obrazu: sprawdzamy w dokumentacji obrazu, czy dana zmienna ma odpowiednik z `_FILE`, zanim założymy, że zadziała. Obraz, który tego nie obsługuje, dostaje po prostu nieużywaną zmienną `_FILE` i dalej oczekuje wartości wprost.

Gdzie trzymamy hasła do tych plików: Vaultwarden

Plik `.env` i pliki sekretów rozwiązują, jak hasło trafia do kontenera. Nie rozwiązują, skąd bierze się samo hasło, gdy ktoś z zespołu zakłada nową aplikację, zmienia hasło po wycieku, albo drugi administrator musi odtworzyć `.env` na nowym hoście po awarii. Te wartości w Zielonej trzymamy w Vaultwardenie, opisanym w książce Vaultwarden: sejf na hasła, który trzymamy u siebie, w osobnym elemencie sejfu na każdą aplikację, z nazwą pliku i ścieżką w notatce.

> Hasło do bazy danych nowej aplikacji generujemy od razu w Vaultwardenie, wklejamy wygenerowaną wartość do `.env` na hoście i tam kończymy: sam Vaultwarden nigdy nie generuje pliku `.env` za nas ani go nie synchronizuje, jest tylko miejscem, gdzie wartość jest bezpiecznie zapisana do odtworzenia później.

Rotacja hasła

Rotacja sekretu to zawsze trzy kroki w tej kolejności: zmiana hasła w samej aplikacji albo bazie danych, aktualizacja wartości w `.env` albo w pliku sekretu na hoście, `docker compose up -d`, żeby kontener wstał z nową wartością w środowisku. Zapisanie nowej wartości w Vaultwardenie, z odnotowaniem daty rotacji w notatce elementu, zamyka procedurę. Kolejność ma znaczenie: zmiana samego `.env` bez zmiany hasła w aplikacji zostawia kontener z wartością, która nigdzie już nie pasuje, i aplikacja przestaje się łączyć z bazą po najbliższym restarcie.

Co robimy po wycieku do repozytorium

Sam `git rm .env` i nowy commit nie usuwają pliku z historii: poprzedni commit z pełną zawartością `.env` dalej jest dostępny każdemu, kto ma dostęp do repozytorium, przez `git log` albo `git checkout` na starszy commit. Kolejność postępowania po wykryciu wycieku:

1. Zmieniamy każde hasło, które było w wyciekłym pliku, od razu, niezależnie od tego, czy repozytorium jest publiczne, czy tylko wewnętrzne: samą obecność w historii traktujemy jak potencjalne ujawnienie.
2. Dopiero po zmianie haseł czyścimy historię repozytorium (na przykład narzędziem `git filter-repo`), żeby usunąć commit z wartościami, które i tak już straciły ważność.
3. Dopisujemy `.env` i `secrets/` do `.gitignore`, jeśli jeszcze tam nie ma, i sprawdzamy `git status --ignored` na każdym innym katalogu Compose w repozytorium, czy problem nie powtarza się gdzie indziej.
4. Zapisujemy nowe hasła w Vaultwardenie i odnotowujemy datę oraz przyczynę rotacji w notatce elementu.

>

> ⚠ Krok pierwszy zawsze przed drugim. Czyszczenie historii repozytorium bez wcześniejszej zmiany haseł zostawia stare, wciąż działające wartości w każdej lokalnej kopii repozytorium, którą ktokolwiek już wcześniej pobrał, łącznie z kopiami, o których zespół IT nie wie.

Częste błędy

Objaw	Przyczyna	Co robimy
`.env` widoczny w `git status` jako plik do commitu	brak wpisu `.env` w `.gitignore` przy zakładaniu nowego katalogu Compose	dopisujemy `.env` i `secrets/` do `.gitignore`, zanim wykonamy pierwszy `git add`
Hasło z `.env` czytelne dla każdego konta na `docker01`	plik ma domyślne prawa `644` zamiast `600`	`chmod 600` na pliku
Zmienna `POSTGRES_PASSWORD_FILE` nie działa, aplikacja i tak pyta o hasło	obraz aplikacji nie obsługuje konwencji `_FILE` dla tej konkretnej zmiennej	sprawdzamy dokumentację obrazu, wracamy do zwykłej zmiennej z `.env`, jeśli `_FILE` nie jest wspierane
Po zmianie hasła w Vaultwardenie aplikacja przestaje się łączyć z bazą	zmieniono wartość tylko w Vaultwardenie, bez aktualizacji `.env` na hoście i bez `docker compose up -d`	aktualizujemy `.env`, restart kontenera, dopiero potem hasło w Vaultwardenie traktujemy jako aktualne

□□ Ćwiczenia

1. Zakładamy nową aplikację testową z plikiem `.env` i sprawdzamy trzema komendami z tej strony, że hasło ma prawa `600`, nie trafiło do repozytorium i nie ma go w historii commitów.
2. Przepisujemy fragment `environment` na mechanizm `secrets` z plikiem, dla zmiennej `DB_PASSWORD` istniejącej aplikacji.
3. Kolega z zespołu commituje `.env` z prawdziwym hasłem do wspólnego repozytorium i sam to zauważa po dwudziestu minutach. Wymieniamy kroki postępowania w kolejności, w jakiej je wykonujemy.

Odpowiedź

Najpierw zmieniamy każde hasło, które było w tym pliku, niezależnie od tego, jak krótko wisiało w repozytorium. Dopiero potem czyścimy historię repozytorium z commita zawierającego stare

wartości. Na końcu dopisujemy `.env` do `.gitignore`, jeśli go tam brakowało, i zapisujemy nowe hasła w Vaultwardenie z notatką o przyczynie rotacji.

Powiązane: Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Docker Compose · Vaultwarden: sejf na hasła, który trzymamy u siebie

Źródła: docs.docker.com, rozdział „Manage sensitive data with Docker Compose" · docs.docker.com, rozdział „Secrets in Compose"

Revision #22
Created 2026-09-28 17:15:49 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:37 UTC by Tomasz Konieczny