Kopia zapasowa i aktualizacja Vaultwardena

Poziom: operacyjny
Dotyczy: Vaultwarden 1.37.3 w Dockerze, baza SQLite na wolumenie **Czas: ~35 min**

Cała zawartość sejfu Zielonej mieści się w jednym pliku SQLite i kilku katalogach na wolumenie. Zwykłe skopiowanie tego pliku podczas pracy kontenera potrafi dać uszkodzoną kopię, więc kopię bazy robimy narzędziem `sqlite3` w trybie `.backup`, nie poleceniem `cp`. Aktualizacja to podmiana tagu obrazu, ale bez świeżej kopii sprzed niej wycofanie zmian po migracji bazy jest bardzo trudne.

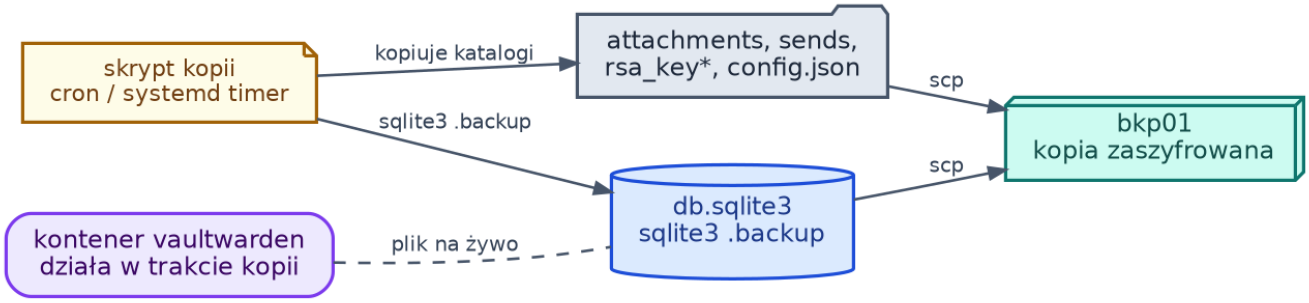
Skróty użyte na tej stronie (2)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
SQLite	mała baza danych zapisana w jednym pliku; Home Assistant trzyma w niej historię w pliku home-assistant_v2.db

Zanim zaczniemy

Wymaganie	Uwaga
Pakiet `sqlite3` na `docker01`	`sudo dnf install -y sqlite`, sprawdzamy `sqlite3 --version`
Dostęp do katalogu `/docker/apps/vaultwarden`	konto z prawem odczytu wolumenu kontenera na hoście
Miejsce na `bkp01` na regularne kopie	patrz Świat przykładów: Zielona po adres i rolę serwera
Znajomość obecnego tagu obrazu	`docker compose images` w `/docker/compose/vaultwarden/`

Cała droga danych, od pliku na wolumenie do serwera kopii:



Kopia bazy SQLite przez sqlite3 .backup i osobno skopiowane katalogi, wysłane na bkp01 (kliknięcie otwiera pełny rozmiar)

Uważaj: aktualizacja Vaultwardena migruje schemat bazy SQLite przy pierwszym starcie nowej wersji, podobnie jak Gitea migruje swoją bazę PostgreSQL. Migracja jest jednokierunkowa: cofnięcie tagu obrazu do starszej wersji po tym, jak migracja już przeszła, zostawia starego Vaultwardena wobec schematu bazy, którego nie rozumie. Jedyna pewna droga wycofania po migracji to odtworzenie z kopii sprzed aktualizacji.

Przebieg

1. Dlaczego nie kopiujemy pliku na żywo

> **Pojęcie: WAL.** Tryb dziennika zapisu w SQLite, w którym świeże zmiany trafiają najpierw do osobnego pliku `db.sqlite3-wal`, a do głównego pliku bazy wracają dopiero przy okazji punktu kontrolnego. Kontener Vaultwarden trzyma bazę otwartą cały czas, więc zwykłe `cp db.sqlite3 kopia.sqlite3` w dowolnym momencie może skopiować sam plik główny bez świeżych zmian z `db.sqlite3-wal`, dając kopię, która wygląda poprawnie, ale brakuje w niej najnowszych zapisów, albo w gorszym wypadku kopię niespójną w środku transakcji.

Polecenie `.backup` narzędzia `sqlite3` korzysta z tego samego mechanizmu blokad co sama baza, więc bezpiecznie czyta spójny stan danych nawet wtedy, gdy Vaultwarden akurat coś zapisuje. Nie musimy w tym celu zatrzymywać kontenera.

2. Wykonanie kopii bazy: sqlite3 .backup

```
mkdir -p /docker/apps/vaultwarden/kopie
sqlite3 /docker/apps/vaultwarden/db.sqlite3 ".backup '/docker/apps/vaultwarden/kopie/db-2026-09-28.sqlite3'"
```

Przykład sprawdzenia, że plik kopii rzeczywiście powstał i ma sensowny rozmiar:

```
ls -lh /docker/apps/vaultwarden/kopie/
```

Przykład:

```
-rw-r--r-- 1 root root 2.3M Sep 28 03:00 db-2026-09-28.sqlite3
```

3. Kopia pozostałych katalogów i plików

```
tar -czf /docker/apps/vaultwarden/kopie/dane-2026-09-28.tar.gz \
  -C /docker/apps/vaultwarden attachments sends rsa_key.pem rsa_key.pub.pem config.json
```

`attachments` i `sends` trzymają załączniki dodane do haseł i pliki wysłane funkcją Send. Pliki `rsa_key*` to klucze, którymi Vaultwarden podpisuje tokeny sesji: bez nich po odtworzeniu wszystkie istniejące sesje i część integracji trzeba by zestawić od nowa. `config.json` trzyma ustawienia zapisane z poziomu panelu `/admin`, niepokrywające się z tym, co stoi w `compose.yaml`.

4. Szyfrowanie kopii przed wysyłką

```
openssl enc -aes-256-cbc -pbkdf2 -salt \
  -in /docker/apps/vaultwarden/kopie/dane-2026-09-28.tar.gz \
  -out /docker/apps/vaultwarden/kopie/dane-2026-09-28.tar.gz.enc \
  -pass file:/root/.vaultwarden-kopia-haslo
```

Kopia sejfu haseł to sam w sobie najbardziej wrażliwy plik w firmie: zawiera zaszyfrowane hasła wszystkich osób, ale też klucze `rsa_key*` w postaci jawnej. Szyfrujemy archiwum osobnym hasłem, zanim trafi na `bkp01`, i trzymamy to hasło poza samym `bkp01`, na przykład w innym Vaultwardenie albo w sejfie fizycznym.

5. Wysyłka na bkp01

```
scp /docker/apps/vaultwarden/kopie/db-2026-09-28.sqlite3
nazwa_usera@bkp01:/backup/vaultwarden/
scp /docker/apps/vaultwarden/kopie/dane-2026-09-28.tar.gz.enc
nazwa_usera@bkp01:/backup/vaultwarden/
```

6. Harmonogram: cron albo timer systemd

Wybieramy jeden z dwóch mechanizmów opisanych osobno, cron: zadania cykliczne albo systemd timery: OnCalendar zamiast crona, i spinamy nim kroki 2 do 5 w jeden skrypt uruchamiany co noc.

```
crontab -e
```

Przykład:

```
0 3 * * * /docker/apps/vaultwarden/skrypty/kopia-nocna.sh >> /var/log/vaultwarden-kopia.log
2>&1
```

> W Zielonej nocną kopię Vaultwardena robi osobne zadanie restic do repozytorium `vaultwarden` z poziomem retencji 365 dni (decyzja autora z 11 października 2026), opisane w kroku 7 strony Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach, a poziomy i uzasadnienie są na stronie Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje. Zadanie używa tego samego `sqlite3 .backup`, a repozytorium restic jest szyfrowane, więc osobne `openssl enc` nie jest tam potrzebne. Kroki 2 do 5 z tej strony zostają do kopii ręcznej przed aktualizacją, a cron z kroku 6 włączamy tylko tam, gdzie nie ma zadania restic, żeby dwa mechanizmy nie kopiowały tych samych danych.

7. Test odtworzenia

Raz na kwartał odtwarzamy najnowszą kopię na osobnej, testowej instalacji tej samej wersji Vaultwardena: zatrzymanie testowego kontenera, podmiana `db.sqlite3` i katalogów `attachments`/`sends`/`rsa_key*`/`config.json` na te z kopii, ponowne uruchomienie, logowanie testowym kontem. Kopia, której nikt nigdy nie spróbował odtworzyć, jest tylko przypuszczeniem, że działa.

```
docker compose -f /docker/compose/vaultwarden-test/compose.yaml down
cp /docker/apps/vaultwarden/kopie/db-2026-09-28.sqlite3 /docker/apps/vaultwarden-
test/db.sqlite3
tar -xzf /docker/apps/vaultwarden/kopie/dane-2026-09-28.tar.gz -C /docker/apps/vaultwarden-
test
docker compose -f /docker/compose/vaultwarden-test/compose.yaml up -d
```

Aktualizacja

1. Przeczytanie informacji o wydaniu

Przed zmianą tagu sprawdzamy release notes na `github.com/dani-garcia/vaultwarden/releases` pod kątem zmian wymagających ręcznej ingerencji: usuniętych zmiennych środowiskowych, zmian w formacie `ADMIN_TOKEN` albo ostrzeżeń o migracji bazy.

2. Wykonanie kopii sprzed aktualizacji

Powtarzamy kroki 2 do 5 z sekcji wyżej, poza harmonogramem, bezpośrednio przed aktualizacją: to jest kopia, do której wracamy w razie problemu, świeższa niż ostatnia nocna.

3. Zmiana tagu obrazu

`/docker/compose/vaultwarden/.env`:

```
VAULTWARDEN_TAG=1.38.0
```

`/docker/compose/vaultwarden/compose.yaml`, fragment:

```
services:
  vaultwarden:
    image: vaultwarden/server:${VAULTWARDEN_TAG}
```

```
docker compose -f /docker/compose/vaultwarden/compose.yaml pull
docker compose -f /docker/compose/vaultwarden/compose.yaml up -d
```

Przykład:

```
[+] Running 1/1
 ✔ Container vaultwarden  Started
```

Migracja schematu bazy wykonuje się sama przy starcie nowego obrazu: nie ma osobnej komendy migracji do wywołania ręcznie.

4. Weryfikacja po aktualizacji

```
docker compose -f /docker/compose/vaultwarden/compose.yaml logs vaultwarden | tail -n 30
```

Sprawdzamy w logu brak błędów przy starcie i w stopce strony logowania w przeglądarce nowy numer wersji.

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: stopka strony logowania Vaultwardena po zalogowaniu do panelu `/admin`, z widocznym numerem wersji zgodnym z nowym tagiem obrazu.

Weryfikacja

```
docker compose -f /docker/compose/vaultwarden/compose.yaml ps
```

Przykład:

```
NAME          IMAGE                          STATUS
vaultwarden   vaultwarden/server:1.38.0      Up 3 minutes
```

```
curl -s -o /dev/null -w "%{http_code}\n" https://hasla.zielona.example/alive
```

Przykład:

```
200
```

Wycofanie zmian

Przed migracją (kopia z kroku 2 aktualizacji jeszcze nie była potrzebna): przerywamy, wracamy do poprzedniego tagu w `.env`, `docker compose up -d` uruchamia starą wersję na niezmienionej jeszcze bazie.

Po migracji, gdy nowa wersja działa niepoprawnie: samo cofnięcie tagu obrazu nie wystarcza, baza ma już nowy schemat, którego stara wersja nie rozumie. Zatrzymujemy kontener, podmieniamy `db.sqlite3` i pozostałe pliki na kopię z kroku 2 aktualizacji, dopiero potem uruchamiamy Vaultwardena ze starym tagiem obrazu wskazanym w `.env`.

```
docker compose -f /docker/compose/vaultwarden/compose.yaml down
cp /docker/apps/vaultwarden/kopie/db-przed-aktualizacja.sqlite3
/docker/apps/vaultwarden/db.sqlite3
```

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
`sqlite3 .backup` kończy się błędem `database is locked`	równoczesny punkt kontrolny WAL akurat w toku	ponawiamy komendę, przy stałych błędach sprawdzamy obciążenie dysku pod `/docker/apps/vaultwarden`
Kopia pliku `db.sqlite3` zrobiona zwykłym `cp` nie odtwarza się poprawnie	plik skopiowany bez uwzględnienia `db.sqlite3-wal`, niespójny stan	powtarzamy kopię komendą `sqlite3 .backup` zamiast `cp`, patrz sekcja „Dlaczego nie kopiujemy pliku na żywo"
Po aktualizacji kontener bez końca się restartuje	migracja przerwana w połowie, na przykład restart hosta w trakcie	odtwarzamy z kopii sprzed aktualizacji, nie próbujemy uruchamiać ponownie tego samego obrazu na uszkodzonej w połowie bazie
Po odtworzeniu kopii logowanie działa, ale wszystkie sesje wylogowane	plik `rsa_key.pem` nie został odtworzony razem z bazą	odtwarzamy `rsa_key.pem` i `rsa_key.pub.pem` z tej samej kopii co `db.sqlite3`, nigdy osobno

Powiązane: Instalacja Vaultwardena w Dockerze · Świat przykładów: Zielona · cron: zadania cykliczne · systemd timery: OnCalendar zamiast crona · Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach · Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje · Vaultwarden: częste problemy

Źródła: github.com/dani-garcia/vaultwarden, wiki „Backing up your vault" · dokumentacja SQLite, rozdział „Online Backup API" i polecenie `.backup` · `sqlite3 --help`

Revision #23
Created 2026-09-28 16:58:05 UTC by Tomasz Konieczny
Updated 2026-10-26 14:18:31 UTC by Tomasz Konieczny