Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach

Typ: procedura **Dotyczy: restic 0.19.1** **Czas: ~50 min**

*Strona Kopia zapasowa aplikacji dockerowych opisuje kopię pojedynczej aplikacji, jedną komendą `tar` i `scp` na raz. Ta strona opisuje kopię **całego** `docker01` jednym mechanizmem: wszystkich katalogów pod `/docker/apps` naraz, z osobnym zrzutem każdego silnika bazy danych uruchomionego w kontenerze, przez restic zamiast ręcznych archiwów.*

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DKIM	DomainKeys Identified Mail: cyfrowy podpis wiadomości e-mail
EPEL	Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów
GFS	Grandfather-Father-Son: schemat retencji kopii, w którym zachowujemy kopie dzienne, tygodniowe i miesięczne (dziadek, ojciec, syn), a starsze wygaszamy
GLPI	Gestionnaire Libre de Parc Informatique: otwarty system do zgłoszeń, inwentarza i bazy wiedzy IT; w Zielonej działa w kontenerze na docker01
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy
SFTP	SSH File Transfer Protocol: bezpieczne przesyłanie plików przez SSH
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Zanim zaczniemy

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.

Z ŻYCIA WZIĘTE

Poniedziałek po weekendzie. Liczba aplikacji na `docker01` przekroczyła dziesięć, a skrypt z osobnym blokiem `tar` i `scp` na każdą z nich, opisany na stronie Kopia zapasowa aplikacji dockerowych, urósł do trzystu linii i nikt już nie pamięta, czy dopisano nową aplikację przy ostatnim wdrożeniu. My decydujemy się na jeden mechanizm kopii dla całego hosta, z listą wyjątków zamiast listy wszystkiego z osobna, i z deduplikacją, która nie każe co noc przesyłać tych samych megabajtów wspólnych warstw.

Komendy wykonujemy z CLI bash z uprawnieniami root (sudo), na `docker01`.

Wymaganie	Uwaga
Dostęp SSH z `docker01` na `bkp01` kluczem, bez hasła	to samo konto co przy kopii pojedynczej aplikacji, patrz Kopia zapasowa aplikacji dockerowych
Miejsce na `bkp01` pod repozytorium restic	repozytorium rośnie wolniej niż suma archiwów `tar`, bo restic przechowuje tylko zmienione bloki
Znajomość silnika bazy każdej aplikacji w kontenerze	tabela w kroku 2, tak jak przy kopii pojedynczej aplikacji

Uważaj: restic kopiujący pliki działającej bazy danych w trakcie zapisu daje ten sam problem niespójności co zwykły `tar`, opisany na stronie Kopia zapasowa aplikacji dockerowych. Katalogi baz danych bind mountowanych pod kontenery wykluczamy z zakresu restic i zamiast nich zrzucamy logiczną kopię narzędziem samej bazy, tak jak dotychczas.

Ta kopia obejmuje tylko `/docker/apps`, `/docker/compose` i zrzuty baz. Dane MailCow (skrzynki, baza, klucze DKIM) leżą w wolumenach Dockera pod `/docker/lib/volumes`, a nie w katalogu `/docker/apps/mailcow`, więc ten skrypt ich nie zabiera. Mamy dwie drogi: dopisać `/docker/lib/volumes` do kopii (wtedy bazę MailCow trzeba traktować jak każdą bazę w kontenerze i unikać kopiowania jej plików w trakcie zapisu) albo, zalecane, kopiować MailCow jego własnym skryptem `backup_and_restore.sh` i dodać katalog `/docker/kopie/mailcow` z jego kopiami do polecenia `restic backup` (według propozycji poziomów retencji do osobnego repozytorium `poczta`, patrz sekcja „Retencja kopii" niżej).

Opis skryptu jest na stronie MailCow: kopia zapasowa i aktualizacja.

Dlaczego restic zamiast tar i scp na cały host

> **Pojęcie: deduplikacja.** Mechanizm, w którym program kopii zapasowej dzieli dane na bloki i zapisuje w repozytorium tylko te bloki, których jeszcze tam nie ma, zamiast całego pliku przy każdym uruchomieniu. Dwie kopie tego samego, prawie niezmienionego katalogu `/docker/apps` różnią się zwykle tylko ułamkiem swojej objętości, więc kolejna kopia po pierwszej pełnej zajmuje realnie dużo mniej miejsca i czasu transferu niż nowy plik `tar.gz` w całości.

Restic i borgbackup rozwiązują ten sam problem: obydwa mają deduplikację, szyfrowanie repozytorium i możliwość trzymania wielu migawek w czasie. Wybieramy restic z jednego konkretnego powodu: repozytorium restic po stronie `bkp01` to zwykły katalog dostępny przez SFTP, czyli zwykłe konto SSH bez żadnego dodatkowego oprogramowania po stronie serwera. Borgbackup w typowym trybie wymaga, żeby binarka `borg` była zainstalowana po obu stronach, na `docker01` i na `bkp01`, i utrzymywana w zgodnej wersji na obu. Restic instalujemy tylko na `docker01`, a `bkp01` zostaje tym samym serwerem plików, którym jest już dla kopii pojedynczych aplikacji.

Przebieg

1. Instalacja restic na docker01

```
curl -LO
https://github.com/restic/restic/releases/download/v0.19.1/restic_0.19.1_linux_amd64.bz2
```

```
bunzip2 restic_0.19.1_linux_amd64.bz2
sudo install -m 755 restic_0.19.1_linux_amd64 /usr/local/bin/restic
```

Wersja `0.19.1` sprawdzona na stronie projektu 28 września 2026. Instalujemy konkretny numer wydania zamiast pakietu z repozytorium dystrybucji, żeby mieć tę samą, świadomie wybraną wersję na `docker01` niezależnie od tego, co akurat trafiło do EPEL.

```
restic version
```

Przykład:

```
restic 0.19.1 compiled with go1.23.4 on linux/amd64
```

2. Rozpoznanie silników baz danych w kontenerach

Ta sama tabela co przy kopii pojedynczej aplikacji, rozszerzona o kolejne silniki spotykane na `docker01`:

Silnik	Przykład aplikacji	Komenda zrzutu
PostgreSQL w kontenerze	NetBox, Gitea	`docker compose exec db pg_dump -U NazwaUzytkownika NazwaBazy`
MariaDB w kontenerze	sklep B2B wewnętrzny, GLPI (osobne zadanie w kroku 8)	`docker compose exec db mariadb-dump -u NazwaUzytkownika -pHASLO NazwaBazy`
MongoDB w kontenerze	katalog produktów	`docker compose exec db mongodump --archive=/tmp/zrzut.archive`
SQLite w pliku (Vaultwarden i inne)	Vaultwarden	na hoście: `sqlite3 /docker/apps/NazwaAplikacji/db.sqlite3 ".backup '/docker/apps/NazwaAplikacji/kopie/zrzut.sqlite3'"` (obraz Vaultwardena nie ma `sqlite3`, osobne zadanie w kroku 7)

`docker compose exec` uruchamia narzędzie zrzutu wewnątrz działającego kontenera bazy, bez zatrzymywania jej i bez instalowania klienta danego silnika na samym `docker01`: każda baza zabiera ze sobą własne narzędzie zrzutu w tym samym obrazie.

3. Katalog na repozytorium restic po stronie bkp01

```
ssh nazwa_usera@bkp01 "mkdir -p /backup/docker01-restic"
```

Restic zakłada strukturę repozytorium sam przy pierwszej inicjalizacji, katalog musi tylko istnieć i być zapisywalny przez konto SSH.

4. Inicjalizacja repozytorium

```
sudo mkdir -p /docker/skrypty
sudo vim /docker/skrypty/restic-haslo.txt
```

```
HasloRepozytoriumResticBardzoDlugieIUnikalne
```

```
sudo chmod 600 /docker/skrypty/restic-haslo.txt
```

> **Pojęcie: hasło repozytorium restic.** Klucz, którym restic szyfruje każdy blok danych przed wysłaniem do repozytorium: bez niego nikt, kto ma dostęp do samego `bkp01`, nie odczyta zawartości kopii, ale też sam zespół IT bez tego hasła nie odtworzy niczego.
>
> Trzymamy je w Vaultwardenie, opisanym na stronie Hasła i sekrety poza repozytorium: .env, Docker secrets i Vaultwarden, nie tylko w pliku na `docker01`.

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/docker01-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo.txt
restic init
```

Przykład:

```
created restic repository a1b2c3d4e5 at sftp:nazwa_usera@bkp01:/backup/docker01-restic
```

5. Skrypt kopii całego hosta

```
sudo vim /docker/skrypty/kopia-docker01-restic.sh
```

```
#!/usr/bin/env bash
set -euo pipefail

# Kopia zapasowa calego docker01: pliki /docker/apps i /docker/compose
# plus logiczne zrzuty baz danych uruchomionych w kontenerach.

export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/docker01-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo.txt
ZRZUTY=/docker/skrypty/zrzuty-tymczasowe
LOG=/docker/skrypty/kopia-docker01-restic.log

mkdir -p "${ZRZUTY}"
echo "=== Start kopii: $(date -Iseconds) ===" >> "${LOG}"

# Zrzuty logiczne baz danych, kazdy do osobnego pliku w katalogu tymczasowym.
docker compose -f /docker/compose/netbox/compose.yaml exec -T db \
```

```
  pg_dump -U netbox netbox > "${ZRZUTY}/netbox.sql"
docker compose -f /docker/compose/gitea/compose.yaml exec -T db \
  pg_dump -U gitea gitea > "${ZRZUTY}/gitea.sql"

echo "Zrzuty baz danych: OK, kod $?" >> "${LOG}"

# Kopia calego /docker/apps i /docker/compose, plus katalog ze swiezymi zrzutami,
# z wylaczeniem katalogow zrodlowych baz danych, ktore zrzucamy logicznie wyzej.
# Vaultwarden (krok 7) i GLPI (krok 8) maja osobne repozytoria, wiec tu je pomijamy.
restic backup /docker/apps /docker/compose "${ZRZUTY}" \
  --exclude=/docker/apps/netbox/postgres \
  --exclude=/docker/apps/gitea/postgres \
  --exclude=/docker/apps/vaultwarden \
  --exclude=/docker/apps/glpi \
  --tag docker01 >> "${LOG}" 2>&1

echo "Backup restic: kod wyjscia $?" >> "${LOG}"

# Retencja, poziom 90 dni: dzienne z 7 dni i jedna kopia tygodniowo przez 90 dni.
restic forget --keep-within 7d --keep-within-weekly 90d --prune >> "${LOG}" 2>&1

echo "Retencja: kod wyjscia $?" >> "${LOG}"

rm -rf "${ZRZUTY:?}"/*
echo "=== Koniec kopii: $(date -Iseconds) ===" >> "${LOG}"
```

```
sudo chmod +x /docker/skrypty/kopia-docker01-restic.sh
```

Lista aplikacji ze zrzutem bazy w skrypcie rośnie ręcznie przy każdym wdrożeniu nowej aplikacji z bazą w kontenerze, dokładnie tak jak tabela z kroku 2: `restic backup` sam znajdzie nowy katalog pod `/docker/apps`, ale logicznego zrzutu jego bazy nikt nie doda za nas.

`set -euo pipefail` zatrzymuje skrypt na pierwszym błędzie, zamiast wysłać kopię bez jednej z baz i zostawić o tym tylko cichy wpis w logu. Każdy `>> "${LOG}" 2>&1` po komendzie restic dopisuje pełne wyjście do jednego pliku logu, a jawne `echo` z `$?` po każdym etapie potwierdza w tym samym pliku, że dany krok faktycznie zakończył się kodem zero, zanim skrypt przejdzie dalej.

6. Timer systemd

Wybieramy systemd timery: OnCalendar zamiast crona, tak jak przy kopii pojedynczej aplikacji.

```
sudo vim /etc/systemd/system/kopia-docker01-restic.service
```

```
[Unit]
Description=Kopia zapasowa calego docker01 przez restic

[Service]
Type=oneshot
ExecStart=/docker/skrypty/kopia-docker01-restic.sh
```

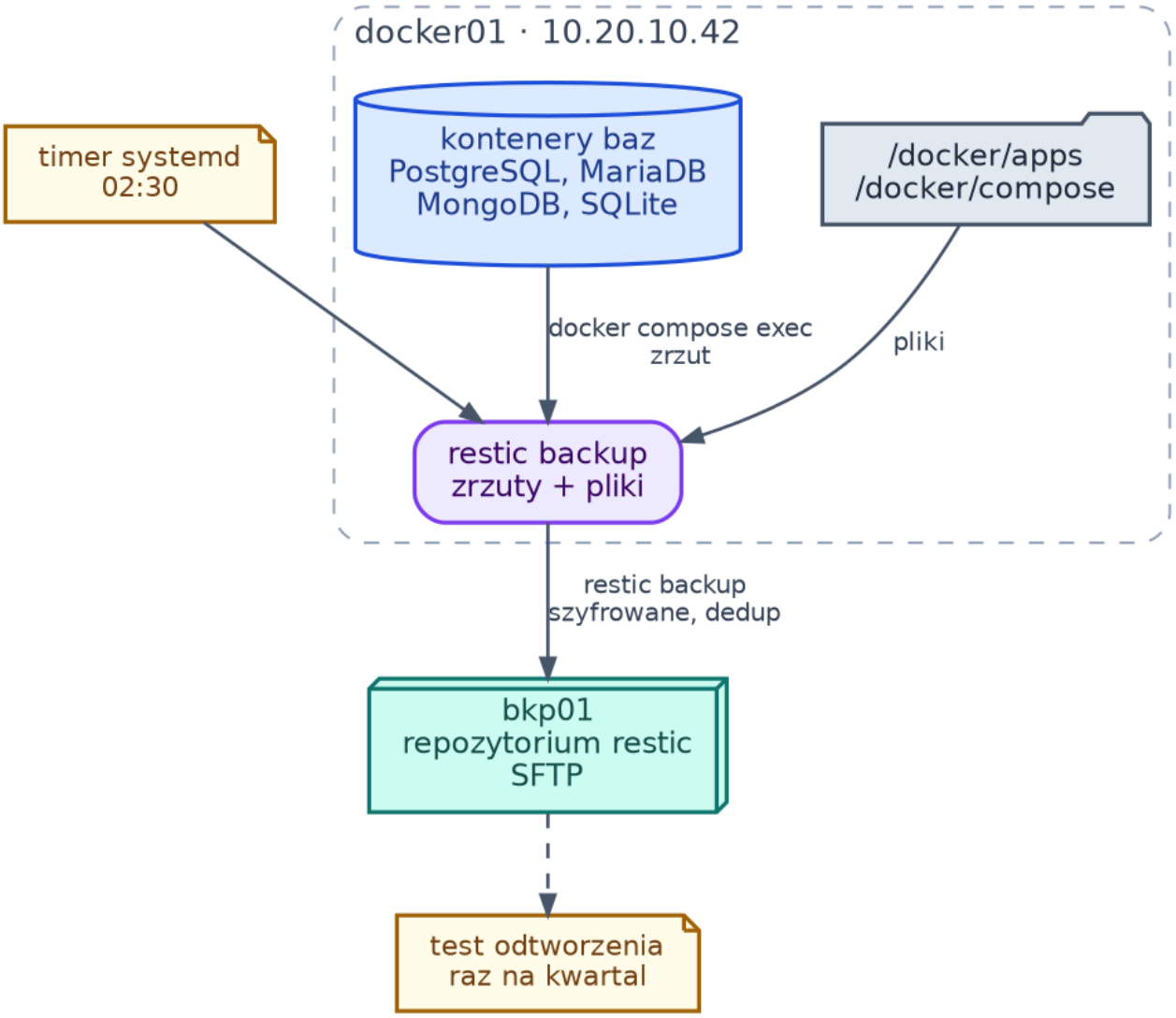
```
sudo vim /etc/systemd/system/kopia-docker01-restic.timer
```

```
[Unit]
Description=Harmonogram kopii zapasowej docker01 przez restic

[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
sudo systemctl enable --now kopia-docker01-restic.timer
```



Restic na docker01 zrzuca bazy z kontenerow i pliki z /docker, wysyla do repozytorium na bkp01, kopia testowana raz na kwartal (kliknięcie otwiera pełny rozmiar)

> Czas uruchomienia `02:30` pokrywa się z czasem kopii pojedynczych aplikacji ze strony Kopia zapasowa aplikacji dockerowych. Docelowo ta strona zastępuje tamten mechanizm w całości, więc nie uruchamiamy obu naraz na tym samym hoście: albo osobne skrypty per aplikacja, albo jeden mechanizm restic dla całego `docker01`, nigdy oba jednocześnie na te same dane.

7. Osobne zadanie: kopia Vaultwardena do repozytorium vaultwarden

Vaultwarden trzyma hasła całej firmy, więc według decyzji autora z 11 października 2026 ma własne repozytorium `vaultwarden` z poziomem 365 dni, a nie miejsce w repozytorium `docker01` z poziomem 90 dni (tabela przypisania usług na stronie Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje). Poziom dotyczy całego repozytorium, dlatego kopia jest osobnym zadaniem z osobnym hasłem, a skrypt z kroku 5 wyklucza `/docker/apps/vaultwarden` w całości.

Dlaczego wykluczamy, a nie kopiujemy w obu repozytoriach. Repozytorium `vaultwarden` ma te same dzienne i tygodniowe kopie z ostatnich 90 dni co `docker01` i do tego paczki miesięczne i kwartalne, więc druga kopia w `docker01` nie daje żadnego punktu odtworzenia więcej. Dokłada za to drugie miejsce z zawartością sejfu i kluczami `rsa_key*`, chronione innym hasłem, i dwie wersje tej samej bazy z różnych godzin, z których przy odtworzeniu trzeba by wybierać. Ceną jest to, że pełne odtworzenie `docker01` wymaga dwóch repozytoriów: `docker01` dla reszty aplikacji i `vaultwarden` dla sejfu.

Spójną kopię bazy robimy na hoście poleceniem `.backup` narzędzia `sqlite3`, tak jak na stronie Kopia zapasowa i aktualizacja Vaultwardena. Dokumentacja Vaultwardena zaleca `.backup` dla bazy w użyciu i podaje, że obrazy Dockera nie zawierają `sqlite3`, więc polecenie uruchamiamy na `docker01` (pakiet `sqlite`). Od wersji 1.32.1 jest też wbudowane polecenie `docker exec -it vaultwarden /vaultwarden backup`; miejsce i nazwę pliku, który tworzy, sprawdzamy przed użyciem (do potwierdzenia). Z repozytorium wykluczamy plik bazy z plikami `-wal` i `-shm`, bo kopiowane w trakcie zapisu dają niespójny stan, oraz `icon_cache`, który według dokumentacji jest opcjonalny. Zostają `attachments`, `sends`, `rsa_key*` i `config.json`.

```
ssh nazwa_usera@bkp01 "mkdir -p /backup/vaultwarden-restic"
sudo vim /docker/skrypty/restic-haslo-vaultwarden.txt
sudo chmod 600 /docker/skrypty/restic-haslo-vaultwarden.txt
```

Hasło repozytorium `vaultwarden` jest inne niż hasło `docker01`, a jego wydruk trafia do koperty w sejfie, bo sam Vaultwarden nie może być jedynym miejscem hasła do własnej kopii.

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/vaultwarden-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo-vaultwarden.txt
restic init
```

```
sudo vim /docker/skrypty/kopia-vaultwarden-restic.sh
```

```
#!/usr/bin/env bash
set -euo pipefail

# Kopia Vaultwardena do osobnego repozytorium restic vaultwarden (poziom 365 dni).
# Najpierw spojna kopia bazy SQLite przez sqlite3 .backup na hoscie, potem restic.
```

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/vaultwarden-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo-vaultwarden.txt
DANE=/docker/apps/vaultwarden
ZRZUT="${DANE}/kopie/db-restic.sqlite3"
LOG=/docker/skrypty/kopia-vaultwarden-restic.log

mkdir -p "${DANE}/kopie"
echo "=== Start kopii Vaultwardena: $(date -Iseconds) ===" >> "${LOG}"

# .backup korzysta z blokad SQLite, wiec kontener nie musi stac.
rm -f "${ZRZUT}"
sqlite3 "${DANE}/db.sqlite3" ".backup '${ZRZUT}'"
if [ "$(sqlite3 "${ZRZUT}" 'PRAGMA integrity_check;')" != "ok" ]; then
  echo "BLAD: zrzut bazy nie przeszedl integrity_check" >> "${LOG}"
  exit 1
fi
echo "Zrzut bazy: OK" >> "${LOG}"

restic backup "${DANE}" \
  --exclude="${DANE}/db.sqlite3" \
  --exclude="${DANE}/db.sqlite3-wal" \
  --exclude="${DANE}/db.sqlite3-shm" \
  --exclude="${DANE}/icon_cache" \
  --tag vaultwarden >> "${LOG}" 2>&1
echo "Backup restic: OK" >> "${LOG}"

# Na bkp01 poziom 365 dni to dzienne z 7 dni i tygodniowe przez 90 dni.
# Paczki miesieczne i kwartalne robi archiwum na bkp01, nie forget.
restic forget --keep-within 7d --keep-within-weekly 90d --prune >> "${LOG}" 2>&1
echo "Retencja: OK" >> "${LOG}"
echo "=== Koniec kopii Vaultwardena: $(date -Iseconds) ===" >> "${LOG}"
```

```
sudo chmod +x /docker/skrypty/kopia-vaultwarden-restic.sh
```

Zrzut `kopie/db-restic.sqlite3` ma stałą nazwę i co noc jest nadpisywany: historię trzymają migawki restic, a nie kolejne pliki na dysku. `PRAGMA integrity_check` zwraca `ok`, gdy baza w zrzucie jest spójna, i dopiero wtedy kopia idzie dalej. Przy odtwarzaniu zrzut kopiujemy na miejsce `db.sqlite3`, a stary plik `db.sqlite3-wal` usuwamy wcześniej, bo dokumentacja Vaultwardena ostrzega, że pozostawiony obok kopii z `.backup` może uszkodzić bazę.

Usługa i timer mają tę samą postać co w kroku 6, z inną nazwą, skryptem i godziną:

```
[Unit]
Description=Kopia zapasowa Vaultwardena do repozytorium restic vaultwarden

[Service]
Type=oneshot
ExecStart=/docker/skrypty/kopia-vaultwarden-restic.sh
```

```
[Unit]
Description=Harmonogram kopii Vaultwardena przez restic

[Timer]
OnCalendar=*-*-* 02:45:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
sudo systemctl enable --now kopia-vaultwarden-restic.timer
```

Pliki zapisujemy jako `/etc/systemd/system/kopia-vaultwarden-restic.service` i `/etc/systemd/system/kopia-vaultwarden-restic.timer`. Godzina 02:45 nie nakłada się na kopię `docker01` o 02:30 ani na synchronizację z `docker02` w minutach `07`, `22`, `37` i `52` ze strony Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie, a przy ciepłej rezerwie dopisujemy do skryptu ten sam warunek „tylko na aktywnym hoście” co w skrypcie z kroku 5.

> Pełny poziom 365 dni działa dopiero po przejściu na rest-server ze strony Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność. Repozytorium leży wtedy w `/backup/rest/vaultwarden`, `forget` znika ze skryptu i wykonuje go `bkp01`, a paczki miesięczne i kwartalne robi skrypt `archiwum-miesieczne.sh` według wpisu `[vaultwarden]=365` w `poziomy.sh`. Do tego czasu na `bkp01` leżą tylko kopie dzienne z 7 dni i tygodniowe z 90 dni.

8. Osobne zadanie: kopia GLPI do repozytorium glpi

System zgłoszeń GLPI ze strony GLPI w Zielonej: zgłoszenia, SLA i katalog usług na docker01 ma według decyzji autora z 11 października 2026 własne repozytorium `glpi` z poziomem 5 lat, bo historia zgłoszeń i zmian jest dowodem dla audytora. Tak jak przy Vaultwardenie skrypt z kroku 5 wyklucza `/docker/apps/glpi` w całości, a GLPI ma osobne zadanie z osobnym hasłem repozytorium, z tych samych powodów: jedno źródło kopii i jedno hasło do danych, które trzymamy dłużej niż resztę.

Według Compose z tamtej strony baza działa w usłudze `db` (obraz `mariadb:11.4.13`, pliki w `/docker/apps/glpi/mariadb`), a GLPI w usłudze `glpi` z wolumenem `/docker/apps/glpi/glpi` zamontowanym jako `/var/glpi`. Opis obrazu `glpi/glpi` podaje w `/var/glpi` katalogi `config`, `files`, `logs` i `marketplace`, więc wtyczki z Marketplace są w tym wolumenie. Wtyczki wgrane ręcznie leżą w `/var/www/glpi/plugins`, poza wolumenem, dlatego Compose montuje ten katalog z `/docker/apps/glpi/plugins`. Do kopii wchodzą więc zrzut bazy, cały `/docker/apps/glpi/glpi` (z kluczem szyfrowania w `config`), `/docker/apps/glpi/plugins` i `/docker/compose/glpi`, a plików bazy z `/docker/apps/glpi/mariadb` nie kopiujemy, bo kopiowane w trakcie zapisu dają niespójny stan. `/docker/compose/glpi` zostaje także w repozytorium `docker01`, razem z resztą `/docker/compose`.

```
ssh nazwa_usera@bkp01 "mkdir -p /backup/glpi-restic"
sudo vim /docker/skrypty/restic-haslo-glpi.txt
sudo chmod 600 /docker/skrypty/restic-haslo-glpi.txt
sudo mkdir -p /docker/kopie/glpi
```

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/glpi-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo-glpi.txt
restic init
```

```
sudo vim /docker/skrypty/kopia-glpi-restic.sh
```

```
#!/usr/bin/env bash
set -euo pipefail
umask 077

# Kopia GLPI do osobnego repozytorium restic glpi (poziom 5 lat).
# Najpierw spojny zrzut MariaDB z kontenera db, potem restic.

export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/glpi-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo-glpi.txt
COMPOSE=/docker/compose/glpi/compose.yaml
ZRZUT=/docker/kopie/glpi/glpi.sql
LOG=/docker/skrypty/kopia-glpi-restic.log
```

```
mkdir -p /docker/kopie/glpi
echo "=== Start kopii GLPI: $(date -Iseconds) ===" >> "${LOG}"

# --single-transaction daje spojny stan tabel InnoDB bez zatrzymywania GLPI.
# Haslo czytamy ze zmiennych wewnatrz kontenera bazy, nie z wiersza polecen hosta.
# shellcheck disable=SC2016
docker compose -f "${COMPOSE}" exec -T db \
  sh -c 'mariadb-dump --single-transaction -u "$MARIADB_USER" -p"$MARIADB_PASSWORD"
"$MARIADB_DATABASE"' \
  > "${ZRZUT}.tmp"
if ! tail -n 1 "${ZRZUT}.tmp" | grep -q '^-- Dump completed'; then
  echo "BLAD: zrzut bazy GLPI niepelny" >> "${LOG}"
  exit 1
fi
mv "${ZRZUT}.tmp" "${ZRZUT}"
echo "Zrzut bazy: OK" >> "${LOG}"

restic backup "${ZRZUT}" /docker/apps/glpi/glpi /docker/apps/glpi/plugins /docker/compose/glpi
\
  --tag glpi >> "${LOG}" 2>&1
echo "Backup restic: OK" >> "${LOG}"

# Na bkp01 poziom 5 lat to lokalnie dzienne z 7 dni i tygodniowe przez 90 dni.
# Paczki miesieczne, kwartalne i roczne robi archiwum na bkp01, nie forget.
restic forget --keep-within 7d --keep-within-weekly 90d --prune >> "${LOG}" 2>&1
echo "Retencja: OK" >> "${LOG}"
echo "=== Koniec kopii GLPI: $(date -Iseconds) ===" >> "${LOG}"
```

```
sudo chmod +x /docker/skrypty/kopia-glpi-restic.sh
```

Zrzut powstaje najpierw jako plik `.tmp` i dopiero po sprawdzeniu zastępuje `glpi.sql`, więc przerwany zrzut nie nadpisze ostatniego dobrego. `mariadb-dump` z domyślnymi komentarzami kończy plik linią `-- Dump completed`, a jej brak oznacza, że zrzut urwał się w połowie (zachowanie do potwierdzenia na naszej wersji MariaDB). `umask 077` sprawia, że plik zrzutu z danymi zgłoszeń czyta tylko root. Zrzut ma stałą nazwę, a historię trzymają migawki restic.

Usługa i timer mają tę samą postać co w kroku 6:

```
[Unit]
Description=Kopia zapasowa GLPI do repozytorium restic glpi
```

```
[Service]
Type=oneshot
ExecStart=/docker/skrypty/kopia-glpi-restic.sh
```

```
[Unit]
Description=Harmonogram kopii GLPI przez restic

[Timer]
OnCalendar=*-*-* 03:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
sudo systemctl enable --now kopia-glpi-restic.timer
```

Pliki zapisujemy jako `/etc/systemd/system/kopia-glpi-restic.service` i `/etc/systemd/system/kopia-glpi-restic.timer`. Godzina 03:00 nie nakłada się na kopie `docker01` (02:30) i Vaultwardena (02:45) ani na minuty synchronizacji z `docker02`, a przy ciepłej rezerwie dopisujemy ten sam warunek „tylko na aktywnym hoście” co w skrypcie z kroku 5. Odtworzenie: zrzut wczytujemy do pustej bazy w usłudze `db` narzędziem `mariadb`, a katalogi `/docker/apps/glpi/glpi` i `/docker/apps/glpi/plugins` odtwarzamy razem z nim, bo bez klucza z `config` hasła zapisane w bazie trzeba wpisać od nowa, a bez katalogu `plugins` przestaną działać wtyczki wgrane ręcznie.

> Poziom 5 lat działa w pełni dopiero po przejściu na rest-server ze strony Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność: repozytorium leży wtedy w `/backup/rest/glpi`, `forget` wykonuje `bkp01`, a skrypt `archiwum-miesieczne.sh` według wpisu `[glpi]=1825` w `poziomy.sh` robi paczki miesięczne, kwartalne i roczną (w marcu) z niezmiennością. Do tego czasu na `bkp01` leżą tylko kopie dzienne z 7 dni i tygodniowe z 90 dni.

Weryfikacja

```
journalctl -u kopia-docker01-restic.service --since today
```

```
restic snapshots
```

Przykład:

```
ID        Time                 Host      Tags       Paths
a1b2c3d4  2026-09-28 02:30:11  docker01  docker01   /docker/apps, /docker/compose,
/docker/skrypty/zrzuty-tymczasowe
```

```
restic stats latest
```

Przykład:

```
Total File Count:   48213
Total Size:          62.104 GiB
```

Test odtworzenia jednej aplikacji na maszynie testowej

Raz na kwartał, tak jak przy kopii pojedynczej aplikacji, odtwarzamy jedną aplikację z najświeższej migawki na osobnym, testowym hoście, niezależnym od `docker01`.

```
restic restore latest --target /tmp/test-odtworzenia --include /docker/apps/netbox
```

Kolejność jest zawsze taka sama: odtworzenie plików z migawki, odtworzenie zrzutu bazy do pustej bazy narzędziem `psql` albo `pg_restore`, skopiowanie `compose.yaml` i `.env` z odtworzonego `/docker/compose/netbox`, `docker compose up -d`, sprawdzenie w przeglądarce, że aplikacja wygląda tak jak na `docker01`. Migawka, której nikt nigdy nie spróbował odtworzyć na osobnym hoście, jest tylko przypuszczeniem, że działa, dokładnie tak jak przy każdym innym mechanizmie kopii w tej książce.

Retencja kopii

Skrypt z kroku 5 zostawia kopie dzienne z ostatnich 7 dni i jedną kopię tygodniowo przez 90 dni (poziom 90 dni z tabeli poziomów, przypisany repozytorium `docker01`), a `restic forget --prune` wywołuje po stronie `docker01`. To najprostsza droga i zostaje na tej stronie. Poniżej dwie rzeczy: retencja liczona w dniach, uruchamiana osobno z harmonogramem, oraz wariant „czyste SFTP bez restica" dla kopii, które są zwykłymi plikami.

> Retencję najpierw ustalamy jako politykę (ile dni i w jakiej gęstości trzymamy), a dopiero potem wpisujemy w narzędzie. Poziomy retencji (7, 30, 90, 180, 365 dni, 5 i 10 lat) i

> przypisanie do nich usług są w sekcjach „Retencja GFS" i „Polityka retencji" na stronie Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje. Gdy firma potrzebuje niezmienności, raportów albo kopii całej maszyny, patrzymy na stronę Kopie hostów Dockera narzędziami do kopii: Veeam i odpowiedniki.

Retencja restic: 90 dni albo poziomy z harmonogramu

Wariant z `--keep-within` zostawia wszystkie migawki z ostatnich 90 dni, a resztę usuwa. Polecenie najpierw uruchamiamy z `--dry-run`, który pokazuje, co zostałoby usunięte, i niczego nie zmienia (w sesji, w której wyeksportowaliśmy `RESTIC_REPOSITORY` i `RESTIC_PASSWORD_FILE` jak w kroku 4):

```
restic forget --keep-within 90d --dry-run
```

Po obejrzeniu wyniku wykonujemy tę samą komendę z `--prune` zamiast `--dry-run`. Dokumentacja restic opisuje `--keep-within` jako okres liczony od najnowszej migawki, nie od dziś, więc przy zatrzymanej kopii nic nie wygasa. Wariant „wszystkie kopie przez 90 dni" zostaje jako prosty przykład, a poziom 90 dni z tabeli poziomów (dzienne z 7 dni i tygodniowe przez 90 dni) ma postać:

```
restic forget --keep-within 7d --keep-within-weekly 90d --prune
```

Dłuższe poziomy (180 dni i więcej) restic nie realizuje przez `forget`, tylko przez osobne repozytoria archiwalne ze strony Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność, a dla Dockera proponujemy poziom 90 dni. MailCow ma w propozycji poziom 365 dni, więc jego kopie z `/docker/kopie/mailcow` idą do osobnego repozytorium `poczta`, a nie do `docker01`. Tak samo Vaultwarden: poziom 365 dni i osobne repozytorium `vaultwarden` z kroku 7, oraz GLPI: poziom 5 lat i osobne repozytorium `glpi` z kroku 8. `prune` jest ciężki, więc wykonujemy go rzadziej niż kopię, na przykład raz w tygodniu. Gdy wybieramy osobny harmonogram, linię z `forget` w skrypcie z kroku 5 usuwamy, żeby nie działały dwie polityki naraz, a retencję wykonuje osobny skrypt:

```
sudo vim /docker/skrypty/retencja-docker01-restic.sh
```

```
#!/usr/bin/env bash
set -euo pipefail

# Retencja repozytorium restic docker01: wszystkie migawki z ostatnich 90 dni.
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/docker01-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo.txt
LOG=/docker/skrypty/retencja-docker01-restic.log
```

```
echo "=== Start retencji: $(date -Iseconds) ===" >> "${LOG}"
if restic forget --keep-within 90d --prune >> "${LOG}" 2>&1; then
  echo "Retencja: OK" >> "${LOG}"
else
  echo "Retencja: BLAD" >> "${LOG}"
  exit 1
fi
echo "=== Koniec retencji: $(date -Iseconds) ===" >> "${LOG}"
```

```
sudo chmod +x /docker/skrypty/retencja-docker01-restic.sh
```

Harmonogram jest drugim timerem systemd, w niedzielę rano, po kopii z nocy:

```
sudo vim /etc/systemd/system/retencja-docker01-restic.service
```

```
[Unit]
Description=Retencja repozytorium restic docker01

[Service]
Type=oneshot
ExecStart=/docker/skrypty/retencja-docker01-restic.sh
```

```
sudo vim /etc/systemd/system/retencja-docker01-restic.timer
```

```
[Unit]
Description=Harmonogram retencji repozytorium restic docker01

[Timer]
OnCalendar=Sun *-*-* 06:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
sudo systemctl enable --now retencja-docker01-restic.timer
```

Wariant: czyste SFTP bez restica

Kopie robione zwykłym `tar` i `scp`, jak na stronie Kopia zapasowa aplikacji dockerowych, są pojedynczymi plikami w katalogu na `bkp01`, na przykład `/backup/netbox`. Nie ma tu migawek ani `forget`, więc stare pliki usuwa skrypt na `bkp01`, uruchamiany tam z harmonogramu. Najpierw podgląd, co by zniknęło (komendy wykonujemy na `bkp01`):

```
find /backup/NazwaAplikacji -type f -mtime +90 -print
```

Zapis `-mtime +90` oznacza pliki zmienione ponad 90 pełnych dób temu, a `-type f` pomija katalogi. Lista musi zawierać tylko stare kopie tej aplikacji: dopiero potem wykonujemy usuwanie.

```
find /backup/NazwaAplikacji -type f -mtime +90 -delete
```

Usunięcie jest nieodwracalne, a literówka w ścieżce (na przykład `/backup` zamiast `/backup/NazwaAplikacji`) usuwa kopie wszystkich aplikacji starsze niż 90 dni. Dlatego ścieżkę zawsze wpisujemy po podglądzie z `-print`, a w skrypcie harmonogramu nie używamy zmiennych, które mogą być puste.

> Konto, którym `docker01` zapisuje kopie przez SFTP (zwykłe SFTP albo restic przez SFTP), może też kasować pliki w katalogu kopii. Napastnik, który przejmie `docker01` i jego klucz SSH, może skasować kopie razem z danymi, i nie pomoże mu w tym żadna retencja. Dlatego w książce Kopie zapasowe zalecamy repozytorium tylko do dopisywania (rest-server z `--append-only`, retencja po stronie serwera kopii) albo narzędzie z niezmiennością: strony Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność i Kopie hostów Dockera narzędziami do kopii: Veeam i odpowiedniki.

Wycofanie zmian

```
sudo systemctl disable --now kopia-docker01-restic.timer
sudo systemctl disable --now kopia-vaultwarden-restic.timer
sudo systemctl disable --now kopia-glpi-restic.timer
```

Istniejące migawki w repozytorium na `bkp01` zostają nietknięte, wyłączenie timera zatrzymuje tylko tworzenie nowych.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy

Skrypt kończy się natychmiast bez nowej migawki	`set -euo pipefail` zatrzymał go na pierwszym błędzie, zwykle w jednym ze zrzutów baz	`journalctl -u kopia-docker01-restic.service`, sprawdzamy plik `LOG`, w którym kroku padło
`restic backup` zwraca `unable to open repository`	zmienna `RESTIC_REPOSITORY` albo `RESTIC_PASSWORD_FILE` nie ustawiona w bieżącej sesji skryptu	sprawdzamy, czy `export` obu zmiennych stoi na początku skryptu, nie tylko w sesji interaktywnej
`restic forget --prune` trwa bardzo długo	pierwsze uruchomienie prune po długim czasie bez retencji, dużo bloków do przeliczenia	normalne przy pierwszym uruchomieniu, kolejne przebiegi są szybsze
Repozytorium rośnie mimo włączonej retencji	`restic forget` bez `--prune` tylko oznacza migawki jako usunięte, nie zwalnia miejsca	dopisujemy `--prune` do tego samego wywołania, tak jak w skrypcie wyżej
Test odtworzenia na czystym hoście nie startuje kontenera	inna wersja Dockera na hoście testowym albo brakujący plik `.env`, którego restic też kopiuje razem z resztą `/docker/compose`	sprawdzamy `docker version` na obu hostach, `.env` powinien odtworzyć się razem z `compose.yaml`

□□ Ćwiczenia

1. Dopisujemy do skryptu kopii zrzut bazy MariaDB dla aplikacji `sklep-b2b`, według wzoru z tabeli w kroku 2.
2. Wyjaśniamy, dlaczego katalog bazy danych bind mountowany pod kontener wykluczamy z `restic backup` przez `--exclude`, mimo że sam restic potrafi kopiować pliki bez zatrzymywania aplikacji.

Odpowiedź

Bo wykluczenie dotyczy nie samej zdolności restic do skopiowania plików w trakcie ich zmiany, tylko spójności tego, co skopiuje: plik bazy danych zmieniający się w trakcie kopiowania może zostać zapisany w repozytorium w stanie w połowie zapisu, nieczytelnym dla silnika bazy przy odtworzeniu. Logiczny zrzut narzędziem samej bazy (`pg_dump`, `mariadb-dump`, `mongodump`) daje zawsze spójny stan na moment zrzutu, niezależnie od tego, że baza cały czas działa.

1. Repozytorium restic na `bkp01` rośnie z dnia na dzień mimo działającej retencji `--keep-within 7d --keep-within-weekly 90d`. Wskazujemy, jakiego elementu w skrypcie prawdopodobnie brakuje.

Odpowiedź

Prawdopodobnie brakuje flagi `--prune` przy wywołaniu `restic forget`. Bez niej restic tylko oznacza stare migawki jako usunięte z listy, ale nie zwalnia zajmowanych przez nie bloków w

repozytorium, więc miejsce na `bkp01` rośnie dalej, mimo że liczba widocznych migawek faktycznie się zmniejsza.

Powiązane: Kopia zapasowa aplikacji dockerowych · Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · systemd timery: OnCalendar zamiast crona · Hasła i sekrety poza repozytorium: .env, Docker secrets i Vaultwarden · Kopia zapasowa i aktualizacja Vaultwardena · GLPI w Zielonej: zgłoszenia, SLA i katalog usług na docker01

Źródła: restic.readthedocs.io, rozdział „Installation" · restic.readthedocs.io, rozdział „Backing up" · restic.net, wydanie 0.19.1 · github.com/dani-garcia/vaultwarden, wiki „Backing up your vault" (`.backup` dla bazy w użyciu, brak `sqlite3` w obrazach Dockera, polecenie `/vaultwarden backup` od 1.32.1, pliki do kopii, usunięcie `db.sqlite3-wal` przed odtworzeniem; stan na 11.10.2026) · hub.docker.com/r/glpi/glpi (katalogi `/var/glpi/config`, `/var/glpi/files`, `/var/glpi/logs`, `/var/glpi/marketplace`, usługa bazy `db`; stan na 11.10.2026)

Revision #23
Created 2026-09-28 17:15:48 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:35 UTC by Tomasz Konieczny