Kontenery i dane

Przenoszenie kontenerów między hostami i replikacja danych.

- Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie
- Replikacja danych między hostami: DRBD, GlusterFS, Ceph, ZFS i replikacja w aplikacji

Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie

Poziom: średnio

Dotyczy: Docker Engine 29 z Compose, restic 0.19.1, keepalived 2.2 (pakiet EL9), rsync, Rocky Linux 9

Czas: ~90 min

Kontenera Dockera nie przenosi się na żywo, przenosimy jego przepis (Compose, konfigurację, sekrety) i dane, a kontener powstaje od nowa na drugim hoście. Strona pokazuje dwa poziomy: zimną rezerwę z kopii restic na `bkp01` i ciepłą rezerwę, w której `docker02` ma dane sprzed kilkunastu minut i przejmuje adres wirtualny. Wracamy tu przy budowie `docker02`, przy teście przełączenia i wtedy, gdy `docker01` przestaje działać.

ZAŁOŻENIA SCENARIUSZA

`docker01` i `docker02` są maszynami wirtualnymi na różnych hostach klastra `hv01` i `hv02`, w jednej sieci VLAN 10, bo VRRP wymaga wspólnego segmentu. Adres wirtualny to 10.20.10.46, a przekierowania portów na pojedynczym `fw01` wskazują właśnie jego.

Skróty użyte na tej stronie (23)

Skrót	Co znaczy
ACME	Automatic Certificate Management Environment: protokół, którym Traefik albo certbot automatycznie zamawiają i odnawiają certyfikat w Let's Encrypt, bez ręcznego wniosku
ARP	Address Resolution Protocol: protokół, który zamienia adres IP w sieci lokalnej na adres sprzętowy karty sieciowej (MAC)
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu

Skrót	Co znaczy
CRIU	Checkpoint/Restore In Userspace: narzędzie do zapisu i wznawiania stanu procesu, na którym opiera się eksperymentalny docker checkpoint
DKMS	Dynamic Kernel Module Support: mechanizm, który sam przebudowuje moduł jądra, na przykład sterownik karty, po aktualizacji jądra
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
EL9	Enterprise Linux 9: wspólna nazwa dystrybucji zgodnych z RHEL 9, takich jak Rocky Linux 9
GiB	gibibajt: 1024 MiB, czyli około 1,07 GB; tej jednostki używa dokumentacja MailCow przy wymaganiach pamięci
GLPI	Gestionnaire Libre de Parc Informatique: otwarty system do zgłoszeń, inwentarza i bazy wiedzy IT; w Zielonej działa w kontenerze na docker01
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
LVM	Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny
MAC	Media Access Control: unikalny adres fizyczny karty sieciowej, przypisany fabrycznie
RFC	Request for Comments: oficjalny dokument opisujący standard internetowy, na przykład sposób działania danego protokołu; w ITIL także Request for Change, czyli wniosek o zmianę
RPO	Recovery Point Objective: ile danych wolno stracić przy awarii
RTO	Recovery Time Objective: jak długo wolno nie działać przy awarii
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TTL	Time To Live: czas życia rekordu lub danych
VLAN	Virtual LAN: wydzielona logicznie sieć na tym samym sprzęcie
VRRP	Virtual Router Redundancy Protocol: protokół, w którym kilka hostów uzgadnia, który z nich trzyma wspólny adres IP (opisany w RFC 5798)
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach
ZFS	system plików z migawkami i replikacją przyrostową zbudowany w jednym narzędziu z menedżerem wolumenów (dawniej Zettabyte File System)

Fiszka

Komendy wykonujemy z CLI bash z uprawnieniami root (sudo), na `docker01` albo `docker02`, chyba że przy komendzie napisano inaczej. Adresy i nazwy z Zielonej: `docker01` to `10.20.10.42`, `docker02` to `10.20.10.47` (propozycja, wolny w całej bazie), adres wirtualny to `10.20.10.46`, interfejs sieciowy

to `ens192`.

TYLKO ODCZYT

Sprawdzenie, który host trzyma adres wirtualny: ip -br -4 addr show

```
ip -br -4 addr show dev NazwaInterfejsu
```

Aktywny host ma na interfejsie dwa adresy: własny i wirtualny. Host rezerwowy ma tylko własny.

Przykład na aktywnym `docker01`:

```
ens192           UP             10.20.10.42/24 10.20.10.46/24
```

ip = narzędzie do adresów, tras i interfejsów sieciowych

TYLKO ODCZYT

Sprawdzenie stanu keepalived: journalctl -u keepalived

```
sudo journalctl -u keepalived --since "1 hour ago"
```

Szukamy wpisów o zmianie stanu, na przykład `Entering MASTER STATE` albo `Entering BACKUP STATE`, z nazwą instancji `VI_DOCKER`.

journalctl = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Sprawdzenie pliku konfiguracyjnego keepalived: keepalived --config-test

```
sudo keepalived --config-test
```

Sprawdza składnię `/etc/keepalived/keepalived.conf` bez uruchamiania usługi. Robimy to przed każdym `systemctl restart keepalived`, bo błąd w pliku na aktywnym hoście kończy się utratą adresu wirtualnego.

keepalived = demon, który wykonuje VRRP i przenosi adres wirtualny między hostami, a skryptami kontroluje stan usługi

ZMIENIA SYSTEM

Wymuszenie synchronizacji na partnera: systemctl start ft-ha-synchronizuj

```
sudo systemctl start ft-ha-synchronizuj.service
sudo tail -n 20 /var/log/ft-ha.log
```

Uruchamia od ręki to, co timer robi co kwadrans. Na hoście bez adresu wirtualnego skrypt kończy się po cichu, bo kopiuje tylko z aktywnego.

systemctl = system control: sterowanie usługami systemd · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

ZMIENIA SYSTEM

Odtworzenie aplikacji z kopii restic na docker02: restic restore

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/docker01-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo.txt
sudo -E restic restore latest --host docker01 --tag docker01 --target / \
  --include /docker/apps/NazwaAplikacji \
  --include /docker/compose/NazwaAplikacji \
  --include /docker/skrypty/zrzuty-tymczasowe
```

> (gdzie `NazwaAplikacji` to katalog aplikacji pod `/docker/apps`, a ostatnia linia przywraca też najświeższe zrzuty baz, które skrypt kopii z `docker01` zostawił w migawce)

Migawka ma bezwzględne ścieżki, więc `--target /` odtwarza pliki w tych samych miejscach, w których leżały na `docker01`. Plik z hasłem repozytorium nazywa się tak samo jak na stronie o kopii całego hosta (`restic-haslo.txt`), żeby nie mieć dwóch nazw.

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

ZMIENIA SYSTEM

Planowe przełączenie na docker02: systemctl stop keepalived

```
sudo systemctl stop keepalived
```

Wykonujemy na aktywnym hoście. Zatrzymanie keepalived oddaje adres wirtualny partnerowi i wywołuje rolę `stop`, która usuwa stosy na tym hoście, więc nie zostają dwie aktywne kopie.

systemctl = system control: sterowanie usługami systemd

ZMIENIA SYSTEM

Zmiana rekordu DNS na rezerwę: Set-DnsServerResourceRecord

```
$stary = Get-DnsServerResourceRecord -ZoneName "zielona.example" -Name "NazwaRekordu" -RRType CNAME
$nowy = $stary.Clone()
$nowy.RecordData.HostNameAlias = "NazwaHostaDocelowego.ad.zielona.example."
Set-DnsServerResourceRecord -ZoneName "zielona.example" -OldInputObject $stary -NewInputObject $nowy
```

> (wykonujemy na `dc01` w PowerShellu administratora, gdzie `NazwaRekordu` to na przykład `git`, a `NazwaHostaDocelowego` to `docker02`)

To droga zapasowa, gdy adres wirtualny z jakiegoś powodu nie wchodzi w grę. Czas, po którym klienci zobaczą zmianę, zależy od TTL rekordu.

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **TTL** = Time To Live: czas życia rekordu lub danych · **CNAME** = Canonical Name: rekord DNS będący aliasem, drugą nazwą wskazującą na już istniejący rekord A

TYLKO ODCZYT

Sprawdzenie, czy usługa odpowiada: curl

```
curl -ks -o /dev/null -w '%{http_code}\n' https://NazwaUslugi.zielona.example/
```

Przykład:

```
200
```

curl = client URL: pobieranie i wysyłanie przez HTTP z terminala

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 10:30, okno serwisowe. Tomek robi pierwszy planowy test przełączenia z `docker01` na `docker02` i po zatrzymaniu keepalived widzi, że adres wirtualny przeszedł, a stosy wstały. Gitea na `docker02` kończy jednak pracę z błędem połączenia z bazą, bo `pg01` nie ma w `pg_hba.conf` adresu `10.20.10.47` i odrzuca nowego klienta. Dobrze, że to środa i test, a nie piątek o 16:00: dopisanie jednej linii na `pg01` zajęło trzy minuty i trafiło na listę kontrolną przygotowania `docker02`.

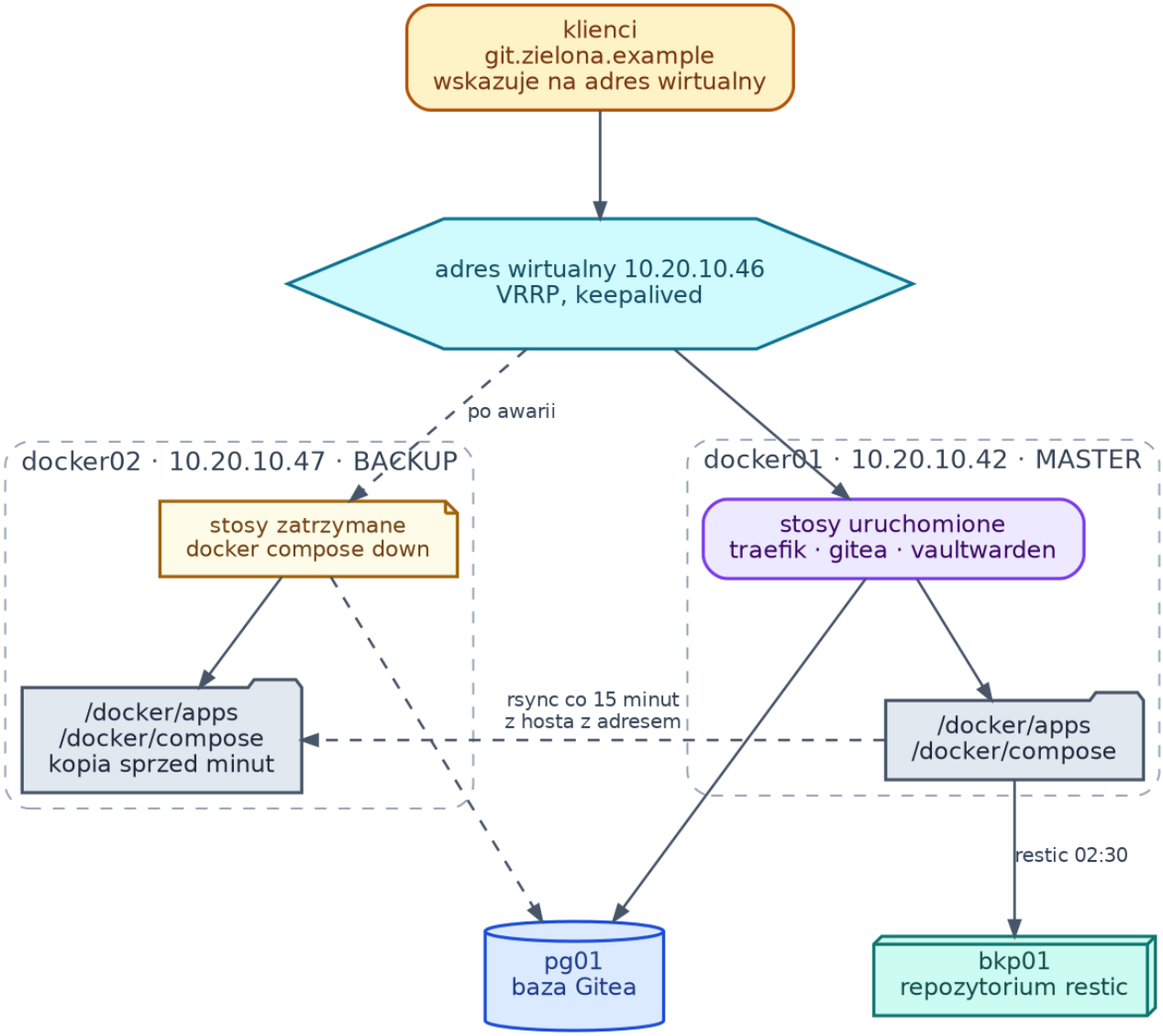
Jak to działa

Rozwiń: jak to działa pod spodem

Co przenosimy: przepis i dane

Pojęcie: stos. Jeden katalog z plikiem `compose.yaml` i `.env`, który uruchamia jedną aplikację razem z jej pomocniczymi kontenerami (na przykład Gitea, albo Traefik). Zakładamy, że stos opisuje całą aplikację, a jej trwałe dane leżą pod `/docker/apps/<nazwa>`.

Kontener jest zawsze odtwarzalny z obrazu, więc przenoszenie sprowadza się do trzech rzeczy: przepisu uruchomieniowego (`/docker/compose/<nazwa>`), danych (`/docker/apps/<nazwa>`) i obrazu o tej samej, przypiętej wersji. Mechanizm `docker checkpoint` oparty na CRIU jest w dokumentacji Dockera oznaczony jako eksperymentalny i nastawiony na jeden host, więc nie opieramy na nim niczego produkcyjnego. Dlatego u nas przenoszenie zawsze znaczy: te same pliki na drugim hoście i nowy start.



Rysunek 1. Architektura pary hostów: adres wirtualny prowadzi do aktywnego docker01, docker02 trzyma kopię plików i zatrzymane stosy, baza Gitea leży na pg01, a restic z docker01 trafia na bkp01. (kliknięcie otwiera pełny rozmiar)

Zasady, bez których przenoszenie nie działa

1. Wszystko, co potrzebne do startu, leży w `/docker/compose/<nazwa>` i `/docker/apps/<nazwa>`. Nic w wolumenach nazwanych, nic w `/var/lib/docker`.
2. Katalog `/docker/docker-data` (u nas `data-root` Dockera) nie jest kopiowany na drugi host. Obrazy drugi host pobiera sam, dlatego `docker compose pull` na `docker02` robimy z wyprzedzeniem, a nie w dniu awarii, kiedy rejestr może być niedostępny.

3. Plik `compose.yaml` trzymamy w repozytorium konfiguracji w Gitea, ale Gitea stoi na `docker01`. Dlatego kopię `/docker/compose` ma też `docker02`, niezależnie od Gitea, i ma ją restic.
4. Plik `.env` nie trafia do Gita. Hasła są w Vaultwardenie, a Vaultwarden też stoi na `docker01`, więc hasło repozytorium restic i hasła krytyczne mają jeszcze jedną kopię poza `docker01` (wydruk w kopercie albo zaszyfrowany plik poza serwerem), opisaną na stronie o sekretach.
5. Bazy danych trzymamy poza hostem (`pg01` dla Gitea) albo przenosimy ich zrzut logiczny. Pliki działającej bazy w kontenerze nie nadają się do kopiowania `rsync` na żywo.
6. Adresy w konfiguracji aplikacji to nazwy z DNS (`git.zielona.example`), nigdy nazwa hosta ani adres IP. Inaczej po przełączeniu odnośniki do klonowania wskazują martwy host.
7. W danej chwili działa jedna kopia każdej aplikacji. W stosach objętych przełączaniem dajemy `restart: "no"`, a stosy uruchamia i usuwa rola nadana przez keepalived.

Ostatnia zasada wynika z zachowania Dockera: polityka `always` uruchamia kontener po restarcie demona nawet wtedy, gdy zatrzymaliśmy go ręcznie, a `unless-stopped` pamięta ręczne zatrzymanie. Host rezerwowy po restarcie nie może sam wstawać z aplikacjami, bo wtedy dwie kopie piszą jednocześnie do `pg01`, każda ze swoimi plikami.

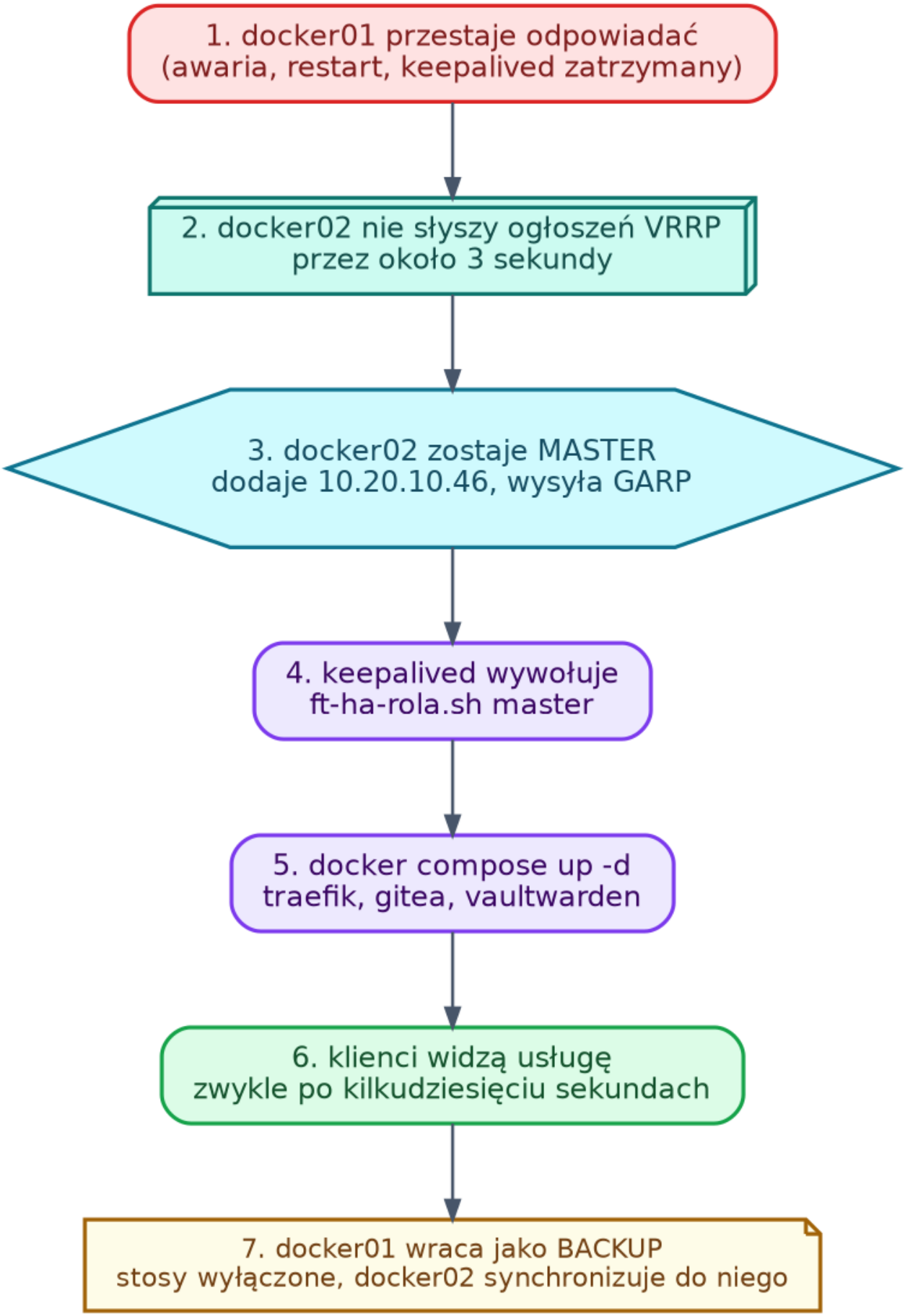
> ⚠ W YAML wartość `no` bez cudzysłowu bywa czytana jako fałsz logiczny. W `compose.yaml` zawsze piszemy `restart: "no"` z cudzysłowem.

Którędy idzie przełączenie

Keepalived na obu hostach rozmawia protokołem VRRP. Aktywny host (MASTER) trzyma adres wirtualny i co sekundę ogłasza się partnerowi. Gdy partner przestaje słyszeć ogłoszenia, według RFC 5798 uznaje aktywnego za martwego po trzykrotności interwału ogłoszeń plus niewielki zapas, czyli przy interwale 1 sekundy po około 3 do 4 sekundach, przejmuje adres i rozsyła gratisowy ARP, żeby przełączniki i `fw01` zaktualizowały swoje tablice.

> **Pojęcie: gratisowy ARP (GARP).** Ramka, którą host wysyła sam z siebie, żeby ogłosić całej sieci: ten adres IP jest teraz pod moim adresem MAC. Dzięki niej klienci nie czekają, aż wygaśnie stary wpis w ich tablicach ARP.

Przejęcie adresu to dopiero początek. Keepalived wywołuje potem skrypt roli, który uruchamia stosy, i dopiero po kilkudziesięciu sekundach (czas startu aplikacji) usługa odpowiada.


1. docker01 przestaje odpowiadać
(awaria, restart, keepalived zatrzymany)
2. docker02 nie słyszy ogłoszeń VRRP
przez około 3 sekundy
3. docker02 zostaje MASTER
dodaje 10.20.10.46, wysyła GARP
4. keepalived wywołuje
ft-ha-rola.sh master
5. docker compose up -d
traefik, gitea, vaultwarden
6. klienci widzą usługę
zwykle po kilkudziesięciu sekundach
7. docker01 wraca jako BACKUP
stosy wyłączone, docker02 synchronizuje do niego

Rysunek 2. Przełączenie krok po kroku: od zaniku ogłoszeń VRRP przez przejęcie adresu i start stosów do powrotu docker01 w roli rezerwy. (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: nopreempt.** Opcja keepalived, po której host o wyższym priorytecie, wracający po awarii, nie odbiera roli aktywnego od hosta, który ją obecnie pełni. U nas jest konieczna: po awarii `docker01` wraca jako rezerwa, a dane płyną od `docker02` do niego, nie odwrotnie.

Poziomy rezerwy w Zielonej

Poziom	Co robimy	RPO (ile tracimy)	RTO (ile trwa)	Kiedy
Zimna	restic odtwarza pliki z `bkp01` na `docker02`, DNS wskazuje na `docker02`	do doby, bo kopia jest o 02:30	godzina albo więcej (zależy od danych)	NetBox, Graylog, Open WebUI, wszystko po utracie obu hostów
Ciepła	`docker02` ma pliki sprzed kwadransa, keepalived przenosi adres i startuje stosy	do kilkunastu minut	od kilkudziesięciu sekund do kilku minut	Traefik, Gitea, Vaultwarden, docelowo MailCow

To są rzędy wielkości do zmierzenia w teście, nie obietnice. Pojęcia RPO i RTO tłumaczy strona RPO i RTO: dwie liczby, od których zaczyna się każdy plan backupu, a dobór poziomu do usługi strona Wysoka dostępność: pojęcia, liczenie dostępności i poziomy rezerwy.

Przebieg

> 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.

Część 1. Przygotowanie docker02 (dla obu poziomów)

Zakładamy maszynę wirtualną `docker02` na `hv02`, z tym samym systemem (Rocky Linux 9), tym samym układem LVM (`vgDocker/lvDocker` pod `/docker`) i tą samą wersją Dockera co `docker01`, według strony Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker. Adres `10.20.10.47` dopisujemy do DNS i do NetBoksa.

Wersje Dockera porównujemy na obu hostach:

```
docker version --format '{{.Server.Version}}'
```

Maszyny `docker01` i `docker02` mają stać na różnych hostach klastra. Na `hv01` ustawiamy regułę rozdzielania (miękką, domyślną), która w czasie pracy utrzymuje je na osobnych węzłach, ale przy wyłączeniu jednego węzła pozwala im chwilowo stać razem:

```
$AntiAffinity = New-Object System.Collections.Specialized.StringCollection
$AntiAffinity.Add("Docker")
(Get-ClusterGroup -Name "docker01").AntiAffinityClassNames = $AntiAffinity
(Get-ClusterGroup -Name "docker02").AntiAffinityClassNames = $AntiAffinity
```

Reguła zadziała po odtworzeniu zasobów, więc istniejące już maszyny rozdzielamy migracją na żywo, według strony Hyper-V na co dzień: przełącznik, maszyny, punkty kontrolne i migracja. Twardego wariantu (`ClusterEnforcedAntiAffinity`) w klastrze dwuwęzłowym nie włączamy, bo po awarii jednego węzła obie maszyny zostałyby wyłączone.

Na `docker02` zakładamy katalogi i sieć Traefika, tak jak na `docker01`:

```
sudo mkdir -p /docker/apps /docker/compose /docker/skrypty
docker network create proxy
```

Na obu hostach wpuszczamy protokół VRRP od partnera (adres partnera podmieniamy na właściwy: na `docker01` wpisujemy `10.20.10.47`, na `docker02` wpisujemy `10.20.10.42`). Szersze omówienie reguł z protokołem opisuje strona firewalld:

```
sudo firewall-cmd --add-rich-rule='rule family="ipv4" source address="AdresPartnera/32"
protocol value="vrrp" accept' --permanent
sudo firewall-cmd --reload
```

Wymiana plików między hostami idzie po SSH, kluczem bez hasła, ograniczonym do adresu partnera. Na `docker01` tworzymy klucz i dopisujemy jego część publiczną na `docker02` (na `docker02` robimy to samo w przeciwną stronę):

```
sudo ssh-keygen -t ed25519 -N "" -f /root/.ssh/ft-ha
sudo cat /root/.ssh/ft-ha.pub
```

W pliku `/root/.ssh/authorized_keys` partnera dopisujemy linię z wklejonym kluczem, poprzedzoną ograniczeniem adresu:

```
from="AdresDrugiegoHosta",restrict ssh-ed25519 AAAA...resztaKlucza ft-ha
```

> (gdzie `AdresDrugiegoHosta` to adres hosta, który się łączy, a `restrict` wyłącza przekierowania portów i terminal; sprawdzamy w `man sshd`, czy opcje działają tak samo w naszej wersji OpenSSH)

W `/root/.ssh/config` na obu hostach wskazujemy ten klucz:

```
Host docker01 docker02
    IdentityFile /root/.ssh/ft-ha
```

Na `pg01` dopisujemy `docker02` do reguł uwierzytelniania, obok istniejącej linii dla `10.20.10.42`. Zasady zapisu `pg_hba.conf` opisuje strona Uwierzytelnianie i bezpieczeństwo połączeń:

```
host    gitea    gitea    10.20.10.47/32    scram-sha-256
```

```
sudo -u postgres psql -c "SELECT pg_reload_conf();"
```

Do listy kontrolnej `docker02` należą jeszcze: agent Zabbixa i monitoring tych samych usług co na `docker01`, wpis w NetBoksie, ta sama reguła przekierowania portów na `fw01`, ale wskazująca adres wirtualny, i obrazy pobrane z wyprzedzeniem (`docker compose pull` w każdym katalogu Compose, po jednej synchronizacji plików z części 3).

Część 2. Zimna rezerwa: odtworzenie z restic na docker02

Zimna rezerwa nie wymaga niczego poza kopią opisaną na stronie Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach. Na `docker02` instalujemy restic w tej samej wersji (0.19.1), kopiujemy plik z hasłem repozytorium i dopisujemy klucz SSH `docker02` na `bkp01`, dla tego samego konta co `docker01`. Hasło repozytorium ma kopię poza Vaultwardenem, patrz zasada 4.

Odtworzenie jednej aplikacji ma stałą kolejność: pliki z migawki, zrzut bazy do pustej bazy (jeśli aplikacja ją ma w kontenerze), `docker compose pull`, `docker compose up -d`, sprawdzenie, na końcu DNS. Dla Gitea z bazą na `pg01` odpada krok zrzutu:

```
export RESTIC_REPOSITORY=sftp:nazwa_usera@bkp01:/backup/docker01-restic
export RESTIC_PASSWORD_FILE=/docker/skrypty/restic-haslo.txt
sudo -E restic snapshots --host docker01 --tag docker01
sudo -E restic restore latest --host docker01 --tag docker01 --target / \
  --include /docker/apps/gitea --include /docker/compose/gitea
cd /docker/compose/gitea
docker compose pull
docker compose up -d
```

Vaultwarden ma od 11 października 2026 osobne repozytorium `vaultwarden` (poziom 365 dni według strony Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje), więc jego dane odtwarzamy tymi samymi poleceniami, ale z `RESTIC_REPOSITORY` i plikiem hasła repozytorium `vaultwarden` ze strony o kopii całego hosta, z `--tag vaultwarden` i `--include /docker/apps/vaultwarden`, a plik Compose z repozytorium `docker01`. Tak samo GLPI (od 11 października 2026 repozytorium `glpi`, poziom 5 lat): z `--tag glpi` odtwarzamy `/docker/apps/glpi/glpi`, `/docker/compose/glpi` i zrzut `/docker/kopie/glpi/glpi.sql`, który wczytujemy do pustej bazy w kontenerze `db`.

Gdy po awarii `docker01` zalega blokada repozytorium po przerwanej kopii, restic odmawia pracy komunikatem o zablokowanym repozytorium. Wtedy sprawdzamy, czy na pewno nic innego nie pisze, i dopiero potem zdejmujemy blokadę poleceniem `restic unlock`.

Na koniec przestawiamy rekord DNS (karta w Fiszce) i mierzymy cały czas od decyzji do odpowiedzi usługi. Dla zbioru 62 GiB z przykładu na stronie o kopii hosta sama transmisja przez sieć 1 Gb/s to co najmniej kilkanaście minut, więc zimny RTO planujemy w godzinach, a prawdziwą liczbę poznajemy dopiero z testu odtworzenia.

Część 3. Ciepła rezerwa: synchronizacja co 15 minut

Cztery pliki i jedna lista aplikacji. Wszystkie leżą identycznie na obu hostach, różni się tylko wiersz `PARTNER` w pliku konfiguracyjnym.

Plik `/etc/ft-ha/ha.conf`, wspólny dla skryptów (na `docker02` wpisujemy `PARTNER=docker01`):

```
# /etc/ft-ha/ha.conf
# Wspólna konfiguracja skryptów HA. Na docker02 zamieniamy tylko PARTNER na docker01.
VIP=10.20.10.46
BRAMA=10.20.10.1
INTERFEJS=ens192
PARTNER=docker02
```

```
# Lista stosów w kolejności uruchamiania, w formie nazwa:tryb.
# live: kopiujemy pliki podczas pracy aplikacji (dane w plikach, baza poza hostem).
# stop: na czas kopiowania zatrzymujemy stos (SQLite i inne pliki baz).
APLIKACJE="traefik:live gitea:live vaultwarden:stop"
```

Plik `/usr/local/sbin/ft-ha-sprawdz.sh`, który keepalived wywołuje co kilka sekund. Sprawdza tylko to, co decyduje o całym hoście: Docker, bramę `fw01` i Traefika. Awaria jednej aplikacji, na przykład Gitea, to sprawa dla Zabbixa i dla nas, a nie powód do przeniesienia całego hosta na kopię sprzed kilkunastu minut.

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-sprawdz.sh
# Wywoływany przez keepalived co kilka sekund.
# Kod 0: host zdolny do roli aktywnej. Kod 1: host niezdolny.
# Sprawdzamy tylko to, co decyduje o całym hoście, nie pojedyncze aplikacje.

set -u
source /etc/ft-ha/ha.conf

# 1. Docker musi działać.
systemctl is-active --quiet docker || exit 1

# 2. Brama fw01 musi odpowiadać: host odcięty od sieci nie może trzymać adresu.
ping -c 1 -W 1 "${BRAMA}" > /dev/null 2>&1 || exit 1

# 3. Host aktywny od ponad dwóch minut musi mieć działający Traefik.
if ip -4 -o addr show dev "${INTERFEJS}" | grep -q " ${VIP}/"; then
  START="$(cat /run/ft-ha/aktywny-od 2> /dev/null || echo 0)"
  TERAZ="$(date +%s)"
  if [ "${START}" -gt 0 ] && [ $((TERAZ - START)) -gt 120 ]; then
    KOD="$(curl -ks --max-time 3 -o /dev/null -w '%{http_code}' https://127.0.0.1/ || true)"
    if [ -z "${KOD}" ] || [ "${KOD}" = "000" ]; then
      exit 1
    fi
  fi
fi

exit 0
```

Dwie minuty karencji dajemy po to, żeby zbyt wolny start stosów po przełączeniu nie wywołał kolejnego przełączenia. Odpowiedź Traefika o dowolnym kodzie HTTP (także 404) znaczy, że proxy

żyje, a kod `000` znaczy, że nic nie słucha.

Plik `/usr/local/sbin/ft-ha-rola.sh`, który keepalived wywołuje przy każdej zmianie roli. Rola master uruchamia stosy w kolejności z listy, pozostałe je usuwają:

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-rola.sh
# Wywoływany przez keepalived przy zmianie roli: master, backup, fault, stop.
# master uruchamia stosy, pozostałe role je usuwają (docker compose down).

set -u
source /etc/ft-ha/ha.conf

ROLA="${1:-}"
LOG=/var/log/ft-ha.log
log() { echo "$(date -Iseconds) $(hostname -s) $*" >> "${LOG}"; }

mkdir -p /run/ft-ha
exec 9> /run/ft-ha/rola.lock
flock 9

case "${ROLA}" in
  master)
    log "rola master: uruchamiam stosy"
    date +%s > /run/ft-ha/aktywny-od
    for wpis in ${APLIKACJE}; do
      app="${wpis%%:*}"
      if ( cd "/docker/compose/${app}" && docker compose up -d ) >> "${LOG}" 2>&1; then
        log "stos ${app}: uruchomiony"
      else
        log "BLAD: stos ${app} nie wystartował"
      fi
    done
    ;;
  backup | fault | stop)
    log "rola ${ROLA}: usuwam stosy"
    rm -f /run/ft-ha/aktywny-od
    for wpis in ${APLIKACJE}; do
      app="${wpis%%:*}"
      if ! ( cd "/docker/compose/${app}" && docker compose down ) >> "${LOG}" 2>&1; then
```

```
        log "BLAD: stos ${app} nie został usunięty"
      fi
    done
    ;;
  *)
    echo "użycie: ft-ha-rola.sh master|backup|fault|stop" >&2
    exit 2
    ;;
esac
```

Plik `/usr/local/sbin/ft-ha-synchronizuj.sh`, który działa tylko na hoście z adresem wirtualnym. Przed kopiowaniem sprawdza trzy rzeczy, które chronią przed najgorszym: czy `/docker` jest zamontowany (puste źródło z `--delete` skasowałoby dane partnera), czy partner odpowiada i czy partner nie trzyma tego samego adresu (to znak split brain, wtedy nie nadpisujemy niczego):

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-synchronizuj.sh
# Kopiuje /docker/apps/<app> i /docker/compose/<app> na partnera.
# Działa wyłącznie na hoście, który trzyma adres wirtualny.

set -u
source /etc/ft-ha/ha.conf

LOG=/var/log/ft-ha.log
SSH="ssh -o BatchMode=yes -o ConnectTimeout=5"
log() { echo "$(date -Iseconds) $(hostname -s) $*" >> "${LOG}"; }

mkdir -p /run/ft-ha
exec 8> /run/ft-ha/sync.lock
flock -n 8 || { log "synchronizacja już trwa"; exit 0; }

# 1. Nie trzymamy adresu wirtualnego: to nie nasza kolej, nic nie robimy.
if ! ip -4 -o addr show dev "${INTERFEJS}" | grep -q " ${VIP}/"; then
  exit 0
fi

# 2. Zawsze kopiujemy z działającego wolumenu, nigdy z pustego katalogu.
if ! mountpoint -q /docker; then
  log "BLAD: /docker nie jest zamontowany, przerywam"
  exit 1
```

```
fi

# 3. Partner musi odpowiadać i nie może trzymać adresu (podejrzenie split brain).
if ! ${SSH} "root@${PARTNER}" true; then
  log "BLAD: partner ${PARTNER} nieosiągalny"
  exit 1
fi
if ${SSH} "root@${PARTNER}" "ip -4 -o addr show dev ${INTERFEJS} | grep -q ' ${VIP}/'"; then
  log "BLAD: oba hosty trzymają ${VIP}, przerywam synchronizację"
  exit 1
fi

BLAD=0
for wpis in ${APLIKACJE}; do
  app="${wpis%%:*}"
  tryb="${wpis##*:}"
  ${SSH} "root@${PARTNER}" "mkdir -p /docker/apps/${app} /docker/compose/${app}"
  if [ "${tryb}" = "stop" ]; then
    ( cd "/docker/compose/${app}" && docker compose stop ) >> "${LOG}" 2>&1
  fi
  for katalog in apps compose; do
    if ! rsync -aHAX --numeric-ids --delete -e "${SSH}" \
        "/docker/${katalog}/${app}/" "root@${PARTNER}:/docker/${katalog}/${app}/" >> "${LOG}"
2>&1; then
      log "BLAD: rsync ${katalog}/${app}"
      BLAD=1
    fi
  done
  if [ "${tryb}" = "stop" ]; then
    ( cd "/docker/compose/${app}" && docker compose start ) >> "${LOG}" 2>&1
  fi
done

log "synchronizacja zakończona, kod ${BLAD}"
exit "${BLAD}"
```

Wszystkie trzy skrypty nadajemy z prawami root i wykonywalne, a plik z hasłami (`.env`) jedzie razem z katalogiem `compose`, więc połączenie SSH między hostami traktujemy jak kanał do sekretów:

```
sudo mkdir -p /etc/ft-ha
sudo chmod 755 /usr/local/sbin/ft-ha-sprawdz.sh /usr/local/sbin/ft-ha-rola.sh
/usr/local/sbin/ft-ha-synchronizuj.sh
sudo chown root:root /usr/local/sbin/ft-ha-*.sh
```

Timer systemd. Minuty `07`, `22`, `37` i `52` omijają godziny 02:30, 02:45 i 03:00, o których działają kopie restic `docker01`, Vaultwardena i GLPI, bo kopia i synchronizacja z zatrzymaniem stosu Vaultwardena nie mogą się nakładać. Szerzej o timerach pisze strona systemd timery: OnCalendar zamiast crona.

```
sudo vim /etc/systemd/system/ft-ha-synchronizuj.service
```

```
[Unit]
Description=Synchronizacja plików aplikacji na partnera HA
After=network-online.target
Wants=network-online.target

[Service]
Type=oneshot
ExecStart=/usr/local/sbin/ft-ha-synchronizuj.sh
```

```
sudo vim /etc/systemd/system/ft-ha-synchronizuj.timer
```

```
[Unit]
Description=Synchronizacja HA co 15 minut

[Timer]
OnCalendar=*:07/15
RandomizedDelaySec=20
Persistent=false

[Install]
WantedBy=timers.target
```

```
sudo systemctl daemon-reload
sudo systemctl enable --now ft-ha-synchronizuj.timer
```

Jeden timer na obu hostach wystarcza, bo skrypt na hoście rezerwowym kończy pracę od razu. Po zamianie ról kopiowanie samo zmienia kierunek.

Stos Gitea w wersji do przełączania. Różnice wobec strony o instalacji w Dockerze to `restart: "no"`, sieć `proxy` z etykietami Traefika (jak na stronie o Traefiku) i brak publikowania portu 3000 na hoście. Plik `/docker/compose/gitea/.env` zawiera tylko `GITEA_DB_PASSWORD`, tak samo jak dotąd:

```
services:
  server:
    image: docker.gitea.com/gitea:1.27.3
    container_name: gitea
    restart: "no"
    environment:
      - USER_UID=1000
      - USER_GID=1000
      - GITEA__database__DB_TYPE=postgres
      - GITEA__database__HOST=pg01:5432
      - GITEA__database__NAME=gitea
      - GITEA__database__USER=gitea
      - GITEA__database__PASSWD=${GITEA_DB_PASSWORD}
    volumes:
      - /docker/apps/gitea/data:/data
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "2222:22"
    networks:
      - default
      - proxy
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.gitea.rule=Host(`git.zielona.example`)"
      - "traefik.http.routers.gitea.entrypoints=websecure"
      - "traefik.http.routers.gitea.tls=true"
      - "traefik.http.services.gitea.loadbalancer.server.port=3000"

networks:
  default:
  proxy:
    external: true
```

Ten sam zabieg `restart: "no"` robimy w stosach Traefika i Vaultwardena. Port SSH `2222` jest opublikowany na wszystkich adresach hosta, więc działa też na adresie wirtualnym.

Ciepła rezerwa wymaga jeszcze jednej zmiany w istniejącej kopii restic. Skrypty `kopia-docker01-restic.sh`, `kopia-vaultwarden-restic.sh` i `kopia-glpi-restic.sh` ze strony o kopii całego hosta mają działać tylko na aktywnym hoście, inaczej `docker02` zrobi kopię danych sprzed kwadransa obok kopii prawdziwych. Na początku każdego z nich, po `set -euo pipefail`, na obu hostach dopisujemy:

```
# Kopia tylko z hosta, który trzyma adres wirtualny (patrz strona o przenoszeniu kontenerów).
source /etc/ft-ha/ha.conf
if ! ip -4 -o addr show dev "${INTERFEJS}" | grep -q " ${VIP}/"; then
  exit 0
fi
```

Do rozważenia przez autora: żeby historia migawek była jedna, niezależna od tego, który host jest aktywny, w poleceniu `restic backup` można dopisać `--host docker-ha` na obu hostach, a w poleceniach odtwarzania pisać `--host docker-ha`. Strona o kopii hosta tego dziś nie robi, więc ją zostawiamy bez zmian.

Część 4. Przełączenie adresem wirtualnym (keepalived)

Instalujemy keepalived na obu hostach (w EL9 to wersja 2.2.x z AppStream):

```
sudo dnf install -y keepalived
```

Keepalived ma wstać po Dockerze, bo rola master wywołuje `docker compose`. Dopisujemy zależność:

```
sudo mkdir -p /etc/systemd/system/keepalived.service.d
sudo vim /etc/systemd/system/keepalived.service.d/docker.conf
```

```
[Unit]
After=docker.service
Wants=docker.service
```

Plik `/etc/keepalived/keepalived.conf` na `docker01`:

Pokaż pełny przykład (43 linii)

```
global_defs {
    router_id docker01
    enable_script_security
```

```
    script_user root
}

# Brak jawnej wagi: domyślna waga 0 oznacza, że po trzech porażkach z rzędu
# instancja przechodzi w stan FAULT i oddaje adres partnerowi.
vrrp_script ft_ha_sprawdz {
    script "/usr/local/sbin/ft-ha-sprawdz.sh"
    interval 5
    timeout 4
    fall 3
    rise 2
}

vrrp_instance VI_DOCKER {
    # Oba hosty startują jako BACKUP, a nopreempt działa tylko przy takim stanie
początkowym.
    state BACKUP
    nopreempt
    interface ens192
    virtual_router_id 46
    priority 150
    advert_int 1

    unicast_src_ip 10.20.10.42
    unicast_peer {
        10.20.10.47
    }

    virtual_ipaddress {
        10.20.10.46/24 dev ens192
    }

    track_script {
        ft_ha_sprawdz
    }

    notify_master "/usr/local/sbin/ft-ha-rola.sh master"
    notify_backup "/usr/local/sbin/ft-ha-rola.sh backup"
    notify_fault  "/usr/local/sbin/ft-ha-rola.sh fault"
```

```
    notify_stop   "/usr/local/sbin/ft-ha-rola.sh stop"
}
```

Na `docker02` ten sam plik zmienia się w czterech miejscach:

Parametr	docker01	docker02
`router_id`	`docker01`	`docker02`
`priority`	`150`	`100`
`unicast_src_ip`	`10.20.10.42`	`10.20.10.47`
`unicast_peer`	`10.20.10.47`	`10.20.10.42`

Unicast zamiast domyślnej grupy multicast wybieramy dlatego, że ogłoszenia idą wprost od hosta do hosta i nie zależą od przełączników wirtualnych ani od obsługi multicastu w sieci. `virtual_router_id` ma być unikalny w całym VLAN serwerowym, a `46` pasuje do ostatniego oktetu adresu, żeby łatwo go zapamiętać. Plik sprawdzamy komendą z Fiszki, a potem uruchamiamy na obu hostach, najpierw na `docker02`, żeby nie dostać dwóch aktywnych na starcie:

```
sudo keepalived --config-test
sudo systemctl enable --now keepalived
```

> Na Rocky Linux z SELinux w trybie enforcing skrypty wywołane przez keepalived mogą dostać odmowę dostępu do gniazda Dockera. Po pierwszym uruchomieniu sprawdzamy `getenforce` i `sudo ausearch -m avc -ts recent`. Jeśli widzimy odmowy dla `keepalived_t`, w teście przełączenia stosy nie wstaną, a wyjątek w polityce SELinux omawiamy z autorem, zamiast wyłączać SELinux. Zachowania w naszym środowisku nie mieliśmy jak sprawdzić, więc to punkt obowiązkowego testu.

Na koniec przenosimy nazwy aplikacji z `docker01` na adres wirtualny. Zakładamy jeden rekord `A` dla adresu wirtualnego i kierujemy na niego `CNAME` wszystkich aplikacji z `docker01`:

```
Add-DnsServerResourceRecordA -ZoneName "ad.zielona.example" -Name "docker-ha" -IPv4Address
"10.20.10.46" -CreatePtr
Add-DnsServerResourceRecordCName -ZoneName "zielona.example" -Name "git" -HostNameAlias
"docker-ha.ad.zielona.example"
```

Rekordy już istniejące zmieniamy komendą z Fiszki. Przekierowania portów z internetu na `fw01` (80, 443 dla Let's Encrypt i ruchu z zewnątrz) kierujemy na adres wirtualny, nie na `docker01`, inaczej po przełączeniu świat zewnętrzny dalej puka do martwego hosta.

Certyfikat serwera `wildcard-zielona.crt` z kluczem (wystawiony przez `CA-Zielona`) jest w katalogu danych Traefika, więc jedzie z synchronizacją. To samo dotyczy pliku `acme.json` z certyfikatami

Let's Encrypt. Otwarta wersja Traefika nie dzieli wyzwań ACME między równolegle działające instancje, a opiekunowie projektu odradzają współdzielenie `acme.json`, ale u nas drugi Traefik w danej chwili nie działa, więc kopia pliku wystarcza.

> Po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona nowe certyfikaty wystawia `Zielona-CA01` na `ca01`, więc wszędzie, gdzie dziś ufamy tylko `CA-Zielona`, dokładamy zaufanie do `Zielona-Root-CA`.

Część 5. Przełączenie przez DNS (gdy adres wirtualny nie wchodzi w grę)

Cecha	Adres wirtualny (keepalived)	Zmiana rekordu DNS
Czas przełączenia	kilka sekund na adres, potem czas startu aplikacji	TTL rekordu plus pamięci podręczne klientów, przeglądarek i aplikacji Java
Działa między sieciami	nie, oba hosty muszą być w jednej podsieci	tak
Ryzyko	split brain przy utracie łączności między hostami	zapomniany rekord, długie pamięci podręczne
Automatyka	tak, z naszymi skryptami	ręcznie albo własnym skryptem
Gdzie u nas	docker01 i docker02, ta sama podsieć 10.20.10.0/24	zimna rezerwa i przypadki szczególne

Dla zimnej rezerwy przełączamy DNS ręcznie, bo i tak decydujemy człowiekiem, a dla ciepłej rezerwy używamy adresu wirtualnego, bo sekundy ważą więcej niż prostota. Niezależnie od wyboru obniżamy TTL rekordów aplikacji do 300 sekund z wyprzedzeniem, najlepiej na dzień przed planowym testem, a po teście wracamy do zwykłej wartości.

Część 6. Wariant z ZFS send/receive zamiast rsync

ZFS zamiast rsync daje spójne migawki z jednej chwili i przyrostową wysyłkę samych zmienionych bloków, więc RPO może spaść do kilku minut. Cena to zmiana układu dysków: `/docker/apps` i `/docker/compose` muszą być osobnymi zbiorami danych ZFS (na nowym dysku, obok woluminu `vgDocker/lvDocker`, który zostaje pod `/docker/docker-data`), a moduł ZFS na Rocky Linux 9 pochodzi z repozytorium projektu OpenZFS, nie z dystrybucji. Wersję i sposób instalacji (DKMS albo moduł śledzący kABI) sprawdzamy w dokumentacji OpenZFS w dniu wdrożenia. Szkic skryptu poniżej przeszedł tylko `bash -n`, nie był uruchamiany na prawdziwej puli, więc najpierw próbujemy go w laboratorium.

Zbiory danych tworzymy tak, żeby miały stałe punkty montowania na obu hostach:

```
sudo zfs create -o mountpoint=/docker/apps tank/docker-apps
sudo zfs create -o mountpoint=/docker/compose tank/docker-compose
```

Skrypt `/usr/local/sbin/ft-ha-zfs.sh` zastępuje wywołanie `rsync` w pliku synchronizacji (albo działa obok niego, wywoływany z tego samego timera):

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-zfs.sh
# Wariant ciepłej rezerwy na ZFS: migawka i przyrostowa wysyłka na partnera.
# Szkic: sprawdzony tylko składniowo, najpierw próba w laboratorium.
set -euo pipefail
source /etc/ft-ha/ha.conf

DATASETY="tank/docker-apps tank/docker-compose"
ZNACZNIK="ha-$(date +%Y%m%d-%H%M)"
ZACHOWAJ=24
SSH="ssh -o BatchMode=yes -o ConnectTimeout=5"

# Tylko host z adresem wirtualnym wysyła.
ip -4 -o addr show dev "${INTERFEJS}" | grep -q " ${VIP}/" || exit 0

for ds in ${DATASETY}; do
  zfs snapshot "${ds}@${ZNACZNIK}"

  # Ostatnia migawka ha-* po stronie partnera to punkt wyjścia przyrostu.
  POPRZEDNIA="$(${SSH} "root@${PARTNER}" "zfs list -H -t snapshot -o name -s creation -d 1
${ds}" \
    | grep "@ha-" | tail -n 1 | cut -d@ -f2 || true)"

  if [ -n "${POPRZEDNIA}" ]; then
    zfs send -i "@${POPRZEDNIA}" "${ds}@${ZNACZNIK}" | ${SSH} "root@${PARTNER}" "zfs receive -
s -u -F ${ds}"
  else
    zfs send "${ds}@${ZNACZNIK}" | ${SSH} "root@${PARTNER}" "zfs receive -s -u -F ${ds}"
  fi

  # Zostawiamy ostatnie ZACHOWAJ migawek ha-* po obu stronach.
  zfs list -H -t snapshot -o name -s creation -d 1 "${ds}" | grep "@ha-" | head -n "-
```

```
${ZACHOWAJ}" \
    | xargs -r -n 1 zfs destroy
  ${SSH} "root@${PARTNER}" "zfs list -H -t snapshot -o name -s creation -d 1 ${ds} | grep
'@ha-' | head -n -${ZACHOWAJ} | xargs -r -n 1 zfs destroy"
done
```

Opcja `-u` w `zfs receive` zostawia odebrane zbiory niezamontowane na partnerze, a `-s` pozwala wznowić przerwany transfer z tokenem. Opcja `-F` niszczy po stronie odbiorcy zmiany, których nie ma w nadawcy, więc skrypt wolno uruchamiać wyłącznie z hosta z adresem wirtualnym. W roli master na hoście z ZFS dopisujemy przed `docker compose up` jeszcze `zfs mount -a`, bo odebrane zbiory nie są zamontowane.

Test przełączenia

Testu nie robimy po raz pierwszy w dniu awarii i nie robimy go od 15 listopada do 6 stycznia, kiedy w Zielonej obowiązuje zakaz zmian. Przed testem zapisujemy godzinę ostatniej synchronizacji i godzinę ostatniego zatwierdzenia w Gitea.

Test planowy, w oknie serwisowym:

1. Na aktywnym hoście wymuszamy synchronizację i sprawdzamy wynik (karta „Wymuszenie synchronizacji” w Fiszce). W logu ma być linia z kodem 0.
2. Zapisujemy czas i na aktywnym `docker01` wykonujemy `sudo systemctl stop keepalived`.
3. Na `docker02` sprawdzamy adres (`ip -br -4 addr show`), listę kontenerów (`docker ps`) i odpowiedź usługi (`curl`). Mierzymy czas do kodu 200.
4. Logujemy się do Gitea i porównujemy ostatnie zatwierdzenie z zapisaną godziną. Różnica to nasze faktyczne RPO.
5. Na `docker01` uruchamiamy `sudo systemctl start keepalived`. Host ma wrócić jako BACKUP (opcja `nopreempt`), a jego stosy mają być usunięte. Czekamy na najbliższą synchronizację, która teraz idzie z `docker02` na `docker01`.
6. Rolę oddajemy tak samo, jak ją wzięliśmy: `stop`, a potem `start` keepalived na `docker02`, po świeżej synchronizacji w stronę `docker01`.

Test awaryjny, na klastrze Hyper-V, w tym samym oknie. Polecenie `Stop-ClusterGroup` wyłącza rolę bez restartu przez klaster, więc daje czystą awarię `docker01`, a `Start-ClusterGroup` ją kończy:

```
Stop-ClusterGroup -Name "docker01"
```

```
Start-ClusterGroup -Name "docker01"
```

Tu `docker01` znika bez wywołania roli `stop`, więc sprawdzamy, że `docker02` przejmuje adres sam po kilku sekundach i że po powrocie `docker01` jego stosy nie startują same (to dowód, że `restart: "no"`

działa). Osobno sprawdzamy migrację na żywo `docker01` między `hv01` a `hv02`: krótki zanik łączności nie powinien wywołać przełączenia, a jeśli wywołuje, zwiększamy `fall` w skrypcie sprawdzającym.

Test bezpieczeństwa, wyłącznie w laboratorium albo w oknie serwisowym przy obu hostach pod ręką: blokujemy VRRP na `docker02` (`firewall-cmd --remove-rich-rule` bez `--permanent`) i patrzymy, jak oba hosty uznają się za aktywne. W logu `/var/log/ft-ha.log` skrypt synchronizacji ma wtedy zgłosić `oba hosty trzymają`, a nie nadpisać dane. Regułę przywracamy i sprawdzamy, że wraca jeden aktywny. Tak wygląda split brain w praktyce i dlatego go ćwiczymy.

W wyniku testu wpisujemy do tabeli: czas przełączenia adresu, czas do odpowiedzi usługi, RPO i uwagi. Te liczby trafią do macierzy usług w kolejnej części księgi.

Wycofanie zmian

Wyłączamy automatykę i wracamy do jednego hosta, nie ruszając danych:

```
sudo systemctl disable --now ft-ha-synchronizuj.timer
sudo systemctl disable --now keepalived
```

Po wyłączeniu keepalived na obu hostach adres wirtualny znika. Na `docker01` stosy trzeba uruchomić ręcznie (`docker compose up -d` w katalogu każdej aplikacji), bo rola `stop` mogła je usunąć, a w plikach Compose przywracamy dotychczasową politykę `restart: always` albo `unless-stopped`. Rekordy DNS wracają na `docker01` komendą z Fiszki. Na `docker02` stosów nie zostawiamy uruchomionych.

Częste błędy

Objaw	Przyczyna	Co robimy
Gitea na `docker02` kończy z błędem o braku wpisu w `pg_hba.conf` dla `10.20.10.47`	`pg01` zna tylko adres `docker01`	dopisujemy linię dla `docker02`, `SELECT pg_reload_conf();`
Oba hosty trzymają adres wirtualny	zapora blokuje protokół VRRP albo `unicast_peer` wskazuje zły adres	sprawdzamy regułę z `protocol value="vrrp"` na obu hostach i `sudo tcpdump -ni ens192 proto 112`
Po restarcie hosta rezerwowego stosy wstają same	`restart: always` albo `unless-stopped` w `compose.yaml` zamiast `restart: "no"`	poprawiamy politykę, rolą `backup` usuwamy stosy
Kopia restic o 02:30 kończy się błędem przy Vaultwardenie	synchronizacja zatrzymała stos akurat w czasie kopii	timer na minuty `07/15`, kopia tylko z aktywnego hosta

Objaw	Przyczyna	Co robimy
Po przełączeniu stos nie startuje, brak obrazu	rejestr niedostępny, a obraz nie był pobrany na `docker02`	`docker compose pull` w każdym katalogu Compose z wyprzedzeniem
Brak sieci `proxy` przy starcie stosu	sieć założona tylko na `docker01`	`docker network create proxy` na `docker02`
Po przełączeniu świat zewnętrzny nie widzi usług	przekierowanie portów na `fw01` wskazuje `docker01`, nie adres wirtualny	zmieniamy cel przekierowania na `10.20.10.46`
Stosy nie wstają po przejęciu adresu, w dzienniku odmowy dla `keepalived_t`	SELinux blokuje wywołanie Dockera ze skryptu keepalived	`sudo ausearch -m avc -ts recent`, omawiamy poprawkę polityki, nie wyłączamy SELinux
Dane na `docker02` starsze niż 15 minut	timer w stanie `failed`, bo partner był nieosiągalny albo klucz SSH wygasł	`systemctl status ft-ha-synchronizuj.service`, log w `/var/log/ft-ha.log`, alarm w Zabbixie na stan `failed`
Odnośniki do klonowania pokazują `docker01` po przełączeniu	w `app.ini` Gitea stoi nazwa hosta, nie nazwa usługi	w `ROOT_URL` i `SSH_DOMAIN` wpisujemy `git.zielona.example`
Stary host po powrocie nadpisał dane nowego	brak `nopreempt` albo timer synchronizacji bez sprawdzania adresu	sprawdzamy `state BACKUP` i `nopreempt` w keepalived, skrypt z kontrolą adresu

□□ Ćwiczenia

1. Dopisujemy do listy `APLIKACJE` aplikację NetBox, która według strony o kopii hosta ma bazę PostgreSQL w kontenerze, i wybieramy tryb `live` albo `stop`. Uzasadniamy wybór.

Odpowiedź

Żaden z dwóch trybów nie jest dobry dla katalogu z plikami bazy działającej w kontenerze: `live` kopiuje niespójny obraz, a `stop` wymaga zatrzymania aplikacji na czas każdej synchronizacji co kwadrans. Właściwe wyjście to wyniesienie bazy NetBoksa na `pg01`, jak w wariancie produkcyjnym Gitea, po czym stos ma już tylko dane w plikach i pasuje do trybu `live`. Dopóki tego nie zrobimy, NetBox zostaje na zimnej rezerwie, czyli w restic ze zrzutem logicznym.

1. Wyjaśniamy, po co skrypt synchronizacji sprawdza, czy partner nie trzyma tego samego adresu wirtualnego.

Odpowiedź

Jeśli oba hosty trzymają adres, to znaczy, że nie widzą się nawzajem, a każdy uważa się za aktywny i może mieć nowsze dane od drugiego. Synchronizacja z `--delete` nadpisałaby wtedy dane partnera, który zapisywał prawdziwe zmiany użytkowników. Przerwanie synchronizacji i wpis w logu zostawiają obie wersje nietknięte do decyzji człowieka.

1. Timer synchronizuje co 15 minut, a pojedyncza synchronizacja trwa 3 minuty. Jakie jest najgorsze RPO przy awarii w złym momencie?

Odpowiedź

Najgorszy przypadek to awaria tuż przed końcem trwającej synchronizacji, która nie zdążyła się zakończyć. Wtedy `docker02` ma dane z poprzedniej, zakończonej synchronizacji, czyli sprzed 15 plus 3 minut, a przy nakładającym się opóźnieniu losowym (`RandomizedDelaySec`) i krótkim czasie przejęcia liczymy około 20 minut. Liczbę w Zielonej poznajemy z testu, mierząc różnicę między ostatnim zatwierdzeniem w Gitea a stanem na `docker02`.

1. Dlaczego hasło repozytorium restic nie może leżeć wyłącznie w Vaultwardenie na `docker01`?

Odpowiedź

Bo w dniu awarii `docker01` Vaultwarden jest tak samo niedostępny jak reszta, a bez hasła repozytorium nie odtworzymy niczego, w tym samego Vaultwardena. Drugą kopię hasła trzymamy poza `docker01`, na przykład w zaklejonej kopercie w sejfie albo w zaszyfrowanym pliku na `bkp01` z hasłem znanym obu administratorom.

Powiązane: Wysoka dostępność: usługa, która nie ma jednego serwera · Wysoka dostępność: pojęcia, liczenie dostępności i poziomy rezerwy · Replikacja danych między hostami: DRBD, GlusterFS, Ceph, ZFS i replikacja w aplikacji · Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach · Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Reverse proxy w Dockerze: Traefik przed wszystkimi aplikacjami na docker01 · Hasła i sekrety poza repozytorium: .env, Docker secrets i Vaultwarden · S2D: wymiana dysku bez przestoju, gdy pula jest prawie pełna

Źródła: docs.docker.com, rozdziały „Start containers automatically” i „docker checkpoint” · RFC 5798, Virtual Router Redundancy Protocol version 3 · keepalived.org (wydanie 2.4.3 z 20 lipca 2026) i keepalived.conf(5) · restic.readthedocs.io, rozdział „Restoring from backup” · learn.microsoft.com, „Cluster affinity” · doc.traefik.io i community.traefik.io, wątek o wielu instancjach z ACME · github.com/go-gitea/gitea, wydanie 1.27.3 z 29 sierpnia 2026 · openzfs.github.io, strony „zfs-send” i „RHEL-based distro”

Replikacja danych między hostami: DRBD, GlusterFS, Ceph, ZFS i replikacja w aplikacji

Poziom: średnio

Dotyczy: DRBD 9.3, GlusterFS 11, Ceph 20 (Tentacle), OpenZFS 2.4, rsync, replikacja baz, Rocky Linux 9

Czas: ~75 min

Kiedy dwa hosty mają trzymać te same dane, wybieramy, na jakim poziomie je kopiujemy: pliki, bloki dysku albo wewnątrz aplikacji. Na tej stronie porównujemy DRBD, GlusterFS, Ceph, replikację ZFS, rsync i replikację natywną baz, a na końcu uzasadniamy wybór dla Zielonej. Wracamy tu, gdy ktoś proponuje „dysk, który sam kopiuje się na drugi serwer”.

ZAŁOŻENIA SCENARIUSZA

replikacja idzie siecią serwerową VLAN 10 między dwiema maszynami `docker01` i `docker02` na `hv01` i `hv02`, w jednej serwerowni i bez drugiej lokalizacji. Trzeci głos (tiebreaker) ma stać na fizycznym `bkp01`, poza klastrem hostów.

Skróty użyte na tej stronie (21)

Skrót	Co znaczy
CDDL	Common Development and Distribution License: licencja ZFS, niezgodna z GPL, przez co moduł ZFS dla Linuksa żyje poza jądrem
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DRBD	Distributed Replicated Block Device: replikowane urządzenie blokowe w jądrze Linuksa, w którym każdy zapis trafia na dwa węzły
ELRepo	Enterprise Linux Repository: repozytorium społeczności z dodatkowymi jądrami i sterownikami dla RHEL i jego pochodnych, na przykład modułami DRBD

Skrót	Co znaczy
etcd	etc distributed: rozproszony magazyn par klucz-wartość, w którym Kubernetes trzyma cały stan klastra, a zmiany uzgadnia protokołem Raft
GPL	General Public License: otwarta licencja oprogramowania, na której opierają się MySQL Community i MariaDB
HA	High Availability: wysoka dostępność, czyli działanie mimo awarii jednego elementu
NFS	Network File System: udostępnianie katalogów po sieci w świecie Linuksa
OSD	Object Storage Daemon: demon Ceph obsługujący jeden dysk z danymi
RBD	RADOS Block Device: urządzenie blokowe udostępniane przez Ceph
RPO	Recovery Point Objective: ile danych wolno stracić przy awarii
RTO	Recovery Time Objective: jak długo wolno nie działać przy awarii
S2D	Storage Spaces Direct: programowa macierz Windows Server z dysków wielu serwerów
SPOF	Single Point of Failure: element, którego awaria zatrzymuje całą usługę, bo nie ma dla niego zamiennika
SQLite	mała baza danych zapisana w jednym pliku; Home Assistant trzyma w niej historię w pliku home-assistant_v2.db
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
VLAN	Virtual LAN: wydzielona logicznie sieć na tym samym sprzęcie
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych
XFS	system plików stworzony przez Silicon Graphics, domyślny w Rocky Linux i RHEL
ZFS	system plików z migawkami i replikacją przyrostową zbudowany w jednym narzędziu z menedżerem wolumenów (dawniej Zettabyte File System)

Fiszka

Komendy tylko odczytują stan i wykonujemy je z CLI bash z uprawnieniami root (sudo). Techniki z tej strony poza rsync nie działają dziś w Zielonej, więc przykłady podajemy po to, żeby rozpoznać stan systemu, gdy ktoś je wdroży albo gdy trafimy na taki system w innej firmie. Nazwy `NazwaZasobu`, `NazwaPuli` i `NazwaWoluminu` zastępujemy nazwami z danego systemu.

□□ TYLKO ODCZYT

Sprawdzenie stanu replikacji DRBD: drbdadm status

```
sudo drbdadm status
```

Pokazuje rolę każdego węzła (Primary albo Secondary) i stan dysku. Zdrowa replikacja to `UpToDate` po obu stronach, a `Inconsistent` oznacza trwającą synchronizację, `Outdated` albo `StandAlone` zasługuje na zatrzymanie i przyjrzenie się.

Przykład z hipotetycznego zasobu `dane` między `docker01` i `docker02`:

```
dane role:Primary
  disk:UpToDate
  docker02 role:Secondary
    peer-disk:UpToDate
```

DRBD = Distributed Replicated Block Device: replikowane urządzenie blokowe w jądrze Linuksa, w którym każdy zapis trafia na dwa węzły · **drbdadm** = administracja DRBD: pokazuje i zmienia stan zasobów replikacji

ZMIENIA SYSTEM

Sprawdzenie migawek ZFS i ich wieku: zfs list -t snapshot

```
sudo zfs list -t snapshot -o name,used,creation -r NazwaPuli/docker
```

Pokazuje migawki zbioru razem z datą utworzenia. Brak świeżej migawki oznacza, że replikacja nie ma z czego wysłać przyrostu, a po stronie odbiorcy ta sama komenda pokazuje, co dotarło.

ZFS = system plików z migawkami i replikacją przyrostową zbudowany w jednym narzędziu z menedżerem wolumenów (dawniej Zettabyte File System) · **zfs** = zarządzanie zbiorami danych ZFS: migawki, wysyłka (send) i odbiór (receive) replikacji

TYLKO ODCZYT

Sprawdzenie rozjazdu w GlusterFS: gluster volume heal info split-brain

```
sudo gluster volume heal NazwaWoluminu info split-brain
```

Wypisuje pliki, które na kopiach mają sprzeczną zawartość, i liczbę wpisów dla każdego brika. Zero wpisów to stan prawidłowy.

Przykład:

```
Brick docker01:/gluster/brick1
Status: Connected
Number of entries in split-brain: 0
```

gluster = administracja GlusterFS: woluminy, brik-i i leczenie rozjazdów (heal)

TYLKO ODCZYT

Sprawdzenie stanu klastra Ceph: ceph -s

```
sudo ceph -s
```

Pokazuje ogólny stan (`HEALTH_OK`, `HEALTH_WARN`, `HEALTH_ERR`), liczbę monitorów w kworum, liczbę OSD i stan grup umieszczania (PG). Szukamy `quorum` z większością monitorów i `active+clean` przy wszystkich PG.

OSD = Object Storage Daemon: demon Ceph obsługujący jeden dysk z danymi ·**ceph** = administracja klastrem Ceph, na przykład ceph -s pokazuje jego stan

TYLKO ODCZYT

Sprawdzenie wieku danych na rezerwie rsync: tail /var/log/ft-ha.log

```
sudo tail -n 5 /var/log/ft-ha.log
```

W Zielonej to jedyna komenda z tej listy, która opisuje działającą replikację: skrypt ze strony Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie wpisuje tam każdą synchronizację. Brak wpisu z ostatnich 15 minut oznacza, że timer nie działa albo host nie trzyma adresu wirtualnego.

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

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 14:00. Ola rysuje na tablicy dwa prostokąty, `docker01` i `docker02`, i strzałkę między nimi z podpisem „replikacja”. Tomek pyta, co się stanie, gdy ktoś na `docker01` wykona `rm -rf /docker/apps/gitea/data`. Ola kreśli drugi krzyżyk przy `docker02` i mówi, że to samo, tylko

> po kilku sekundach.

Replikacja kopiuje także pomyłki, więc nie zastępuje kopii zapasowej, tylko ją uzupełnia. Gdy Ola dopisuje na tablicy strzałkę „restic na bkp01”, zostaje jeszcze pytanie, w jaki sposób zapis ma trafiać na drugi host. Na to odpowiada ta strona.

Jak to działa

Rozwiń: jak to działa pod spodem

Trzy pytania przed wyborem

Technikę wybieramy po odpowiedzi na trzy pytania. Po pierwsze, co kopiujemy: pliki, bloki dysku czy rekordy w bazie. Po drugie, kiedy zapis uznajemy za zakończony: po zapisie lokalnym czy po potwierdzeniu z drugiego węzła. Po trzecie, kto rozstrzyga, który węzeł ma rację, gdy sieć między nimi zniknie.

> **Pojęcie: replikacja synchroniczna i asynchroniczna.** W synchronicznej aplikacja dostaje potwierdzenie zapisu dopiero po tym, jak dane trafiły na oba węzły, więc po awarii nie tracimy nic, ale każdy zapis jest wolniejszy o opóźnienie sieci. W asynchronicznej potwierdzenie przychodzi po zapisie lokalnym, a kopia dociera do drugiego węzła chwilę później, więc po awarii tracimy to, co nie zdążyło dotrzeć.

> **Pojęcie: replikacja a kopia zapasowa.** Replika odzwierciedla bieżący stan źródła, więc razem z danymi przenosi skasowanie, ransomware i błąd logiczny. Kopia zapasowa trzyma stan z przeszłości, do którego można wrócić.

Zestawienie technik

Technika	Na jakim poziomie	Minimum	RPO	Koszt i złożoność	Główna pułapka

rsync z timerem	pliki	dwa hosty	okres timera plus czas kopiowania	niski, niska	`--delete` powiela skasowanie, baza w trakcie kopiowania jest niespójna
ZFS send/receive	migawki zbioru	dwa hosty, dyski pod ZFS	częstość migawek (minuty)	średni, średnia	moduł jądra spoza dystrybucji, `receive -F` kasuje to, czego nie ma w nadawcy
DRBD	bloki urządzenia	dwa węzły, trzeci głos dla kworum	zero (protokół C) albo sekundy (A)	średni, wysoka	split brain, system plików tylko na jednym węźle naraz, pakiety spoza dystrybucji
GlusterFS	pliki w rozproszonym systemie	trzy węzły albo dwa i arbiter	zero (replica 3)	wysoki, wysoka	projekt wygasa, rozjazd kopii przy replica 2
Ceph	obiekty, bloki (RBD), pliki (CephFS)	trzy węzły, sieć 10 Gb	zero	bardzo wysoki, bardzo wysoka	wymagania sprzętowe, brak Rocky 9 w tabelach wsparcia Tentacle
Replikacja w aplikacji (bazy)	rekordy lub strumień logu	dwa węzły i zasady przełączenia	sekundy lub zero	średni, średnia	awans repliki bez kworum albo arbitra grozi dwoma serwerami głównymi

rsync z timerem systemd

Rsync porównuje pliki na obu hostach i kopiuje różnice. W Zielonej jest podstawą ciepłej rezerwy z poprzedniej strony, a jego atutem jest to, że znamy go wszyscy i nie wymaga niczego poza SSH. Ceną jest rozmiar okna: dane na `docker02` są tak stare jak ostatnia synchronizacja, a baza zapisująca pliki w trakcie kopii daje niespójny obraz.

Dlatego aplikacje z bazą w plikach (SQLite, katalog danych PostgreSQL w kontenerze) kopiujemy w trybie z zatrzymaniem albo bazę wyprowadzamy z kontenera. Pełny przykład z kontrolą kierunku i zabezpieczeniem przed pustym `/docker` jest na stronie Przenoszenie

kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie.

ZFS send/receive

> **Pojęcie: migawka (snapshot).** Niezmienny obraz zbioru danych z jednej chwili. W ZFS powstaje w ułamku sekundy i na początku nie zajmuje miejsca, bo zapamiętuje tylko późniejsze zmiany.

Replikacja ZFS wysyła migawkę przez `zfs send` i odbiera ją przez `zfs receive`. Pierwsza wysyłka to cały zbiór, kolejne mogą być przyrostowe: opcja `-i` wysyła różnicę między dwiema migawkami, a `-I` także wszystkie pośrednie [1]. Odbiorca dostaje spójny obraz z jednej chwili, więc baza zapisana w tej samej migawce jest odtwarzalna tak samo jak po nagłym wyłączeniu zasilania.

```
sudo zfs snapshot NazwaPuli/docker@ha-$(date +%F-%H%M)
sudo zfs send -i NazwaPuli/docker@PoprzedniaMigawka NazwaPuli/docker@NowaMigawka \
  | ssh nazwa_usera@docker02 sudo zfs receive -s -u NazwaPuli/docker
```

Opcji `-F` po stronie odbiorcy świadomie nie używamy: wycofuje ona zbiór odbiorcy do jego ostatniej migawki przed odbiorem, a przy strumieniu replikacyjnym usuwa to, czego nie ma w nadawcy [1]. Opcja `-s` pozwala wznowić przerwany transfer z tokenu, a `-u` nie montuje odebranego zbioru, żeby pozostał nieaktywny do chwili przełączenia [1].

Cena ZFS jest organizacyjna. Dyski rezerwy i aktywnego hosta muszą być pod ZFS, więc to zmiana układu `vgDocker/lvDocker` na pulę ZFS. Moduł jądra pochodzi spoza dystrybucji (repozytorium `zfs-release-3-0` obejmuje EL8 do EL10) i po aktualizacji jądra musimy sprawdzić, czy się zbudował [2]. Licencja CDDL nie jest zgodna z GPL, dlatego ZFS na Linuksie żyje poza jądrem, co sprawdzamy w dokumentacji OpenZFS przed wdrożeniem [2].

Wariant ZFS dla Zielonej opisujemy jako szkic na poprzedniej stronie.

DRBD

> **Pojęcie: DRBD (Distributed Replicated Block Device).** Replikowane urządzenie blokowe w jądrze Linuksa. Każdy zapis na urządzenie na jednym węźle trafia też do drugiego, więc oba mają te same bloki, a system plików montujemy tylko na jednym, aktywnym (Primary).

DRBD ma trzy protokoły. W protokole A zapis jest potwierdzony po zapisie lokalnym i wysłaniu do bufora TCP, w B po dotarciu pakietu do partnera, a w C po zapisie na obu dyskach, czyli synchronicznie [3]. Dla danych, których strata boli, wybieramy C i płacimy opóźnieniem sieci przy każdym zapisie.

Dwa fakty decydują o przydatności DRBD w Zielonej. Pierwszy to kworum: dwa węzły nie wystarczą, więc DRBD 9 pozwala dodać trzeci, bezdyskowy węzeł (tiebreaker), który tylko głosuje [3]. Drugi to tryb dual-primary: DRBD samo nie zamienia zwykłego XFS albo ext4 w system aktywny-aktywny, bo wymaga to klastrowego systemu plików z rozproszonym menedżerem blokad [3].

Wersja 9.3.4 pochodzi z 15 września 2026, a pakiety wstępnie zbudowane dostajemy w subskrypcji LINBIT; ścieżkę bez subskrypcji (ELRepo albo kompilacja) trzeba sprawdzić, bo w tej sesji nie zdołaliśmy jej potwierdzić [3]. Moduł jądra jest poza dystrybucją, tak jak przy ZFS, więc aktualizacje jądra wymagają planu. Replikacja bloków przenosi także uszkodzony system plików, a przełączenie to dwie czynności więcej niż przy rsync: promocja (Secondary na Primary) i montowanie.

GlusterFS

> **Pojęcie: GlusterFS.** Rozproszony system plików, w którym pliki są kopiowane na kilka serwerów (brików) i widoczne jako jeden wolumin. Zwykle używa się replica 3 albo replica 2 z arbitrem, który trzyma tylko metadane.

Dla nowych wdrożeń go nie rekomendujemy. Red Hat zakończył wsparcie dla Gluster Storage 31 grudnia 2024 roku, liczba zmian w repozytorium spadła z 78 w 2023 roku do 31 w 2024 roku, a CentOS Storage SIG nie buduje pakietów dla Stream 10 [4]. Ostatnie wydanie, które znaleźliśmy, to 11.2, a jego rok trzeba sprawdzić. Replica 2 bez arbitra jest podatna na split brain, a przy zaostrzeniu kworum traci dostępność [4].

Ceph

> **Pojęcie: Ceph.** Rozproszony magazyn obiektów, bloków (RBD) i plików (CephFS), który sam wykrywa awarie i odtwarza brakujące kopie. Składa się z monitorów, które trzymają mapę klastra i głosują, oraz z demonów OSD, po jednym na każdy dysk.

Ceph jest najlepszy z opisanych technik i najdroższy w utrzymaniu. Potrzebuje co najmniej trzech węzłów i trzech monitorów, a dokumentacja zaleca sieć 10 Gb i osobny dysk na każdy

OSD [5]. Gałąź Tentacle (20.2.4) ma wsparcie do 1 czerwca 2027, Squid (19.2.6) do 31 października 2026, czyli za miesiąc, a tabele systemów dla Tentacle wymieniają CentOS 9 i Rocky 10, ale nie Rocky 9 [5]. Dla firmy z dwoma hostami Hyper-V, na których trzy maszyny wirtualne i tak tracą kworum przy awarii hosta, Ceph jest więc przerostem.

Współdzielony NFS

Dwa hosty montujące jeden udział NFS wyglądają jak najprostsza gorąca rezerwa, ale serwer NFS sam staje się punktem awarii (SPOF). Bazy i SQLite na NFS są ryzykowne ze względu na blokady plików. Używamy NFS do danych, które i tak są tylko do odczytu albo do kopii, a nie jako podkładu pod aplikacje z zapisem.

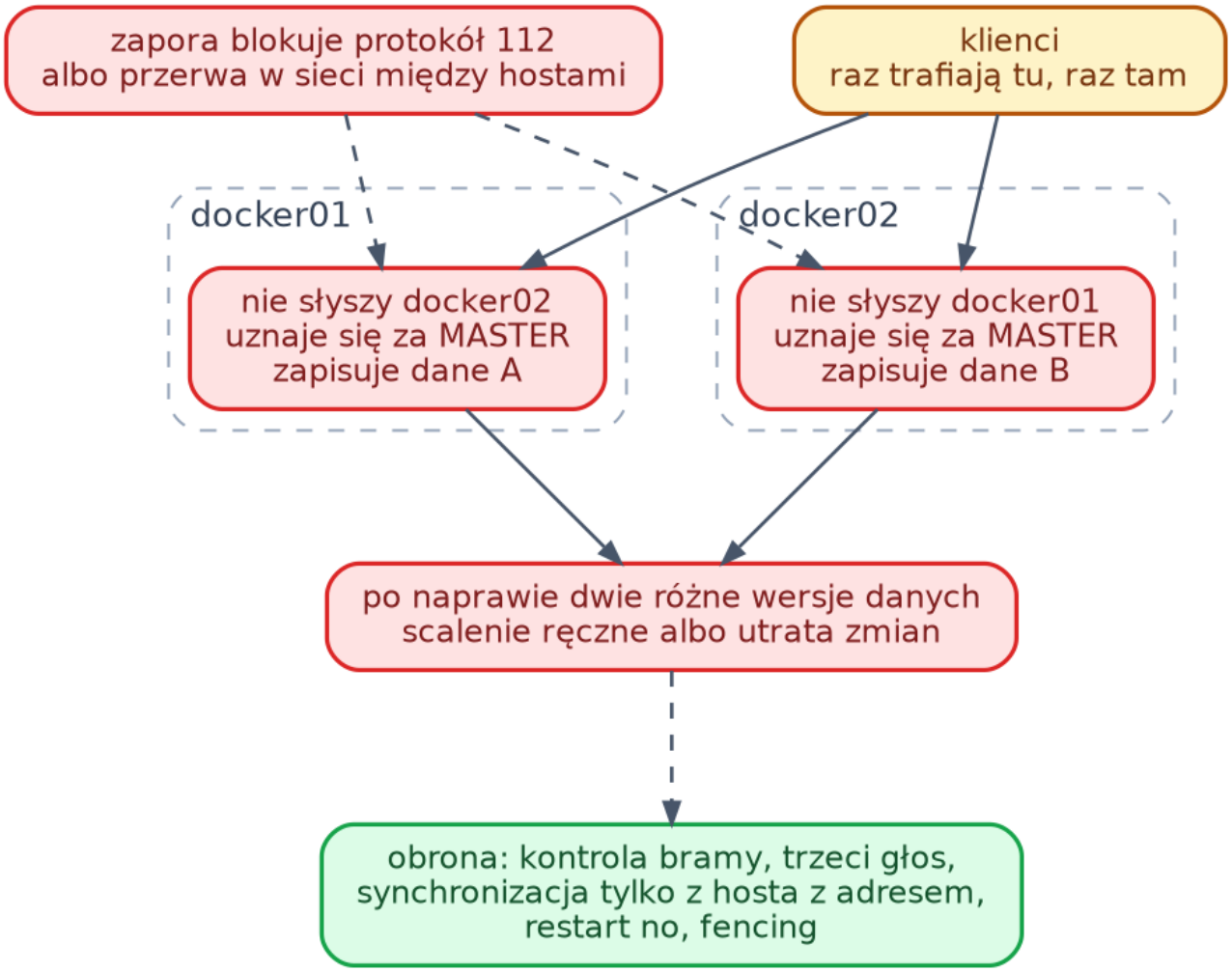
Replikacja w samej aplikacji

Najlepszy mechanizm gorącej rezerwy dla danych strukturalnych to ten wbudowany w silnik. PostgreSQL replikuje strumień dziennika WAL do repliki (piszemy o tym w Replikacja strumieniowa), baza MongoDB ma zestawy replik, Cassandra trzyma kilka kopii (u nas współczynnik 3 na `cass01` do `cass03`), a Active Directory replikuje się natywnie między `dc01` i `dc02`. Silnik wie, które zapisy są potwierdzone, więc nie ma problemu niespójnego obrazu pliku.

Dla Zielonej to oznacza, że gorąca rezerwa danych jest przede wszystkim sprawą baz, a nie systemu plików. Gitea trzyma repozytoria w plikach i metadane w PostgreSQL na `pg01`, więc replika `pg01` załatwia drugą połowę.

Pliki idą rsync-iem. Awans repliki PostgreSQL (`pg_ctl promote`) wykonuje ręcznie człowiek albo narzędzie z zewnątrz, a różnicę między planowanym przełączeniem i awaryjnym opisuje strona o replikacji strumieniowej [6]. Dla automatycznego przełączenia potrzebujemy kworum albo arbitra, o co pytamy w przebiegu z macierzą HA.

Split brain w każdej technice



Rysunek 1. Split brain w dwóch hostach. Po zerwaniu łączności oba uznają się za aktywne i przyjmują zapisy, a po jej przywróceniu każdy ma zmiany, których nie ma drugi. Ratuje trzeci głos albo ręczna decyzja, który zbiór danych zostaje. (kliknięcie otwiera pełny rozmiar)

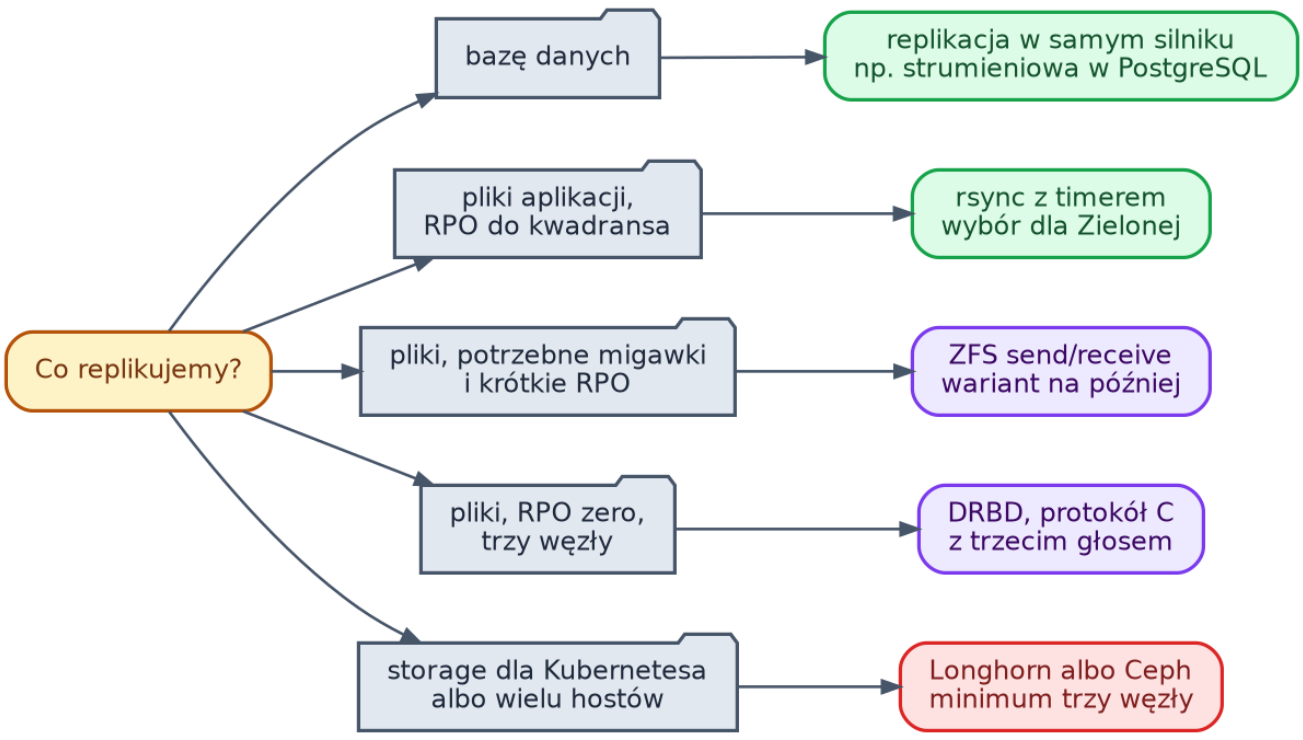
Każda technika ma swój sposób na rozjazd i swój koszt jego naprawy. Tabela pokazuje, co dzieje się po zerwaniu sieci i jak zwykle wracamy do jednego zbioru.

Technika	Co się psuje po zerwaniu sieci	Jak wracamy do porządku
rsync z VIP	oba hosty trzymają adres wirtualny i oba zapisują	decyzja człowieka, którą stronę zostawiamy, potem synchronizacja z wybranej strony
ZFS	zbiór po stronie odbiorcy ma migawki, których nie ma nadawca	rozstrzygamy, którą historię zostawiamy, a `zfs receive -F` wycofuje zbiór odbiorcy [1]

Technika	Co się psuje po zerwaniu sieci	Jak wracamy do porządku
DRBD	dwa węzły Primary z rozbieżnymi blokami	polityki automatycznej naprawy albo ręczny wybór ofiary, której zmiany tracimy [3]
GlusterFS	pliki z różną zawartością na kopiach	`gluster volume heal` z polityką wyboru źródła [4]
Ceph	mniejszość monitorów traci kworum i przestaje działać	po powrocie łączności mniejszość dołącza do większości, bo przy braku kworum nie przyjmowała zapisów [5]
Baza natywna	dwa węzły uznają się za główne	arbiter, kworum, ręczne `pg_rewind` albo odbudowa repliki

Najlepszą obroną jest trzeci głos albo bezpiecznik w skryptach. W Zielonej drugi host zabezpieczamy skryptem synchronizacji, który nie nadpisuje danych, gdy partner też trzyma adres wirtualny, a decyzję o powrocie starego hosta podejmuje człowiek.

Wybór dla Zielonej



Rysunek 2. Drzewo decyzji: od typu danych (baza czy pliki), przez liczbę węzłów i wymagane RPO, do techniki. Dla Zielonej kończy się na bazach z replikacją natywną i na plikach z rsync albo ZFS. (kliknięcie otwiera pełny rozmiar)

Rekomendacja dla Zielonej składa się z czterech decyzji i jednej zasady. Zasada brzmi: replikacja zawsze obok kopii, nigdy zamiast niej.

1. Bazy replikujemy natywnie. Dla `pg01` oznacza to repliki strumieniowe i sprawdzenie, czy potrzebujemy automatycznego przełączenia. Baza w kontenerze na `docker01` (NetBox) trafia na `pg01` albo zostaje na zimnej rezerwie.
2. Pliki aplikacji z `docker01` kopiujemy rsync-iem z timerem co 15 minut na `docker02` i dodatkowo restic o 02:30 na `bkp01`. Daje to RPO rzędu kwadransa i RTO rzędu kilku minut.
3. ZFS send/receive jest wariantem, który wybieramy, jeśli RPO ma spaść do kilku minut albo jeśli zależy nam na spójnych migawkach. Wymaga przebudowy układu dysków i modułu jądra, więc nie wdrażamy go na początek.
4. DRBD bierzemy pod uwagę tylko z trzecim głosem, na przykład tiebreakerem na `bkp01`, co wymaga sprawdzenia możliwości instalacji tam modułu. GlusterFS odrzucamy, Ceph zostawiamy na sytuację z co najmniej trzema fizycznymi hostami.

Uzasadnienie ma cztery części.

Zielona ma dwa hosty Hyper-V i dwóch administratorów, a każda technika z trzema węzłami zaczyna od tego, że kworum nie przeżyje awarii hosta z dwoma węzłami. S2D pod spodem już wykonuje replikację dysków (dwie kopie), więc replikacja bloków nad nim dubluje ochronę, a nie dodaje nowej. Zakładamy, że ilość danych aplikacji na `docker01` jest niewielka (sprawdzamy to niżej), a RPO rzędu kwadransa dla Gitea, Vaultwardena i poczty jest do zaakceptowania, co potwierdzamy z właścicielami usług. Ostatnia część to prostota: rsync i restic znamy, a przy stresie w piątek o 20:00 działa to, co znamy.

Wielkość danych sprawdzamy komendą `sudo du -sh /docker/apps/*` na `docker01`, bo od niej zależy czas synchronizacji. Jeśli któraś aplikacja urośnie tak, że synchronizacja nie mieści się w kwadransie, przesuwamy ją na ZFS albo wynosimy dane na `pg01`.

Kiedy zmieniamy rekomendację

Rekomendację zmieniamy, gdy zmieni się któryś z założeń. RPO poniżej minuty dla plików wskazuje na DRBD z trzecim głosem albo ZFS z częstymi migawkami. Trzeci fizyczny host pozwala na Ceph albo k3s z Longhornem, czemu poświęcimy osobne strony. Potrzeba aktywnego-aktywnego dla aplikacji przerzuca sprawę do samej aplikacji, bo żaden system plików tego za nią nie załatwi.

Częste błędy

Błąd	Skutek	Jak unikamy
Replikacja zamiast kopii zapasowej	skasowanie lub ransomware dociera do drugiego hosta	restic na `bkp01` zostaje na każdym poziomie
DRBD na dwóch węzłach bez trzeciego głosu	po zerwaniu sieci oba węzły mogą stać się Primary	tiebreaker, kworum i polityki, nie wybór „na oko”
Zwykły XFS albo ext4 zamontowany na obu węzłach DRBD	uszkodzenie systemu plików	system plików tylko na Primary, dual-primary tylko z klastrowym systemem plików
`zfs receive -F` jako domyślny nawyk	kasuje po stronie odbiorcy dane, których nie ma w nadawcy	pierwsza wysyłka bez `-F`, `-F` dopiero po decyzji i sprawdzeniu
Replikacja żywej bazy rsync-iem	niespójny obraz bazy po awarii	replikacja natywna albo zrzut, albo rsync z zatrzymaniem aplikacji
GlusterFS replica 2 bez arbitra	rozjazd kopii przy zerwaniu sieci	jeśli już działa, replica 3 albo arbiter, a w nowym wdrożeniu inna technika
Ceph z trzema węzłami na dwóch hostach Hyper-V	awaria hosta zabiera kworum monitorów	trzy fizyczne hosty albo inna technika
Brak alarmu na wiek danych rezerwy	po awarii okazuje się, że rezerwa ma dane sprzed tygodnia	Zabbix sprawdza ostatni wpis w `/var/log/ft-ha.log` albo stan timera

□□ Ćwiczenia

1. Dlaczego przy DRBD w protokole C po awarii nie tracimy danych, a przy protokole A możemy?

Odpowiedź

W protokole C potwierdzenie zapisu wraca do aplikacji dopiero po zapisie na dysku lokalnym i zdalnym, więc każdy potwierdzony zapis jest na obu węzłach. W protokole A potwierdzenie przychodzi po zapisie lokalnym i wysłaniu do bufora, więc przy awarii węzła główne dane, które nie zdążyły dotrzeć do partnera, giną. C płaci za to opóźnieniem sieci przy każdym zapisie.

1. Na `docker02` po awarii sieci okazuje się, że zbiór ZFS ma migawki, których nie ma `docker01`. Co sprawdzamy przed `zfs receive -F`?

Odpowiedź

Sprawdzamy, które migawki są nowsze i czy zawierają zapisy, które chcemy zachować, bo `-F` zniszczy po stronie odbiorcy to, czego nie ma w nadawcy. Jeśli prawdziwe zmiany użytkowników trafiły na `docker02`, kierunek replikacji odwracamy albo zachowujemy migawki jako klony przed

odbiorem.

1. Zielona ma dwa hosty Hyper-V. Dlaczego trzy maszyny wirtualne z Cepha albo etcd nie dają odporności na awarię hosta?

Odpowiedź

Trzy węzły na dwóch hostach oznaczają, że jeden host trzyma co najmniej dwa węzły. Awaria tego hosta zabiera większość głosów (dwa z trzech), więc klaster traci kworum i staje. Odporność na awarię hosta dają trzy węzły na trzech niezależnych hostach.

1. Dla której z aplikacji z `docker01` mamy najlepszą gorącą rezerwę danych, jeśli replikujemy tylko `pg01`?

Odpowiedź

Dla tych, których dane leżą wyłącznie w PostgreSQL na `pg01`. Gitea ma tam metadane, ale repozytoria leżą w plikach w `/docker/apps/gitea/data`, więc replika `pg01` załatwia tylko połowę. Pozostałą część chronimy rsync-iem i restic, a pełna gorąca rezerwa wymagałaby także replikacji plików.

Powiązane: Wysoka dostępność: usługa, która nie ma jednego serwera · Wysoka dostępność: pojęcia, liczenie dostępności i poziomy rezerwy · Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie · Replikacja: synchroniczna, asynchroniczna, logiczna, fizyczna · Replikacja strumieniowa · Kopia całego docker01: /docker/apps, /docker/compose i bazy w kontenerach

Źródła:

1. openzfs.github.io: zfs-send(8) i zfs-receive(8)
2. openzfs.github.io: Getting Started, RHEL-based distro; wydania na GitHubie
3. LINBIT: DRBD 9 User Guide, repozytorium DRBD i strona pobierania
4. docs.gluster.org: Arbiter volumes and quorum; Phoronix o wycofaniu GlusterFS z Fedory; Red Hat o zakończeniu wsparcia
5. docs.ceph.com: Releases, Add or remove monitors, OS recommendations, Hardware recommendations
6. Ta baza wiedzy: Replikacja strumieniowa, sekcje o promocji repliki, failover i split-brain