Równoważenie i adres wirtualny: keepalived i HAProxy na parze lb01 i lb02

Poziom: średnio

Dotyczy: keepalived 2.2 (pakiet EL9), HAProxy (pakiet EL9 z AppStream to gałąź 2.4, plik sprawdzony na 2.8.16), Traefik v3.5, MailCow, Rocky Linux 9, FortiGate fw01

Czas: ~120 min

Dwie maszyny `lb01` i `lb02` dostają wspólny adres `10.20.10.66`, a HAProxy na nich rozdziela ruch z `fw01` na `docker01` i `docker02`. Dzięki temu awaria jednego z hostów Dockera albo jednej maszyny równoważącej nie zmienia adresu, pod którym użytkownicy znajdują usługi. Strona rozwiązuje też kolizję portów 80 i 443 między Traefikiem a MailCow, która wyszła przy sprawdzeniu wcześniejszych stron.

ZAŁOŻENIA SCENARIUSZA

`lb01` i `lb02` stoją w VLAN 10, a pojedynczy `fw01` z jednym łączem (`wan1`, 203.0.113.10) kieruje porty webowe na adres 10.20.10.66, a pocztowe wprost na 10.20.10.46. Sama zapora i łącze zostają więc pojedynczym punktem awarii tej architektury; parę FortiGate opisuje Para FortiGate w klastrze: FGCP aktywny-pasywny.

Skróty użyte na tej stronie (29)

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
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami

Skrót	Co znaczy
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
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
HAProxy	High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
IETF	Internet Engineering Task Force: organizacja publikująca standardy internetowe jako dokumenty RFC
IMAP	Internet Message Access Protocol: protokół, którym klient pocztowy pobiera wiadomości z serwera, zostawiając je tam zamiast kasować
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
LTS	Long Term Support: wersja z długim wsparciem poprawkami
MAC	Media Access Control: unikalny adres fizyczny karty sieciowej, przypisany fabrycznie
POP3	starszy protokół pobierania poczty na dysk lokalny, w odróżnieniu od IMAP
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL
SMTP	Simple Mail Transfer Protocol: protokół wysyłania poczty
SNI	Server Name Indication: pole w pierwszym komunikacie TLS, w którym klient podaje nazwę serwera, z którym chce rozmawiać, co pozwala HAProxy wybrać backend bez rozszyfrowywania ruchu
SOGo	webmail z kalendarzem i kontaktami, który w MailCow jest dostępny w przeglądarce obok panelu administracyjnego
SPF	Sender Policy Framework: rekord DNS mówiący, które serwery mogą wysyłać pocztę domeny
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS
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)
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

Komendy wykonujemy z CLI bash z uprawnieniami root (sudo), na `lb01` albo `lb02`, chyba że przy komendzie napisano inaczej. Adresy z Zielonej: `lb01` to `10.20.10.64`, `lb02` to `10.20.10.65`, adres wirtualny równoważenia to `10.20.10.66`, interfejs sieciowy to `ens192`. Bloki z napisem „Przykład” pokazują kształt wyniku, a część z nich pochodzi z prób w piaskownicy (opisanych w sekcji „Test awarii”), nie z serwerowni Zielonej.

TYLKO ODCZYT

Sprawdzenie składni HAProxy: haproxy -c -f

```
sudo haproxy -c -f /etc/haproxy/haproxy.cfg
```

Sprawdza plik bez uruchamiania usługi. Robimy to przed każdym `reload`, bo błąd w pliku na aktywnej maszynie kończy się brakiem odpowiedzi na porcie 443.

Przykład:

```
Configuration file is valid
```

HAProxy = High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają · **haproxy** = serwer równoważenia obciążenia i proxy; z opcją -c -f sprawdza składnię pliku konfiguracji bez uruchamiania usługi

ZMIENIA SYSTEM

Przeładowanie HAProxy bez zrywania połączeń: systemctl reload haproxy

```
sudo systemctl reload haproxy
```

Pakiet z EL9 uruchamia HAProxy w trybie master-worker, więc po `reload` nowy proces przejmuje nowe połączenia, a stary kończy te, które już trwają. Restart (`restart`) zrywa wszystkie połączenia, więc używamy go tylko po aktualizacji pakietu.

HAProxy = High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają · **EL9** = Enterprise Linux 9: wspólna nazwa dystrybucji zgodnych z RHEL 9, takich jak Rocky Linux 9 · **systemctl** = system control: sterowanie usługami systemd

TYLKO ODCZYT

Odczyt stanu serwerów backendów: socat

```
echo "show stat" | sudo socat stdio /var/lib/haproxy/stats | cut -d, -f1,2,18 | grep -v "^#" | grep -v "FRONTEND\|BACKEND"
```

Pyta gniazdo administracyjne HAProxy o statystyki w formacie CSV i zostawia nazwę backendu, nazwę serwera i status (kolumny 1, 2 i 18). Wymaga pakietu `socat`. Status `UP` oznacza, że kontrola stanu przechodzi, a `DOWN`, że serwer nie odpowiada.

Przykład, gdy aktywny jest `docker01`:

```
be_traefik_http,docker01,UP
be_traefik_http,docker02,DOWN
be_traefik_https,docker01,UP
be_traefik_https,docker02,DOWN
be_mailcow_http,docker01,UP
be_mailcow_http,docker02,DOWN
be_mailcow_https,docker01,UP
be_mailcow_https,docker02,DOWN
be_gitea_ssh,docker01,UP
be_gitea_ssh,docker02,DOWN
```

Host rezerwowy jest `DOWN` z założenia: jego stosy są wyłączone, więc porty są zamknięte.

HAProxy = High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają · **CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami · **echo** = od angielskiego echo: wypisuje podany tekst na standardowe wyjście, często łączone z przekierowaniem do pliku

ZMIENIA SYSTEM

Wyłączenie serwera z rotacji na czas prac: set server state maint

```
echo "set server NazwaBackendu/NazwaSerwera state maint" | sudo socat stdio /var/lib/haproxy/stats
```

> (gdzie `NazwaBackendu/NazwaSerwera` to na przykład `be_traefik_http/docker01`; polecenie działa na jednym backendzie, więc dla pełnego wyłączenia hosta powtarzamy je dla każdego backendu)

Serwer przechodzi w stan `MAINT` i nie dostaje nowych połączeń. Zmiana nie przetrwa przeładowania ani restartu HAProxy, więc po pracach i tak sprawdzamy stan.

HAProxy = High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają · **echo** = od angielskiego echo: wypisuje podany tekst na standardowe wyjście, często łączone z przekierowaniem do pliku

ZMIENIA SYSTEM

Powrót serwera do rotacji: set server state ready

```
echo "set server NazwaBackendu/NazwaSerwera state ready" | sudo socat stdio /var/lib/haproxy/stats
```

Serwer wraca do kontroli stanu i po kilku udanych próbach (`rise`) znów jest `UP`.

echo = od angielskiego echo: wypisuje podany tekst na standardowe wyjście, często łączone z przekierowaniem do pliku

TYLKO ODCZYT

Sprawdzenie składni keepalived: keepalived --config-test

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

Sprawdza `/etc/keepalived/keepalived.conf` bez uruchamiania usługi. Skrót `-t` robi to samo. Brak komunikatów i kod wyjścia 0 oznaczają, że plik jest poprawny.

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

TYLKO ODCZYT

Sprawdzenie, która maszyna trzyma adres wirtualny: ip -br -4 addr show

```
ip -br -4 addr show dev ens192
```

Aktywna maszyna ma dwa adresy: własny i wirtualny. Rezerwowa ma tylko własny.

Przykład na aktywnym `lb01`:

```
ens192           UP             10.20.10.64/24 10.20.10.66/24
```

ip = narzędzie do adresów, tras i interfejsów sieciowych

TYLKO ODCZYT

Podgląd zmian roli keepalived: journalctl -u keepalived

```
sudo journalctl -u keepalived --since "1 hour ago" | grep -i "state"
```

Szukamy wpisów `Entering MASTER STATE`, `Entering BACKUP STATE` i `Entering FAULT STATE` z nazwą instancji `VI_LB`.

journalctl = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Podgląd ogłoszeń VRRP w sieci: tcpdump -ni ens192 proto 112

```
sudo tcpdump -ni ens192 proto 112
```

Pokazuje ogłoszenia VRRP (protokół IP 112) wysyłane przez aktywną maszynę, w naszym układzie unicastem do partnera. Brak ogłoszeń przez kilka sekund oznacza, że aktywna maszyna milczy albo zapora je zatrzymuje.

VRRP = Virtual Router Redundancy Protocol: protokół, w którym kilka hostów uzgadnia, który z nich trzyma wspólny adres IP (opisany w RFC 5798) · **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **tcpdump** = podsłuch ruchu sieciowego na interfejsie

TYLKO ODCZYT

Próba trasy przez adres wirtualny: curl --resolve

```
curl -ks -o /dev/null -w '%{http_code}\n' --resolve
NazwaUslugi.zielona.example:443:10.20.10.66 https://NazwaUslugi.zielona.example/
```

Wymusza rozwiązanie nazwy na adres `10.20.10.66`, więc sprawdzamy całą drogę: HAProxy, backend i aplikację, nie zmieniając DNS.

Przykład dla `git.zielona.example`:

```
200
```

HAProxy = High Availability Proxy: program, który rozdziela połączenia między kilka serwerów i omija te, które nie odpowiadają · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:10. Tomek przestawił w DNS `git.zielona.example` na adres wirtualny `10.20.10.66`, bo przeglądarka działa, a Ola pisze, że `git push` przez SSH przestał działać. HAProxy słuchał wtedy tylko na portach 80 i 443, a Gitea ma SSH na porcie 2222, którego nikt nie dopisał do konfiguracji. Poprawka to osiem linii w `haproxy.cfg` i `reload`, a Tomek dopisuje do listy kontrolnej pytanie: jakie jeszcze porty aplikacje wystawiają poza HTTP.

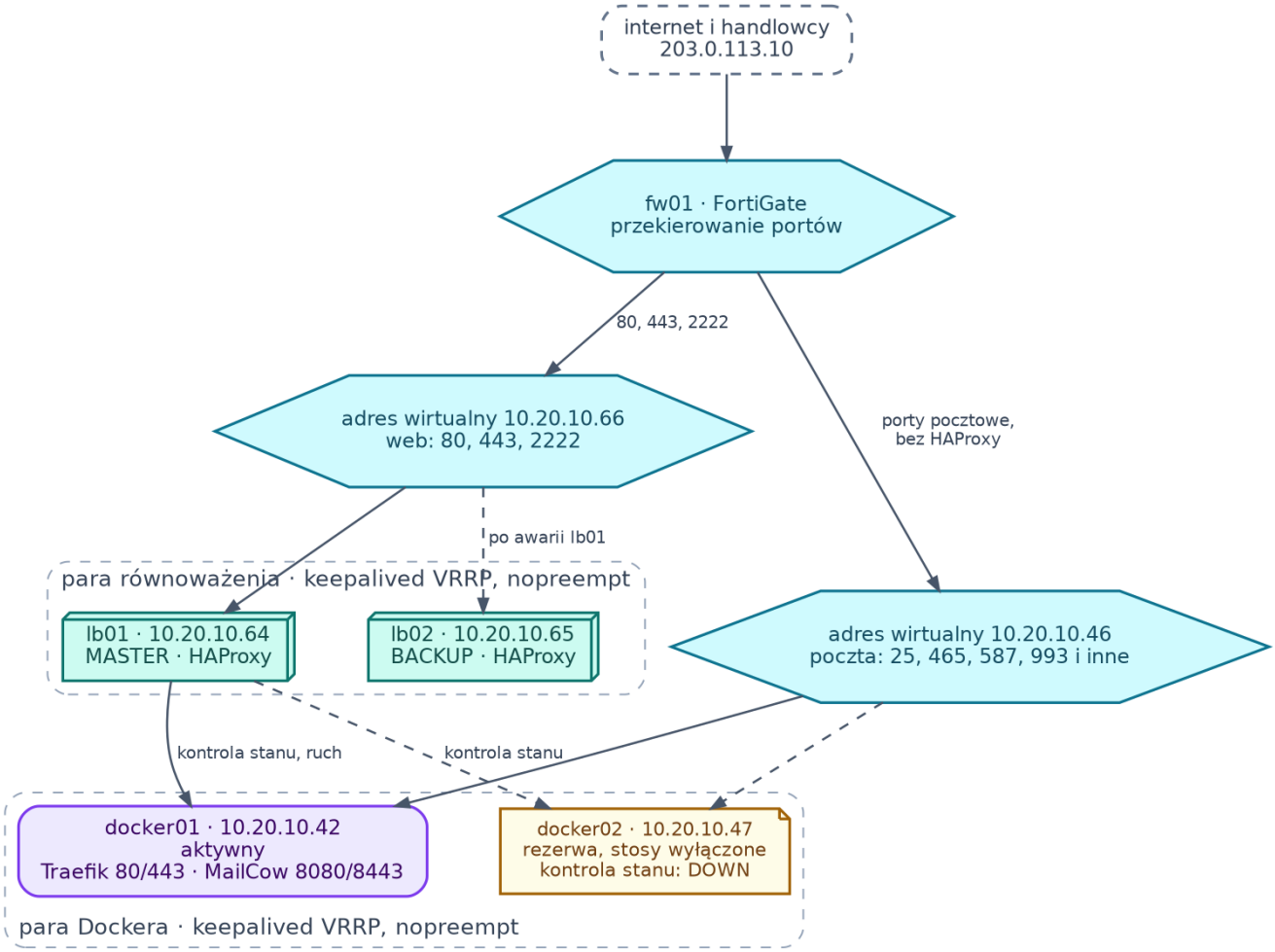
Jak to działa

Rozwiń: jak to działa pod spodem

Dwa adresy wirtualne, dwie pary maszyn

> **Pojęcie: równoważenie obciążenia (load balancing).** Rozdzielanie przychodzących połączeń między kilka serwerów, które robią to samo, z pominięciem tych, które nie odpowiadają. Narzędzie, które to robi, nazywamy równoważnikiem albo po angielsku load balancerem (LB).

W Zielonej mamy teraz dwie pary maszyn, które chroni ten sam mechanizm VRRP i keepalived opisany na stronie Przenoszenie kontenerów z docker01 na docker02: zimna i ciepła rezerwa, przełączenie. Para `docker01` i `docker02` trzyma adres `10.20.10.46` i przełącza stosy. Nowa para `lb01` i `lb02` trzyma adres `10.20.10.66` i przekazuje ruch do tego z hostów Dockera, który akurat odpowiada.



Rysunek 1. Architektura z parą równoważącą: fw01 kieruje porty webowe na adres wirtualny 10.20.10.66 (lb01 i lb02 z HAProxy), a porty pocztowe prosto na adres 10.20.10.46 pary Dockera. HAProxy sprawdza stan obu hostów i kieruje ruch do tego, który odpowiada. (kliknięcie otwiera pełny rozmiar)

Po co dokładać drugą parę, skoro `docker01` i `docker02` już mają adres wirtualny? Są trzy powody. HAProxy sprawdza, czy aplikacja naprawdę odpowiada, a nie tylko czy host żyje, i rozdziela ruch po nazwie usługi, więc MailCow i Traefik mogą stać na tym samym hoście. Jest też miejscem, w którym później dołożymy kolejne backendy (Swarm, k3s, drugą instancję aplikacji), nie ruszając reguł na `fw01`.

Cena to dwie dodatkowe maszyny wirtualne i jeszcze jedno miejsce do pilnowania. Para równoważąca jest sama warstwą, która może się zepsuć, dlatego ma własny adres wirtualny i własny test awarii.

Tryb tcp i tryb http

> **Pojęcie: tryb tcp i tryb http w HAProxy.** W trybie tcp HAProxy widzi tylko strumień bajtów i adresy, więc przekazuje go dalej bez zaglądania do środka. W trybie http czyta nagłówki żądania (na przykład `Host`), może je zmieniać i kierować ruch po adresie, ścieżce albo ciasteczku.

Port 80 obsługujemy w trybie http, bo nagłówek `Host` mówi, o którą usługę chodzi. Port 443 obsługujemy w trybie tcp, bo HAProxy nie deszyfruje ruchu i nie ma certyfikatów. Nazwę usługi odczytuje z początku uzgadniania TLS: pole SNI (Server Name Indication) klient wysyła jawnym tekstem, zanim zaczną się szyfrowane dane. Port 2222 (SSH do Gitea) to także tryb tcp, bo SSH nie ma nic wspólnego z HTTP.

Kto kończy TLS

To pytanie wraca przy każdym równoważniku i ma trzy odpowiedzi, a my wybraliśmy drugą.

Wariant	Co robi HAProxy	Gdzie leżą certyfikaty	Adres klienta w aplikacji	Wady
A. HAProxy kończy TLS	deszyfruje i przekazuje HTTP	na `lb01` i `lb02` (po dwie kopie, odnowienia ACME trzeba tam synchronizować)	w nagłówku `X-Forwarded-For`	druga kopia kluczy prywatnych, Traefik traci rolę
B. HAProxy przepuszcza TLS po SNI (nasz wybór)	kieruje strumień po nazwie z SNI	tam, gdzie dziś: w Traefiku i w MailCow	w Traefiku przez PROXY protocol, w MailCow bez zmian (patrz niżej)	brak kierowania po ścieżce, adres klienta w MailCow to adres `lb`
C. Bez równoważnika	nic	bez zmian	bez zmian	zostaje jeden punkt wejścia na hoście Dockera

Wybieramy wariant B, bo certyfikaty `CA-Zielona` i ACME Let's Encrypt są już rozwiązane w Traefiku (strona Reverse proxy w Dockerze: Traefik przed wszystkimi aplikacjami na docker01), a MailCow ma własny kontener `acme-mailcow`. Równoważnik, który niczego nie deszyfruje, nie jest też celem na klucze prywatne.

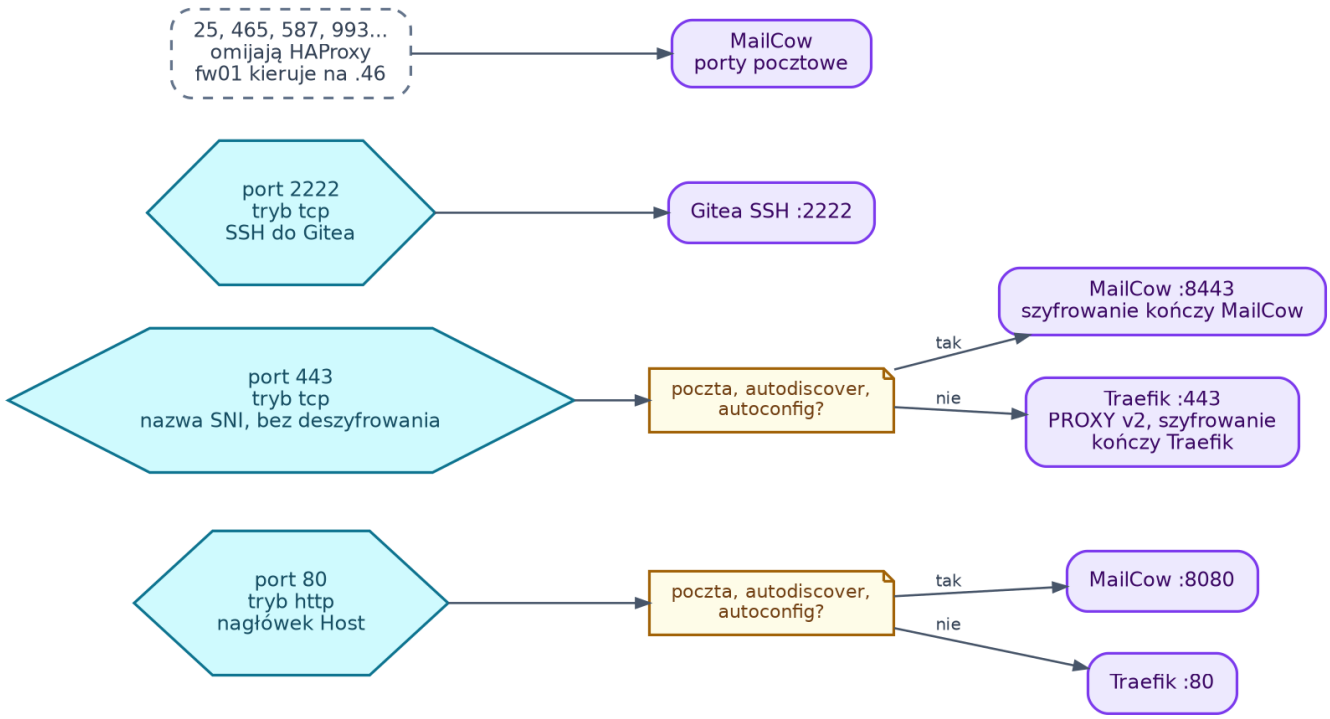
> “ Po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona nowe certyfikaty wystawia `Zielona-CA01` na `ca01`, więc wszędzie, gdzie dziś ufamy tylko `CA-Zielona`, dokładamy zaufanie do `Zielona-Root-CA`.

> **Pojęcie: PROXY protocol.** Krótki nagłówek, który równoważnik wysyła na początku połączenia TCP, żeby serwer za nim poznał prawdziwy adres klienta. Serwer musi być na to przygotowany: bez zgody po jego stronie nagłówek zostałby uznany za zepsute dane.

Traefik obsługuje PROXY protocol na punktach wejścia i pozwala wskazać adresy, którym ufa [6]. Dlatego do Traefika HAProxy dopisuje nagłówek (`send-proxy-v2`), a Traefik przyjmuje go tylko od `lb01` i `lb02`. MailCow tego nie dostaje, bo nie potwierdziliśmy w dokumentacji obsługi PROXY protocol w jego `nginx-mailcow`.

> ⚠ Skutek dla MailCow: w dzienniku panelu i w filtrze `netfilter-mailcow` źródłem połączeń webowych będą adresy `10.20.10.64` i `10.20.10.65`. Ochrona przed zgadywaniem haseł w panelu WWW i w SOGo nie rozróżni wtedy użytkowników. Protokoły pocztowe (SMTP, IMAP, POP3) idą poza HAProxy, więc ich ochrona działa jak dotąd. Jeśli ta różnica jest dla autora nie do przyjęcia, wybieramy wariant A dla nazw MailCow (przykład zawiera dokumentacja MailCow w rozdziale o reverse proxy, a plik z certyfikatem trzeba wtedy utrzymywać na obu `lb`).

Plan portów: co idzie przez HAProxy, a co nie



Rysunek 2. Plan portów: 80 przez HAProxy w trybie http po nagłówku Host, 443 w trybie tcp po SNI, 2222 w trybie tcp do Gitea, a porty pocztowe z pominięciem HAProxy prosto na adres wirtualny pary Dockera. (kliknięcie otwiera pełny rozmiar)

Port	Tryb	Rozstrzyga	Cel	Uwaga
80	http	nagłówek `Host`	MailCow `:8080` dla `poczta`, `autodiscover`, `autoconfig`, reszta do Traefika `:80`	wyzwania ACME HTTP-01 obu systemów przechodzą tą samą drogą
443	tcp	SNI	MailCow `:8443` dla tych samych trzech nazw, reszta do Traefika `:443` z `send-proxy-v2`	klient bez SNI trafia do Traefika
2222	tcp	nic	SSH do Gitea na hoście Dockera	adres klienta w Gitea to adres `lb`
25, 465, 587, 143, 993, 110, 995, 4190	poza HAProxy	`fw01`	adres wirtualny `10.20.10.46`, MailCow na aktywnym hoście	adres nadawcy jest ważny dla Postfiksa i dla SPF, więc nie ukrywamy go za równoważnikiem
8404	http	adres źródłowy	strona statystyk HAProxy	tylko z sieci serwerowej i z hasłem

Kolizja portów 80 i 443 między Traefikiem a MailCow

Strona o Traefiku zakłada, że porty 80 i 443 na `docker01` są wolne, a strona o instalacji MailCow zakłada to samo i ustawia `HTTP_PORT=80` oraz `HTTPS_PORT=443`. Jedna z tych dwóch aplikacji nie wystartuje, a Vaultwarden z kolei opisuje jeszcze Nginksa hosta jako proxy. Na jednym adresie IP i jednym porcie może słuchać tylko jeden proces.

Rozwiązanie jest takie. Traefik zostaje na 80 i 443 i dalej obsługuje wszystkie aplikacje z etykietami. MailCow przechodzi na porty 8080 i 8443 (są to zmienne `HTTP_PORT` i `HTTPS_PORT` w `mailcow.conf`) i nie jest już wystawiony wprost na porty 80 i 443. Rozdział po nazwie robi HAProxy, bo to on jest pierwszym miejscem, w którym nazwa usługi jest widoczna.

Dlaczego nie odwrotnie, czyli Traefik na 8080 i 8443, a MailCow na 80 i 443? Wtedy każde łącze do jakiejkolwiek usługi musiałoby zawierać numer portu albo przechodzić przez `fw01`, a Let's Encrypt wymaga portu 80 dla zwykłego wyzwania HTTP. Dlaczego nie Traefik przed MailCow? Wtedy MailCow potrzebowałby certyfikatu od Traefika albo osobnej obsługi wyzwań ACME dla poczty, a dokumentacja MailCow mówi, że wtedy proxy trzeba przeładowywać po każdym odnowieniu certyfikatu [7].

Zmiana ma konsekwencje, które opisujemy w kroku 8 procedury. Zmieniają się `mailcow.conf`, zapora hostów Dockera i przekierowania na `fw01`. Strona Instalacja MailCow na docker01 od zera opisuje stan bez równoważnika: MailCow za Traefikiem na tym samym hoście (`HTTP_BIND=127.0.0.1`, `SKIP_LETS_ENCRYPT=y`, kontener `nginx-mailcow` w sieci `proxy`). Przy przejściu na parę `lb` zmieniamy `HTTP_BIND` i `HTTPS_BIND` na `0.0.0.0`, usuwamy podpięcie `nginx-mailcow` do Traefika z `docker-compose.override.yml` i wracamy do `acme-mailcow` (`SKIP_LETS_ENCRYPT=n`).

Kontrola stanu

> **Pojęcie: kontrola stanu (health check).** Cykliczna próba połączenia z serwerem backendu, po której równoważnik uznaje go za działający albo nie. Dzięki niej ruch nie trafia do serwera, który padł, a po powrocie serwer wraca do puli bez ręcznej pracy.

Parametry `inter 3s fall 3 rise 2` oznaczają próbę co 3 sekundy, uznanie serwera za niedziałający po trzech porażkach z rzędu (około 9 sekund) i za działający po dwóch sukcesach. W backendzie tcp próba to samo nawiązanie połączenia, a w `be_traefik_http` HAProxy wysyła zapytanie HTTP i przyjmuje każdy kod od 200 do 499. Kod 404 od Traefika jest więc dobrą

odpowiedzią, bo znaczy, że proxy żyje.

W backendzie `be_traefik_https` kontrola stanu wysyła ten sam nagłówek PROXY co normalny ruch: sprawdziliśmy to w HAProxy 2.8.16, gdzie atrapa serwera dostała sygnaturę PROXY v2 także w próbach. Dzięki temu Traefik nie widzi prób jako zepsutych połączeń. Dla gałęzi 2.4 z EL9 zachowanie sprawdzamy w podręczniku (opcja `check-send-proxy`), a jeśli Traefik zgłasza błędy, dopisujemy ją jawnie.

Co ważne, oba hosty Dockera są na liście backendów przez cały czas. Stosy działają tylko na jednym z nich, więc drugi ma porty zamknięte i kontrola stanu sama go pomija. Po przełączeniu przez keepalived na `docker02` HAProxy zauważa to po kilkunastu sekundach, bez żadnej zmiany w konfiguracji.

Utrzymanie sesji: tablica stick

> **Pojęcie: stick-table.** Tablica w pamięci HAProxy, w której zapisujemy, który klient (na przykład po adresie źródłowym) trafił do którego serwera. Dzięki niej ten sam klient wraca do tej samej instancji przez ustalony czas.

W głównym pliku `haproxy.cfg` tablicy nie ma, bo oba hosty Dockera nie pracują równolegle: zawsze działa jeden. Potrzebna jest dopiero wtedy, gdy za HAProxy stoją dwie żywe instancje aplikacji, która trzyma sesję w pamięci procesu. Przykład z `app01` jest w osobnej sekcji niżej.

Strona statystyk i gniazdo administracyjne

Blok `listen stats` wystawia na porcie 8404 stronę z tabelą wszystkich frontendów i backendów, kolorami stanu i licznikami połączeń. Odmawia każdemu spoza sieci `10.20.10.0/24` i wymaga hasła. Gniazdo `/var/lib/haproxy/stats` z poziomem `admin` pozwala zmieniać stan serwerów z wiersza poleceń (karty w Fiszce) i jest dostępne tylko dla roota oraz grupy `haproxy`.

> □□ Zrzut do uzupełnienia: strona `http://10.20.10.64:8404/stats` po zalogowaniu, widoczne sekcje `be_traefik_https`, `be_mailcow_https` i `be_gitea_ssh`, serwer `docker01` na zielono (UP) i `docker02` na czerwono (DOWN).

Jak to łączy się z fw01

FortiGate `fw01` nie zna HAProxy ani keepalived, widzi tylko dwa adresy w sieci serwerowej: `10.20.10.66` dla portów webowych i `10.20.10.46` dla portów pocztowych. Przekierowanie portów (w FortiGate to obiekt typu Virtual IP z włączonym przekierowaniem portu, nie mylić z naszym adresem wirtualnym) wskazuje te adresy, a nie adresy maszyn. Gdy keepalived przenosi adres na drugą maszynę, ta rozsyła gratisowy ARP (opis w stronie o przenoszeniu kontenerów), a `fw01` uczy się nowego adresu MAC.

Przekierowanie na fw01	Adres zewnętrzny	Port	Adres docelowy	Port docelowy
WWW	`203.0.113.10`	80	`10.20.10.66`	80
WWW szyfrowane	`203.0.113.10`	443	`10.20.10.66`	443
SSH do Gitea (jeśli ma być dostępne z zewnątrz)	`203.0.113.10`	2222	`10.20.10.66`	2222
Poczta (osobno każdy port)	`203.0.113.10`	25, 465, 587, 143, 993, 110, 995, 4190	`10.20.10.46`	te same

> Zrzut do uzupełnienia: ekran `fw01` z listą obiektów Virtual IP po zmianie, widoczne trzy przekierowania webowe na `10.20.10.66` i przekierowania pocztowe na `10.20.10.46`. Nazwy menu i składnię poleceń sprawdzamy w dokumentacji FortiOS: nie udało się jej pobrać w czasie pisania tej strony.

Przekierowanie portu zachowuje adres klienta, jeśli polityka nie włącza dodatkowo translacji adresu źródłowego dla ruchu przychodzącego. Sprawdzamy to w pierwszym teście: w statystykach HAProxy albo w dzienniku Traefika ma być adres z internetu, nie adres `fw01`. Komputery z sieci biura nie przechodzą przez `fw01` w tej sprawie: wewnętrzne rekordy DNS wskazują wprost na `10.20.10.66`.

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.

Zanim zaczniemy

Potrzebujemy dwóch maszyn wirtualnych z Rocky Linux 9, ustawionych jak `docker01` (strona Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker, bez części o Dockerze, bo na `lb` go nie instalujemy). `lb01` stoi na `hv01` z adresem `10.20.10.64`, a `lb02` na `hv02` z adresem `10.20.10.65`, obie w VLAN 10. Dwa procesory i 2 GB pamięci wystarczą, bo HAProxy nie deszyfruje ruchu. Adresy `10.20.10.64` do `10.20.10.66` sprawdziliśmy w całej bazie wiedzy: nie są przydzielone nigdzie indziej.

Żeby klaster Hyper-V nie postawił obu maszyn na jednym hoście, ustawiamy miękką regułę rozdzielania tak samo jak dla `docker01` i `docker02` (komenda z tej strony, z nową nazwą klasy):

```
$AntiAffinity = New-Object System.Collections.Specialized.StringCollection
$AntiAffinity.Add("LB")
(Get-ClusterGroup -Name "lb01").AntiAffinityClassNames = $AntiAffinity
(Get-ClusterGroup -Name "lb02").AntiAffinityClassNames = $AntiAffinity
```

Adres `10.20.10.66` z innych sieci wymaga tylko trasy przez `fw01`, jak każdy adres w VLAN 10. Test awarii na końcu robimy poza zakazem zmian od 15 listopada do 6 stycznia i nie w piątek po południu.

1. Instalacja pakietów

```
sudo dnf install -y haproxy keepalived socat
haproxy -v | head -n 1
```

W EL9 pakiet `keepalived` ma wersję 2.2.x, a pakiet `haproxy` gałąź 2.4 według list pakietów RHEL 9. Dokładne wersje sprawdzamy w `dnf info`, bo w tej sesji ich nie potwierdziliśmy dla Rocky 9. Konfiguracja z tej strony używa wyłącznie składni dostępnej w gałęzi 2.4, a sprawdziliśmy ją programem `haproxy -c` w wersji 2.8.16. Gałąź LTS 3.4 (wydana 3 czerwca 2026) nie jest nam do niczego potrzebna, a jej dostępności w repozytoriach EL9 nie sprawdzaliśmy [3].

2. Zapora firewalld i SELinux

Na obu maszynach otwieramy porty usług, stronę statystyk dla sieci serwerowej i protokół VRRP od partnera (na `lb01` w `AdresPartnera` wpisujemy `10.20.10.65`, na `lb02` wpisujemy `10.20.10.64`):

```
sudo firewall-cmd --permanent --add-port=80/tcp --add-port=443/tcp --add-port=2222/tcp
sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source
address="10.20.10.0/24" port port="8404" protocol="tcp" accept'
```

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

Dla HAProxy na SELinux w trybie enforcing łączenie się z portami spoza standardowej listy (8080, 8443, 2222, 8404) wymaga zwykle wartości logicznej `haproxy_connect_any`. Nazwę wartości sprawdzamy w `getsebool -a | grep haproxy`, a zachowania w naszym środowisku nie mieliśmy jak potwierdzić:

```
getenforce
sudo setsebool -P haproxy_connect_any 1
```

Po pierwszym uruchomieniu patrzymy w `sudo ausearch -m avc -ts recent`. Odmowy dla `haproxy_t` albo `keepalived_t` omawiamy z autorem, zamiast wyłączać SELinux.

3. Plik haproxy.cfg

Ten sam plik jest na `lb01` i `lb02`. Zawiera trzy frontendy (80, 443, 2222), pięć backendów i blok statystyk, a każdy z hostów Dockera jest na liście każdego backendu:

```
sudo vim /etc/haproxy/haproxy.cfg
```

Pokaż pełny przykład (93 linii)

```
# /etc/haproxy/haproxy.cfg
# Ten sam plik na lb01 i lb02. HAProxy nie kończy TLS: szyfrowanie kończą Traefik i
MailCow.
global
    log stdout format raw local0 info
    maxconn 4000
    user haproxy
    group haproxy
    stats socket /var/lib/haproxy/stats mode 660 level admin

defaults
    log global
    option redispatch
    retries 2
    timeout connect 5s
```

```
    timeout client 60s
    timeout server 60s
    timeout tunnel 1h

# Port 80: tryb http, host z nagłówka Host decyduje o backendzie.
frontend fe_http
    bind :80
    mode http
    option httplog
    option forwardfor
    timeout http-request 10s
    acl host_poczta hdr(host),field(1,:) -i poczta.zielona.example
autodiscover.zielona.example autoconfig.zielona.example
    use_backend be_mailcow_http if host_poczta
    default_backend be_traefik_http

# Port 443: tryb tcp, nazwa z SNI decyduje o backendzie, TLS przechodzi bez zmian.
frontend fe_https
    bind :443
    mode tcp
    option tcplog
    tcp-request inspect-delay 5s
    tcp-request content accept if { req_ssl_hello_type 1 }
    acl sni_poczta req_ssl_sni -i poczta.zielona.example autodiscover.zielona.example
autoconfig.zielona.example
    use_backend be_mailcow_https if sni_poczta
    default_backend be_traefik_https

# Port 2222: SSH do Gitea, tryb tcp bez żadnej inspekcji.
frontend fe_gitea_ssh
    bind :2222
    mode tcp
    option tcplog
    default_backend be_gitea_ssh

# Backendy: oba hosty Dockera są na liście, ale stosy działają tylko na aktywnym.
# Host rezerwowy nie ma otwartych portów, więc kontrola stanu oznacza go jako DOWN.
backend be_traefik_http
    mode http
```

```
    option httpchk
    http-check send meth GET uri / ver HTTP/1.1 hdr Host traefik.zielona.example
    http-check expect rstatus ^[2-4][0-9][0-9]$
    default-server inter 3s fall 3 rise 2
    server docker01 10.20.10.42:80 check
    server docker02 10.20.10.47:80 check

backend be_traefik_https
    mode tcp
    default-server inter 3s fall 3 rise 2
    server docker01 10.20.10.42:443 check send-proxy-v2
    server docker02 10.20.10.47:443 check send-proxy-v2

backend be_mailcow_http
    mode http
    default-server inter 3s fall 3 rise 2
    server docker01 10.20.10.42:8080 check
    server docker02 10.20.10.47:8080 check

backend be_mailcow_https
    mode tcp
    default-server inter 3s fall 3 rise 2
    server docker01 10.20.10.42:8443 check
    server docker02 10.20.10.47:8443 check

backend be_gitea_ssh
    mode tcp
    default-server inter 3s fall 3 rise 2
    server docker01 10.20.10.42:2222 check
    server docker02 10.20.10.47:2222 check

# Strona statystyk: tylko z sieci serwerowej i z hasłem.
listen stats
    bind :8404
    mode http
    acl siec_serwerow src 10.20.10.0/24
    http-request deny unless siec_serwerow
    stats enable
    stats uri /stats
```

```
    stats refresh 10s
    stats show-node
    stats auth nazwa_usera:HasloStatystyk
```

> (gdzie `nazwa_usera:HasloStatystyk` to login i hasło do strony statystyk; hasło zapisujemy w Vaultwardenie, a plik dostaje prawa `640` z właścicielem `root:haproxy`)

Kilka rzeczy do zapamiętania w tym pliku. Adresy `bind :80` i `bind :443` bez adresu oznaczają, że HAProxy słucha na wszystkich adresach maszyny, więc nie wymaga ustawienia `net.ipv4.ip_nonlocal_bind`: adres wirtualny pojawi się na maszynie dopiero po przejęciu roli. Opcja `option forwardfor` dopisuje do żądań HTTP nagłówek z adresem klienta. Linia `acl host_poczta` odcina numer portu z nagłówka `Host` przez `field(1,:)`.

```
sudo chown root:haproxy /etc/haproxy/haproxy.cfg
sudo chmod 640 /etc/haproxy/haproxy.cfg
sudo haproxy -c -f /etc/haproxy/haproxy.cfg
sudo systemctl enable --now haproxy
```

4. Skrypty dla keepalived

Skrypt sprawdzający wywołuje keepalived co 3 sekundy. Pyta tylko o to, co decyduje o całej maszynie: czy HAProxy żyje, czy słucha na portach i czy widzi bramę `fw01`. Stan backendów nie wchodzi w kontrolę, bo gdy oba hosty Dockera padną, przeniesienie adresu na `lb02` niczego nie naprawi.

```
sudo vim /usr/local/sbin/ft-ha-sprawdz-lb.sh
```

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-sprawdz-lb.sh
# Wywoływany przez keepalived co kilka sekund.
# Kod 0: lb zdolny do roli aktywnej. Kod 1: lb niezdolny.
set -u
BRAMA=10.20.10.1

# 1. Usługa HAProxy musi działać.
systemctl is-active --quiet haproxy || exit 1
```

```
# 2. HAProxy musi słuchać na portach 80 i 443.
for port in 80 443; do
  ss -H -ltn "sport = :${port}" | grep -q . || exit 1
done

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

exit 0
```

Drugi skrypt tylko zapisuje zmianę roli, bo na `lb` nie ma stosów do uruchamiania:

```
sudo vim /usr/local/sbin/ft-ha-lb-rola.sh
```

```
#!/usr/bin/env bash
# /usr/local/sbin/ft-ha-lb-rola.sh
# Wywoływany przez keepalived przy zmianie roli: master, backup, fault, stop.
ROLA="${1:-}"
echo "$(date -Iseconds) $(hostname -s) rola ${ROLA}" >> /var/log/ft-ha.log
logger -t ft-ha "lb: rola ${ROLA}"
exit 0
```

```
sudo chown root:root /usr/local/sbin/ft-ha-sprawdz-lb.sh /usr/local/sbin/ft-ha-lb-rola.sh
sudo chmod 755 /usr/local/sbin/ft-ha-sprawdz-lb.sh /usr/local/sbin/ft-ha-lb-rola.sh
```

5. Plik keepalived.conf

Plik na `lb01`:

```
sudo vim /etc/keepalived/keepalived.conf
```

Pokaż pełny przykład (43 linii)

```
global_defs {
    router_id lb01
    enable_script_security
    script_user root
}
```

```
# Brak jawnej wagi: po trzech porażkach z rzędu instancja przechodzi w stan FAULT
# i oddaje adres partnerowi.
vrrp_script ft_ha_sprawdz_lb {
    script "/usr/local/sbin/ft-ha-sprawdz-lb.sh"
    interval 3
    timeout 3
    fall 3
    rise 2
}

vrrp_instance VI_LB {
    # Oba węzły startują jako BACKUP, a nopreempt działa tylko przy takim stanie
początkowym.
    state BACKUP
    nopreempt
    interface ens192
    virtual_router_id 66
    priority 150
    advert_int 1

    unicast_src_ip 10.20.10.64
    unicast_peer {
        10.20.10.65
    }

    virtual_ipaddress {
        10.20.10.66/24 dev ens192
    }

    track_script {
        ft_ha_sprawdz_lb
    }

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

Na `lb02` ten sam plik zmienia się w czterech miejscach:

Parametr	lb01	lb02
`router_id`	`lb01`	`lb02`
`priority`	`150`	`100`
`unicast_src_ip`	`10.20.10.64`	`10.20.10.65`
`unicast_peer`	`10.20.10.65`	`10.20.10.64`

Identyfikator `virtual_router_id 66` jest inny niż `46` pary Dockera, bo oba adresy wirtualne leżą w tym samym VLAN 10, a dwie instancje VRRP z tym samym identyfikatorem w jednej sieci myliłyby się nawzajem. Plik sprawdziliśmy komendą `keepalived -t` w wersji 2.2.8 (tej samej, którą ma EL9) na dwóch przestrzeniach nazw sieciowych z interfejsem `ens192`, oba pliki przeszły bez ostrzeżeń.

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

Po starcie `lb01` powinien zostać MASTER (wyższy priorytet), a `lb02` BACKUP. Obie maszyny startują w stanie BACKUP, a pierwsza, która nie usłyszy ogłoszeń, po kilku sekundach przejmuje adres.

6. Zmiany na docker01 i docker02

Cztery zmiany robimy na obu hostach (`docker02` dostaje te same pliki z synchronizacji z poprzedniej strony, ale `mailcow.conf` jest poza listą stosów, więc sprawdzamy go ręcznie).

Pierwsza: MailCow na portach 8080 i 8443. W `/docker/apps/mailcow/mailcow.conf` zmieniamy wiersze (wartości `HTTP_BIND` i `HTTPS_BIND` zostają `0.0.0.0`, bo równoważnik jest na innej maszynie, a dokumentacja MailCow podpowiada `127.0.0.1` tylko dla proxy na tym samym hoście [7]):

```
HTTP_PORT=8080
HTTPS_PORT=8443
```

```
cd /docker/apps/mailcow
sudo docker compose up -d
```

> Dokumentacja MailCow przestrzega, żeby nie używać portów 8081, 9081 i 65510, bo zajmują je wewnętrzne usługi [7]. Po zmianie sprawdzamy `docker compose logs --tail 50 acme-mailcow`: kontener ma nadal dostawać wyzwania ACME na porcie 80, tym razem przez HAProxy.

Druga: Traefik ma przyjmować PROXY protocol od równoważnika i ufać nagłówkom z jego adresów. Do listy `command` w `/docker/compose/traefik/compose.yaml` dopisujemy trzy wiersze (adres `10.20.10.64/31` obejmuje dokładnie `lb01` i `lb02`):

```
services:
  traefik:
    command:
      # ... dotychczasowe wiersze bez zmian, a poniżej dopisujemy:
      - "--entrypoints.websecure.proxyProtocol.trustedIPs=10.20.10.64/31"
      - "--entrypoints.web.forwardedHeaders.trustedIPs=10.20.10.64/31"
      - "--entrypoints.websecure.forwardedHeaders.trustedIPs=10.20.10.64/31"
```

```
cd /docker/compose/traefik
docker compose up -d
```

Skrypt `ft-ha-sprawdz.sh` ze strony o przenoszeniu kontenerów pyta Traefika przez `https://127.0.0.1/`, czyli z adresu spoza listy zaufanych. Według dokumentacji Traefika nagłówek PROXY jest wtedy ignorowany, ale nie sprawdziliśmy, czy połączenie bez nagłówka z niezaufanego adresu jest przyjmowane. To punkt testu w sekcji Weryfikacja: jeśli skrypt zacznie widzieć kod `000`, Traefik odrzuca takie połączenia i musimy ponownie przemyśleć kontrolę.

Trzecia: porty 8080, 8443 i 2222 na hostach Dockera ma widzieć równoważnik, a nie cała sieć biura. Reguła dla firewalld wygląda tak (ta sama na obu hostach):

```
sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source
address="10.20.10.64/31" port port="8080" protocol="tcp" accept'
sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source
address="10.20.10.64/31" port port="8443" protocol="tcp" accept'
sudo firewall-cmd --reload
```

> Docker publikuje porty własnymi regułami, które często omijają firewalld, więc reguły powyżej mogą niczego nie ograniczać. Z innego hosta (nie z `lb`) próbujemy `curl http://10.20.10.42:8080`. Jeśli odpowiedź przychodzi, ograniczenie trzeba dodać w łańcuchu `DOCKER-USER`, a to wykracza poza tę stronę i wymaga sprawdzenia w dokumentacji Dockera dla wersji 29.

Czwarta: `poczta`, `autodiscover` i `autoconfig` w publicznym DNS nadal wskazują `203.0.113.10`, więc nic się nie zmienia. Zmienia się tylko to, co `fw01` robi z ruchem na tym adresie (krok 7).

7. Przekierowania na fw01

Zmieniamy cele przekierowań według tabeli z sekcji „Jak to łączy się z fw01”: porty 80, 443 i (jeśli wystawiamy) 2222 na `10.20.10.66`, porty pocztowe na `10.20.10.46`. Dotychczasowe wpisy kierujące na `docker01` zostawiamy wyłączone, nie usunięte, do końca testu, bo to najszybszy sposób powrotu.

8. DNS

Zakładamy rekord adresu wirtualnego równoważnika i przekierowujemy na niego nazwy aplikacji, które do tej pory wskazywały `docker-ha` (komendy z `dc01`, w PowerShellu administratora, w tym samym stylu co na stronie o przenoszeniu kontenerów):

```
Add-DnsServerResourceRecordA -ZoneName "ad.zielona.example" -Name "lb-ha" -IPv4Address
"10.20.10.66" -CreatePtr
```

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

Drugi blok powtarzamy dla każdej nazwy aplikacji z Traefika (`git`, `hasla`, `traefik` i kolejnych). Rekord `docker-ha` z poprzedniej strony zostaje, bo wskazuje parę Dockera, a pocztowe porty nadal idą przez `10.20.10.46`. TTL tych rekordów obniżamy do 300 sekund na dzień przed zmianą i przywracamy po teście.

Weryfikacja

1. Na `lb01` pokazujemy adres wirtualny (karta w Fiszce). Na `lb02` ma go nie być.
2. Na obu maszynach odczytujemy stan serwerów (karta „Odczyt stanu”): aktywny host Dockera ma wszędzie `UP`, rezerwowy `DOWN`.
3. Ze stacji administratora próbujemy trasy przez adres wirtualny (karta `curl --resolve`) dla `git.zielona.example`. Oczekujemy kodu 200.
4. Dla MailCow sprawdzamy, że nazwa z SNI trafia do innego certyfikatu niż pozostałe:

```
echo | openssl s_client -connect 10.20.10.66:443 -servername poczta.zielona.example 2>
/dev/null | openssl x509 -noout -subject -issuer
echo | openssl s_client -connect 10.20.10.66:443 -servername git.zielona.example 2> /dev/null
| openssl x509 -noout -subject -issuer
```

Pierwsza komenda ma pokazać certyfikat MailCow (Let's Encrypt albo własny), druga certyfikat Traefika (`*.zielona.example` wystawiony przez `CA-Zielona`). Wspólny wynik oznacza, że ruch trafia do jednego backendu: sprawdzamy `acl sni_poczta` i to, czy MailCow słucha na `8443`.

1. W dzienniku Traefika (`docker compose logs traefik --tail 20`) ma być adres prawdziwego klienta, nie `10.20.10.64`. Wejście bezpośrednie na `https://10.20.10.42/` z hosta spoza `lb` sprawdzamy osobno: z niezaufanego adresu połączenie powinno działać bez nagłówka PROXY. Bez wsparcia po stronie Traefika próbujemy `curl --haproxy-protocol`, który wysyła nagłówek PROXY v1.
2. Na `docker01` sprawdzamy, że skrypt `ft-ha-sprawdz.sh` nadal przechodzi (`sudo /usr/local/sbin/ft-ha-sprawdz.sh; echo $?`) i że MailCow odpowiada na `8443`.
3. Z internetu (telefon w sieci komórkowej) otwieramy webmail pod `https://poczta.zielona.example` i wysyłamy próbną wiadomość na adres zewnętrzny.

Wariant: równoważenie przed aplikacją app01

HAProxy przydaje się także tam, gdzie nie ma Dockera. Aplikacja Javy zespołu Oli na `app01` może działać w dwóch instancjach na różnych portach (`8081` i `8082`), a HAProxy rozdziela ruch między nie i pomija tę, która nie odpowiada. Taki układ chroni przed padem procesu i pozwala restartować instancje po kolei, ale nie przed awarią samej maszyny `app01`.

Dopisujemy do `haproxy.cfg` jeden backend i jedną regułę w `fe_http` (pod linią `use_backend be_mailcow_http`):

```
    acl host_app hdr(host),field(1,:) -i app.zielona.example
    use_backend be_app01 if host_app
```

```
backend be_app01
    mode http
    balance roundrobin
    option httpchk
    http-check send meth GET uri /health ver HTTP/1.1 hdr Host app.zielona.example
    http-check expect status 200
    # Utrzymanie sesji: ten sam klient wraca do tej samej instancji przez 30 minut.
    stick-table type ip size 200k expire 30m
    stick on src
    default-server inter 3s fall 3 rise 2
    server app01-a 10.20.10.41:8081 check
```

```
    server app01-b 10.20.10.41:8082 check
```

Ścieżka `/health` to założenie: aplikacje zespołu Oli muszą ją wystawiać albo wpisujemy inną, którą zwracają kodem 200. Tablica `stick-table` zapisuje adres klienta i instancję, a `stick on src` odczytuje ją przy kolejnych połączeniach. To najprostsze utrzymanie sesji i działa także w trybie tcp, ale wiele osób za jednym adresem (cała sieć biura za jednym adresem źródłowym) trafi wtedy do tej samej instancji. Alternatywą w trybie http jest ciasteczko (`cookie SRV insert indirect nocache` i `cookie` przy serwerach), a docelowo najlepsza jest aplikacja bez sesji w pamięci procesu.

Jeśli aplikacja ma być dostępna po HTTPS, ruch kierujemy do niej po SNI (jak MailCow) albo przez Traefika. Terminacja TLS na `lb01` i `lb02` (`bind :443 ssl crt ...`) wymagałaby kopii klucza prywatnego na obu maszynach i nie robimy jej bez decyzji autora.

Test awarii

Testu nie robimy po raz pierwszy w dniu awarii i nie robimy go od 15 listopada do 6 stycznia. Przed testem zapisujemy godzinę i uruchamiamy z drugiej maszyny pętlę, która co pół sekundy pyta o usługę przez adres wirtualny:

```
while true; do curl -ks -o /dev/null -m 2 -w "%{time_total} %{http_code}\n" --resolve
git.zielona.example:443:10.20.10.66 https://git.zielona.example/; sleep 0.5; done
```

Wiersze z kodem `000` oznaczają brak odpowiedzi. Liczymy ich czas, a nie ich liczbę, bo przerwy trwają sekundy.

A. Planowe wyłączenie keepalived na aktywnym lb

1. Na `lb01` wykonujemy `sudo systemctl stop keepalived`.
2. Na `lb02` sprawdzamy, że adres wirtualny pojawił się po sekundzie lub dwóch (`ip -br -4 addr show dev ens192`).
3. Na `lb01` uruchamiamy `sudo systemctl start keepalived`. Ma wrócić jako BACKUP i nie odbierać adresu (`nopreempt`).

B. Awaria HAProxy na aktywnym lb

1. Na aktywnej maszynie `sudo systemctl stop haproxy`. Skrypt `ft-ha-sprawdz-lb.sh` zaczyna zwracać kod 1.

2. Po trzech porażkach z rzędu (około 9 sekund) keepalived przechodzi w stan FAULT i oddaje adres.
3. `sudo systemctl start haproxy` przywraca usługę, a maszyna po dwóch udanych kontrolach wraca jako BACKUP.

C. Cicha awaria: maszyna żyje, ale nie ogłasza się

Blokujemy ruch VRRP wychodzący z aktywnej maszyny (reguła `iptables` albo firewalld, w laboratorium albo w oknie serwisowym przy obu maszynach pod ręką). Partner po około trzech sekundach uznaje, że aktywna nie żyje, i przejmuje adres, ale stara maszyna nadal go trzyma. To jest split brain w praktyce: oba adresy odpowiadają, a klienci trafiają raz tu, raz tam. Po zdjęciu blokady wygrywa maszyna o wyższym priorytecie, a druga wraca do BACKUP.

D. Awaria hosta Dockera

1. Na aktywnym `docker01` wykonujemy `Stop-ClusterGroup -Name "docker01"` na klastrze Hyper-V (jak w teście z poprzedniej strony).
2. Pętla z `curl` pokazuje przerwę. W tym czasie keepalived pary Dockera przejmuje adres `10.20.10.46`, uruchamia stosy na `docker02`, a HAProxy po kilku kontrolach oznacza `docker01` jako `DOWN`, a `docker02` jako `UP`.
3. W statystykach (karta „Odczyt stanu”) obserwujemy zmianę. Czas od początku przerwy do pierwszego kodu 200 zapisujemy w tabeli wyników.
4. `Start-ClusterGroup -Name "docker01"` kończy test. Host wraca jako rezerwa, a HAProxy nadal kieruje na `docker02`.

Co zmierzyliśmy w piaskownicy

Konfigurację keepalived przetestowaliśmy w dwóch przestrzeniach nazw sieciowych Linuksa (osobne stosy sieciowe połączone parą interfejsów, nazwanych `ens192`), z keepalived 2.2.8 i prostszą wersją skryptu sprawdzającego. Konfigurację HAProxy przetestowaliśmy w wersji 2.8.16 z atrapami backendów na lokalnych portach (serwery HTTP i `openssl s_server` z dwoma certyfikatami). To są próby mechanizmu, nie pomiary Zielonej: na `lb01` i `lb02` liczby wyjdą inne.

Próba	Wynik w piaskownicy
Oba keepalived startują jednocześnie jako BACKUP	po około 4 sekundach MASTER zostaje maszyna z wyższym priorytetem
`systemctl stop keepalived` na MASTER	partner przejmuje adres w tej samej sekundzie, bo zatrzymany keepalived ogłasza priorytet zero
Powrót zatrzymanego keepalived	wraca jako BACKUP, adres zostaje u partnera (`nopreempt`)

Próba	Wynik w piaskownicy
Skrypt sprawdzający zwraca 1	po trzech porażkach (około 6 do 7 sekund przy `interval 3`) stan FAULT, partner MASTER sekundę później
Zablokowane ogłoszenia VRRP z MASTER	partner MASTER po około 3 sekundach, oba trzymają adres, po zdjęciu blokady wygrywa wyższy priorytet
`set server ... state maint`	serwer `MAINT`, żądania do jedynego serwera backendu dostają 503
Atrapa `docker01` zatrzymana, atrapa `docker02` uruchomiona	`docker01` `DOWN` i `docker02` `UP` po kilkunastu sekundach, odpowiedź od drugiej atrapy
SNI `poczta` i `git` na porcie 443	dwa różne certyfikaty (atrapy backendów)

Z próby ostatnich dwóch wierszy tabeli wynika ważna konsekwencja: `nopreempt` chroni przed odbieraniem roli przez wracającą maszynę o wyższym priorytecie, ale gdy po rozdzieleniu sieci obie są MASTER, wygrywa wyższy priorytet. Dlatego `lb01` ma wyższy priorytet, a nie niższy: to ona zostaje aktywna po naprawie sieci.

Wycofanie zmian

Wracamy do jednego punktu wejścia na `docker01`, nie ruszając danych:

1. Na `fw01` przywracamy wyłączone przekierowania na `docker01` (porty 80, 443, 2222 i pocztowe) i wyłączamy te na `10.20.10.66`.
2. W DNS przywracamy `CNAME` aplikacji na `docker-ha.ad.zielona.example` (komenda z kroku 8 z innym celem).
3. W `/docker/apps/mailcow/mailcow.conf` przywracamy `HTTP_PORT=80` i `HTTPS_PORT=443`, a Traefika przenosimy na inny port albo zatrzymujemy, bo ta kolizja z początku strony wraca. Decyzję, co ma stać na 80 i 443, podejmuje autor.
4. Z `compose.yaml` Traefika usuwamy trzy wiersze `trustedIPs` i robimy `docker compose up -d`.
5. Na `lb01` i `lb02` wyłączamy usługi: `sudo systemctl disable --now keepalived haproxy`.

Częste błędy

Objaw	Przyczyna	Co robimy
`git push` przez SSH nie działa po zmianie DNS na adres wirtualny	HAProxy nie słucha na porcie 2222	frontend `fe_gitea_ssh` z pliku z tej strony, `reload`, port otwarty w firewalld
Oba backendy MailCow i Traefika są `DOWN`, a hosty Dockera działają	zapora hosta zatrzymuje ruch z `lb`, albo MailCow dalej słucha na 80 i 443	`curl` z `lb` na `10.20.10.42:8443`, `ss -ltn` na hoście Dockera

Objaw	Przyczyna	Co robimy
Przeglądarka pokazuje certyfikat Traefika dla `poczta.zielona.example`	nazwa nie jest wpisana w `acl sni_poczta`, albo klient nie wysyła SNI	dopisujemy nazwę, sprawdzamy `openssl s_client -servername`
Traefik zrywa połączenia od `lb`	brak `proxyProtocol.trustedIPs` w Traefiku albo nagłówek PROXY wysyłany do backendu bez obsługi	`send-proxy-v2` tylko przy `be_traefik_https`, `trustedIPs` z adresami `lb`
Zabezpieczenie MailCow banuje `10.20.10.64`	MailCow widzi `lb` jako źródło połączeń webowych	`lb01` i `lb02` na białą listę MailCow albo wariant A (TLS na `lb`), decyzja autora
Oba `lb` trzymają adres wirtualny	zapora blokuje protokół VRRP albo `unicast_peer` wskazuje zły adres	rich rule z `protocol value="vrrp"`, `tcpdump -ni ens192 proto 112`
Po awarii `lb01` wraca i odbiera adres	brak `nopreempt` albo `state MASTER` zamiast `BACKUP`	oba węzły `state BACKUP` i `nopreempt`
Po `reload` HAProxy błąd o `Permission denied` na gnieździe	zły właściciel katalogu `/var/lib/haproxy` albo SELinux	`ls -ld /var/lib/haproxy`, `ausearch -m avc -ts recent`
Strona statystyk odpowiada 403	adres źródłowy poza `10.20.10.0/24`	otwieramy stronę ze stacji w VLAN 10 albo zmieniamy `acl siec_serwerow`
`haproxy -c` zgłasza błąd przy `http-check send`	starsza gałąź niż 2.2	używamy pakietu z EL9 (2.4), sprawdzamy `haproxy -v`
Wiersze `000` w pętli testowej trwają ponad 10 sekund	`fall 3` i `inter 3s` w kontroli stanu plus czas startu stosów	skracamy `inter` albo zostawiamy i podajemy czas właścicielom usług

□□ Ćwiczenia

1. Dlaczego w `be_traefik_https` jest `send-proxy-v2`, a w `be_mailcow_https` go nie ma?

Odpowiedź

Nagłówek PROXY musi zrozumieć serwer po drugiej stronie. Traefik ma włączoną obsługę na punkcie wejścia `websecure` i ufa adresom `lb`, więc dostaje prawdziwy adres klienta. Dla MailCow nie potwierdziliśmy w dokumentacji takiej obsługi, więc nagłówek zostałby potraktowany jak zepsute dane i połączenie by się zerwało.

1. Aktywny jest `lb01`. Tomek zatrzymuje na nim `haproxy`, ale zostawia `keepalived`. Co się dzieje z adresem `10.20.10.66` i po jakim czasie?

Odpowiedź

Skrypt `ft-ha-sprawdz-lb.sh` zaczyna zwracać kod 1. Po trzech porażkach z rzędu, przy `interval 3`, czyli po około 9 sekundach, keepalived przechodzi w stan FAULT i oddaje adres. `lb02` przejmuje go w następnej sekundzie. W tym czasie żądania do adresu wirtualnego nie mają odpowiedzi, bo HAProxy nie działa.

1. `docker01` pada o 16:05, a keepalived pary Dockera przenosi stosy na `docker02`. Czy trzeba zmieniać konfigurację HAProxy, żeby ruch trafił do `docker02`?

Odpowiedź

Nie. Oba hosty są na liście backendów przez cały czas, a kontrola stanu sama oznacza `docker01` jako `DOWN` i `docker02` jako `UP`, gdy tylko Traefik na nim odpowie. Czas przerwy to czas przełączenia stosów plus kilka kontroli stanu (około 6 do 9 sekund przy `inter 3s` i `rise 2`).

1. Dlaczego porty pocztowe idą z `fw01` wprost na `10.20.10.46`, a nie przez HAProxy?

Odpowiedź

Postfix ocenia nadawcę po adresie źródłowym (SPF, limity połączeń, blokady po nieudanych logowaniach). Ukryty za równoważnikiem w trybie tcp widziałby tylko adresy `lb`. MailCow na aktywnym hoście Dockera słucha na wszystkich adresach, więc przejmuje też adres `10.20.10.46`, a przełączanie pary Dockera załatwia keepalived.

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 danych między hostami: DRBD, GlusterFS, Ceph, ZFS i replikacja w aplikacji · Docker Swarm: klaster z trzema menedżerami, na którym przenosimy kolorowy nginx · k3s z trzema węzłami serwerowymi: etcd, adres wirtualny API i Longhorn · Reverse proxy w Dockerze: Traefik przed wszystkimi aplikacjami na docker01 · Instalacja MailCow na docker01 od zera · Firewalld

Źródła:

1. keepalived.org: keepalived.conf(5) oraz strona pobrań (wydanie 2.4.3 z 20 lipca 2026)
2. IETF: RFC 5798, Virtual Router Redundancy Protocol (VRRP) Version 3
3. haproxy.org: strona główna, gałęzie i daty wsparcia (odczyt 3 października 2026)
4. docs.haproxy.org: HAProxy 2.4 Configuration Manual
5. docs.haproxy.org: opcje `send-proxy-v2`, `check-send-proxy`, `option httpchk`, `stick-table` w podręczniku konfiguracji

6. doc.traefik.io: EntryPoints, opcje `proxyProtocol` i `forwardedHeaders`
7. docs.mailcow.email: Reverse proxy oraz strona o HAProxy (opisana jako wspierana przez społeczność)
8. docs.docker.com: Use swarm mode routing mesh, przykład HAProxy przed usługą

Revision #14
Created 2026-10-03 13:44:07 UTC by Tomasz Konieczny
Updated 2026-10-26 14:21:49 UTC by Tomasz Konieczny