Reverse proxy w Dockerze: Traefik przed wszystkimi aplikacjami na docker01

Typ: procedura **Dotyczy: Traefik v3.5.0** **Czas: ~45 min**

Efekt: jeden kontener Traefika na `docker01`, nasłuchujący na portach 80 i 443, który przyjmuje ruch do wszystkich aplikacji dockerowych w Zielonej, sam przekierowuje na HTTPS i sam wystawia certyfikat, zamiast osobnego wpisu w Nginksie dla każdej z nich.

ZAŁOŻENIA SCENARIUSZA

`docker01` stoi za pojedynczym `fw01`, który przekierowuje porty 80 i 443 z adresu 203.0.113.10 (jedno łącze `wan1`, adres stały); Let's Encrypt potrzebuje portu 80 osiągalnego na tym adresie. Przy drugim łączu ze Starlinkiem o publicznym, ale zmiennym adresie przekierowania i rekordy DNS muszą wskazywać adres łącza aktywnego: Łącza operatora: światłowód, radiolinia, LTE i 5G oraz przełączanie na FortiGate.

Skróty użyte na tej stronie (8)

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
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Zanim zaczniemy

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen `vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał systemu.

> **□□ Z ŻYCIA WZIĘTE**
>
> Środa, 11:00. Ola prosi o piąty w tym miesiącu wpis w konfiguracji Nginksa: nowa wewnętrzna usługa jej zespołu ma dostać własny adres. Za każdym razem to ta sama procedura: nowy plik `server` w Nginksie, `nginx -t`, `systemctl reload`, i tak przy każdej z kilkunastu aplikacji na `docker01`. Traefik odwraca ten porządek: to aplikacja, przy starcie, mówi proxy o sobie przez etykiety we własnym pliku Compose, a Traefik dogląda zmian sam, bez ręcznego dopisywania czegokolwiek po stronie proxy.

Procedura zakłada działający Docker Engine z wtyczką Compose, zainstalowany według strony Instalacja Dockera na Rocky Linux 9 z osobnym wolumenem /docker, i katalogi zakładane zgodnie z konwencją opisaną na stronie Jak trzymamy aplikacje w Dockerze: konwencje Zielonej. Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo albo jako user należący do grupy `docker`.

Wymaganie	Uwaga
Porty 80 i 443 wolne na `docker01`	żaden inny kontener nie publikuje ich bezpośrednio na hoście
Rekordy DNS dla adresów `*.zielona.example` wskazujące na `docker01`	wewnętrzne przez `dc01`, publiczne przez dostawcę DNS strefy `zielona.example` (strona Domena i strefa publiczna: rejestrator, dostawca DNS, odnowienia i blokady)
Certyfikat serwera dla adresów wewnętrznych `*.zielona.example`, wystawiony przez urząd `CA-Zielona` (pliki `wildcard-zielona.crt` i `wildcard-zielona.key`)	klucz i żądanie powstają na `docker01`, urząd na `dc01` tylko podpisuje żądanie, patrz Generowanie certyfikatów: OpenSSL i urząd certyfikacji Windows
Adres publiczny biura osiągalny z internetu na porcie 80	wymóg weryfikacji ACME przez Let's Encrypt dla adresów publicznych

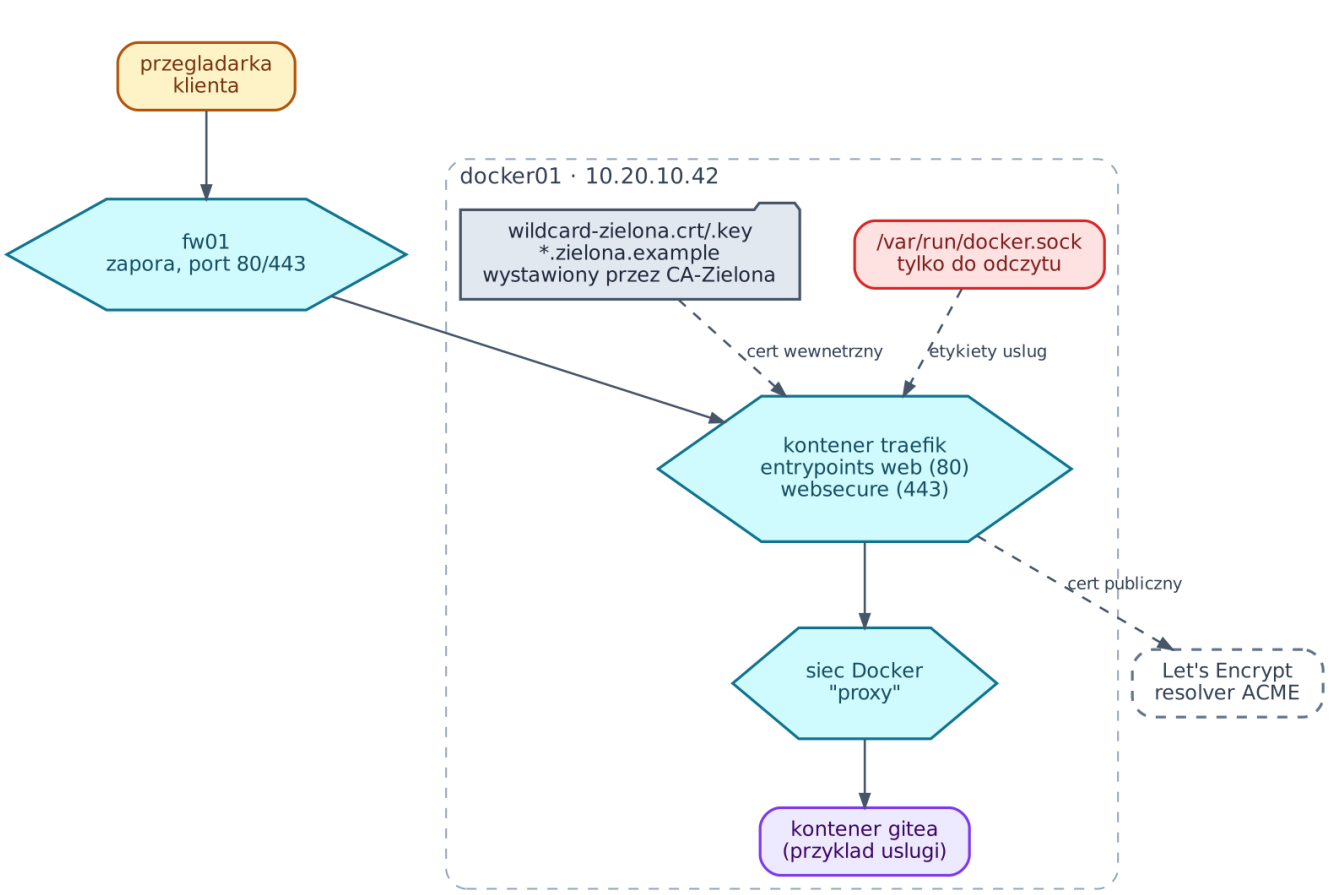
Wersja obrazu `traefik:v3.5.0` sprawdzona na stronie projektu Traefik 28 września 2026. Przy kolejnej instalacji sprawdzamy aktualną wersję w dokumentacji, zanim wpiszemy ją do `.env`.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: reverse proxy. Serwer stojący przed jedną albo wieloma aplikacjami, który dla klienta wygląda jak cel żądania, a w rzeczywistości przekazuje je dalej do właściwej usługi i zwraca klientowi jej odpowiedź. Pełny opis mechanizmu, nagłówków `X-Forwarded-*` i równoważenia obciążenia jest na stronie Reverse proxy i równoważenie obciążenia w książce Nginx: dotyczy Traefika tak samo jak Nginksa, różni się tylko sposób konfiguracji.

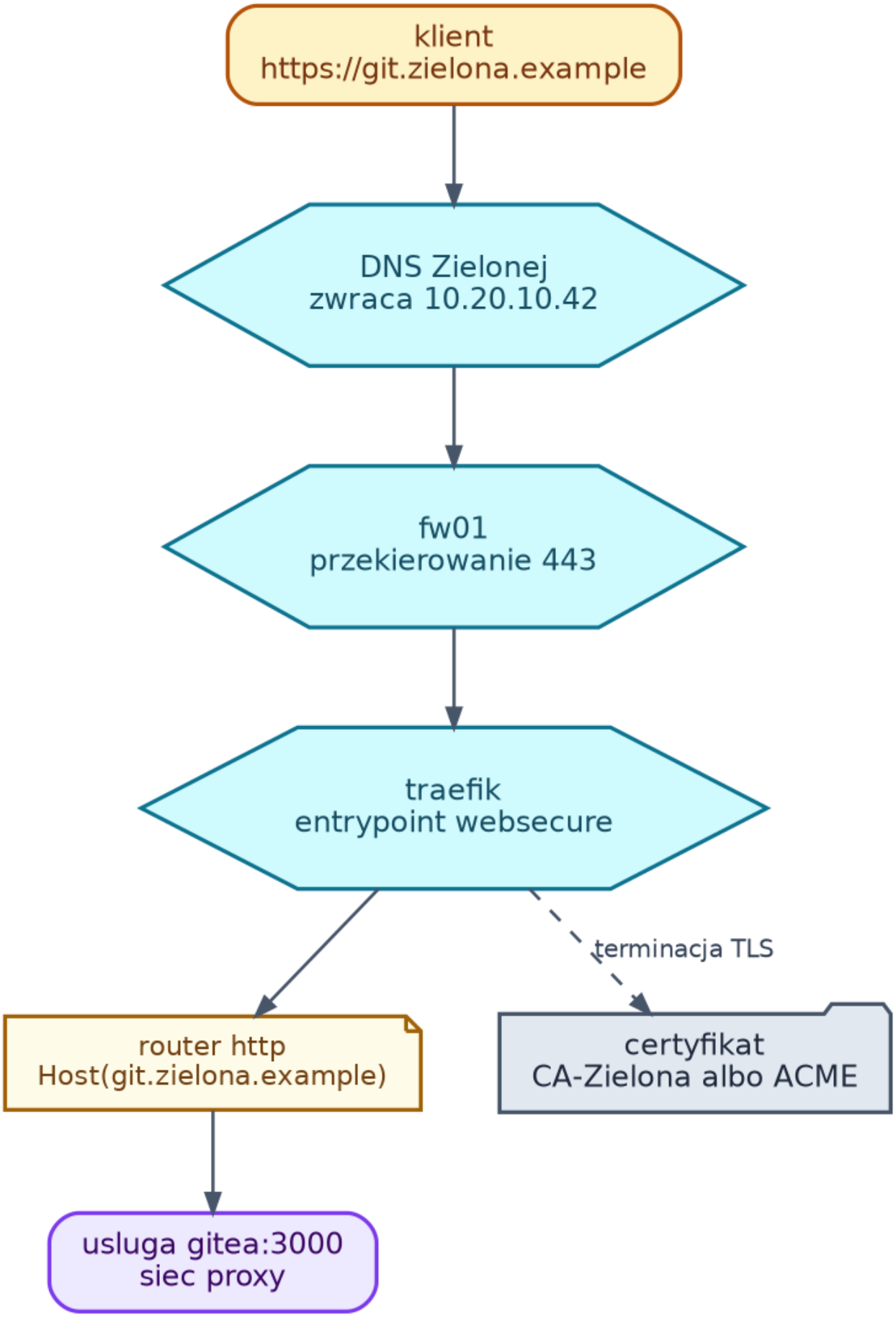
Pojęcie: Traefik. Reverse proxy stworzony z myślą o kontenerach: zamiast osobnego pliku konfiguracyjnego na każdą aplikację, czyta etykiety (`labels`) wprost z definicji kontenera w `compose.yaml` i sam dokłada albo usuwa trasę, gdy kontener wstaje albo znika. W Zielonej stoi przed każdą aplikacją webową na `docker01`, zastępując osobne wpisy w Nginksie opisane na stronie Bloki server i location.



Traefik na docker01 z siecią proxy, certyfikatem serwera wystawionym przez CA-Zielona i resolverem ACME dla adresów publicznych (kliknięcie otwiera pełny rozmiar)

Traefik czyta etykiety kontenerów przez gniazdo Dockera `docker.sock`, montowane w trybie tylko do odczytu, więc widzi każdy kontener podłączony do wspólnej sieci `proxy` bez osobnej rejestracji. Certyfikat wewnętrzny `wildcard-zielona.crt`, wystawiony przez urząd `CA-Zielona`, trafia do Traefika jako statyczny plik przez konfigurację dynamiczną, certyfikaty publiczne Traefik pobiera sam z Let's Encrypt przez protokół ACME.

Którędy idzie żądanie


klient
https://git.zielona.example
DNS Zielonej
zwraca 10.20.10.42
fw01
przekierowanie 443
traefik
entrypoint websecure
terminacja TLS
router http
Host(git.zielona.example)
certyfikat
CA-Zielona albo ACME
usluga gitea:3000
siec proxy

Żądanie klienta trafia przez DNS i fw01 do Traefika, który kończy TLS i przekazuje ruch do usługi po nazwie w sieci proxy (kliknięcie otwiera pełny rozmiar)

Klient rozwiązuje `git.zielona.example` na adres `docker01`, `fw01` przepuszcza ruch na port 443 dalej do hosta, a Traefik na podstawie nagłówka `Host` z żądania dopasowuje router i przekazuje ruch do usługi `gitea` w sieci `proxy`, po nazwie usługi, nie po adresie IP kontenera.

Przebieg

1. Założenie katalogów i wspólnej sieci

```
sudo mkdir -p /docker/apps/traefik /docker/compose/traefik
```

```
docker network create proxy
```

> **Pojęcie: sieć Docker współdzielona między projektami.** Sieć `docker compose` zakłada domyślnie własną, prywatną dla jednego katalogu compose. Sieć `proxy` zakładamy raz, ręcznie, poza jakimkolwiek pojedynczym projektem, żeby każdy kolejny `compose.yaml` w Zielonej mógł się do niej podłączyć jako sieć zewnętrzną i wymienić ruch z Traefikiem, mimo że każda aplikacja ma osobny katalog i osobny cykl życia.

Przykład:

```
docker01:~$ docker network ls | grep proxy
f6e5d4c3b2a1   proxy         bridge    local
```

2. Plik .env z hasłem dashboardu i mailem ACME

```
sudo vim /docker/compose/traefik/.env
```

```
TRAEFIK_DASHBOARD_AUTH=admin:$$apr1$$xyz123$$przykladowyhaszapr1zHasla
ACME_EMAIL=it@zielona.example
```

> (gdzie `TRAEFIK_DASHBOARD_AUTH` to login i hasz basic auth wygenerowany komendą niżej, a znak `$` w pliku `.env` zdublowany na `$$`, bo Compose inaczej próbowałby podstawić go jako zmienną)

Hasz basic auth wygenerowany osobno:

```
openssl passwd -apr1 HasloDashboardu
```

Przykład:

```
$apr1$xyz123$przykladowyhaszapr1zHasla
```

```
sudo chmod 600 /docker/compose/traefik/.env
```

3. Certyfikat serwera od CA-Zielona w katalogu danych

```
sudo mkdir -p /docker/apps/traefik/certy
sudo cp wildcard-zielona.crt wildcard-zielona.key /docker/apps/traefik/certy/
sudo chmod 600 /docker/apps/traefik/certy/wildcard-zielona.key
```

Kopiujemy parę plików: certyfikat serwera dla `*.zielona.example` i jego klucz prywatny. Klucz i żądanie podpisania generujemy OpenSSL-em na `docker01`, żądanie wysyłamy do urzędu `CA-Zielona` na `dc01` z szablonem `ZielonaWWW`, a z powrotem dostajemy tylko podpisany certyfikat, tak jak na stronie Generowanie certyfikatów: OpenSSL i urząd certyfikacji Windows.

> Klucza prywatnego urzędu `CA-Zielona` nigdy nie kopiujemy poza urząd: kto ma ten klucz, wystawi certyfikat na dowolną nazwę w firmie, także na kontroler domeny. Na `docker01` trafiają wyłącznie certyfikat serwera i jego własny klucz, a sam certyfikat urzędu (`CA-Zielona.crt`, bez klucza) jest potrzebny tylko klientom, które mają mu ufać.

4. Konfiguracja dynamiczna: certyfikat wewnętrzny i dashboard

```
sudo vim /docker/apps/traefik/dynamic.yaml
```

```
tls:
  certificates:
    - certFile: /certy/wildcard-zielona.crt
      keyFile: /certy/wildcard-zielona.key
  stores:
    default:
      defaultCertificate:
        certFile: /certy/wildcard-zielona.crt
        keyFile: /certy/wildcard-zielona.key
```

Plik konfiguracji dynamicznej wskazuje Traefikowi certyfikat, którego ma użyć domyślnie dla adresów bez osobnego resolvera ACME, czyli dla wszystkich wewnętrznych adresów `*.zielona.example` obsługiwanych certyfikatem `wildcard-zielona.crt` od urzędu `CA-Zielona`.

5. Plik compose.yaml

```
sudo vim /docker/compose/traefik/compose.yaml
```

```
services:
  traefik:
    image: traefik:v3.5.0
    container_name: traefik
    restart: unless-stopped
    command:
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--providers.docker.network=proxy"
      - "--providers.file.filename=/dynamic.yaml"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--entrypoints.web.http.redirections.entrypoint.to=websecure"
      - "--entrypoints.web.http.redirections.entrypoint.scheme=https"
      - "--certificatesresolvers.letsencrypt.acme.email=${ACME_EMAIL}"
      - "--certificatesresolvers.letsencrypt.acme.storage=/certy/acme.json"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoint=web"
      - "--api.dashboard=true"
    ports:
      - "80:80"
      - "443:443"
```

```
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /docker/apps/traefik/certy:/certy
      - /docker/apps/traefik/dynamic.yaml:/dynamic.yaml:ro
    networks:
      - proxy
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.dashboard.rule=Host(`traefik.zielona.example`)"
      - "traefik.http.routers.dashboard.entrypoints=websecure"
      - "traefik.http.routers.dashboard.tls=true"
      - "traefik.http.routers.dashboard.service=api@internal"
      - "traefik.http.routers.dashboard.middlewares=dashboard-auth"
      - "traefik.http.middlewares.dashboard-auth.basicauth.users=${TRAEFIK_DASHBOARD_AUTH}"

networks:
  proxy:
    external: true
```

Dwa porty publikujemy bezpośrednio na wszystkich interfejsach hosta, w odróżnieniu od każdej innej aplikacji w tej książce: Traefik to jedyny kontener na `docker01`, który faktycznie ma być osiągalny z sieci biurowej i z internetu na porcie 80 i 443, cała reszta chowa się za nim na `127.0.0.1`.

> Na `docker01` stoi też MailCow, który domyślnie również chce portów 80 i 443. W Zielonej te dwa porty należą do Traefika: panel WWW MailCow dostaje w `mailcow.conf` porty `HTTP_PORT=8080` i `HTTPS_PORT=8443` pod adresem `127.0.0.1` (jak w dokumentacji MailCow dla pracy za reverse proxy) i jest podpięty do sieci `proxy` jak każda inna aplikacja. Porty pocztowe (25, 465, 587, 143, 993 i pozostałe) MailCow wystawia sam bezpośrednio na hoście, bo to nie HTTP. Szczegóły w `docker-compose.override.yml` i o certyfikacie są na stronie Instalacja MailCow na docker01 od zera.

> Plik `acme.json`, w którym Traefik trzyma pobrane certyfikaty Let's Encrypt razem z kluczami prywatnymi, musi mieć prawa `600`. Sam Traefik ustawia je przy pierwszym zapisie, ale zakładamy pusty plik z góry, żeby uniknąć okna, w którym leży z domyślnymi prawami `644`.

```
sudo touch /docker/apps/traefik/certy/acme.json
sudo chmod 600 /docker/apps/traefik/certy/acme.json
```

6. Pierwsze uruchomienie

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

Przykład:

```
[+] Running 2/2
 ✔ Container traefik   Started
```

7. Podłączenie przykładowej usługi: Gitea

Istniejący `compose.yaml` Gitea, opisany na stronie Instalacja Gitea w Dockerze, dostaje dopisaną sieć `proxy` i etykiety routingu zamiast publikowania portu bezpośrednio:

```
services:
  gitea:
    # ... reszta definicji bez zmian, patrz strona instalacji Gitea
    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
```

Sekcję `ports` z dotychczasowego pliku usuwamy: ruch do Gitea odtąd wchodzi wyłącznie przez Traefika, port aplikacji przestaje być osiągalny bezpośrednio z hosta.

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

Weryfikacja

```
docker compose logs traefik --tail=20
```

Przykład:

```
traefik  | time="2026-09-28T11:20:03Z" level=info msg="Router gitea@docker created"
```

```
curl -Ik https://git.zielona.example
```

Przykład:

```
HTTP/2 200
server: Caddy
```

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: dashboard Traefika pod `https://traefik.zielona.example`, zakładka „HTTP Routers”, widoczny router `gitea` ze statusem zielonym „Enabled” i przypisanym certyfikatem.

Wycofanie zmian

Zatrzymanie Traefika przywraca wszystkie aplikacje ukryte za nim do stanu nieosiągalnego z zewnątrz, dopóki ich `compose.yaml` znowu nie opublikuje portu bezpośrednio:

```
cd /docker/compose/traefik
docker compose down
```

Powrót do bezpośredniego publikowania portu w Gitea, jako tymczasowy krok wycofania:

```
ports:
  - "127.0.0.1:3000:3000"
```

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
`curl` do aplikacji zwraca `404 page not found` od samego Traefika	brak albo błędna reguła `Host()` w etykietach kontenera	`docker compose logs traefik`, sprawdzamy dashboard, zakładka „HTTP Routers”
Aplikacja widoczna w dashboardzie, ale `curl` zwraca `502 Bad Gateway`	kontener aplikacji nie jest w sieci `proxy` albo zły numer portu w `loadbalancer.server.port`	sprawdzamy `docker network inspect proxy`, poprawiamy etykiety
Certyfikat Let's Encrypt nie zostaje wydany, w logu `unable to obtain ACME certificate`	port 80 niedostępny z internetu w momencie żądania challenge	sprawdzamy przekierowanie portu 80 na `fw01`, próbujemy ponownie po usunięciu wpisu z `acme.json`
Dashboard Traefika odpowiada bez pytania o hasło	etykieta `dashboard-auth` nie została dołączona do routera dashboardu	sprawdzamy literówkę w nazwie middleware między dwiema etykietami

Traefik kontra Nginx na hoście kontra Nginx Proxy Manager

Kiedy co	Nginx na hoście	Traefik	Nginx Proxy Manager
Liczba aplikacji za proxy	kilka, rzadko się zmieniają	wiele, dochodzą i znikają często	kilka do kilkunastu, administrowane przez osoby bez wprawy w CLI
Sposób dodania trasy	ręczna edycja pliku `server` i `reload`	etykiety w `compose.yaml` samej aplikacji, bez dotykania proxy	klikanie w panelu WWW
Certyfikaty Let's Encrypt	ręcznie przez `certbot`, opisane na stronie TLS w Nginx	automatycznie przez wbudowany resolver ACME	automatycznie przez panel
Pasuje do Zielonej	serwer plików, RDS, aplikacje spoza Dockera	`docker01`, rosnąca liczba kontenerów	brak w Zielonej, dobra opcja dla małej firmy bez własnego Nginksa

Traefika wybieramy w Zielonej właśnie dlatego, że `docker01` ciągle rośnie: piąta, dziesiąta i dwudziesta aplikacja dostaje trasę tym samym mechanizmem etykiet, bez ręcznej pracy przy samym proxy. Nginx na hoście zostaje tam, gdzie i tak trzeba obsłużyć coś spoza Dockera. Nginx Proxy Manager rozwiązuje ten sam problem co Traefik, ale przez panel klikany myszą zamiast plików w repozytorium konfiguracji: dobry wybór dla mniejszej firmy bez zespołu, który i tak trzyma wszystko w Gicie.

Częste błędy

Objaw	Przyczyna	Co robimy
Aplikacja niewidoczna w dashboardzie mimo działającego kontenera	brak etykiety `traefik.enable=true`, bo `exposedbydefault=false` wymaga jej jawnie	dopisujemy etykietę, `docker compose up -d`
Dwie aplikacje próbują użyć tego samego adresu `Host()`	literówka w skopiowanej regule albo faktyczny konflikt nazw	`docker compose logs traefik`, szukamy wpisu o nadpisanym routerze
Certyfikat wewnętrzny nie ładuje się, przeglądarka pokazuje domyślny certyfikat Traefika	plik `dynamic.yaml` nie jest wczytany przez flagę `--providers.file.filename`	sprawdzamy `docker compose logs traefik` pod kątem błędu wczytania pliku
Po dodaniu sieci `proxy` aplikacja przestaje widzieć własną bazę danych	usunięta przez pomyłkę sieć `default` z sekcji `networks` usługi	kontener musi być podłączony do obu sieci naraz, nie tylko do `proxy`

□□ Ćwiczenia

1. Podłączamy drugą aplikację, na przykład NetBox, do Traefika: piszemy komplet etykiet `traefik.*` i sieć `proxy` w jej `compose.yaml`.
2. Wyjaśniamy, dlaczego port 80 zostaje otwarty na Traefiku nawet wtedy, gdy cały ruch ma iść po HTTPS.

Odpowiedź

Port 80 służy do dwóch rzeczy naraz: przekierowania klienta na HTTPS przez `redirections.entrypoint` i do odebrania wyzwania ACME `httpchallenge` od Let's Encrypt, które z definicji przychodzi po zwykłym HTTP. Zamknięcie portu 80 zepsułoby wydawanie nowych certyfikatów publicznych.

1. Dashboard Traefika odpowiada bez pytania o hasło, mimo ustawionego `TRAEFIK_DASHBOARD_AUTH` w `.env`. Wskazujemy dwie rzeczy do sprawdzenia w pierwszej kolejności.

Odpowiedź

Po pierwsze etykietę `traefik.http.routers.dashboard.middlewares`, czy w ogóle wskazuje nazwę middleware `dashboard-auth`. Po drugie zawartość zmiennej `TRAEFIK_DASHBOARD_AUTH` w `.env`, czy znak `$` z haszu `apr1` jest zdublowany na `$$`, inaczej Compose próbuje go podstawić jako własną zmienną i hasło wychodzi puste albo błędne.

Powiązane: Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Reverse proxy i równoważenie obciążenia · TLS w Nginx · Portainer: Docker w przeglądarce

Źródła: doc.traefik.io, rozdział „Docker Provider" · doc.traefik.io, rozdział „ACME (Let's Encrypt)" · hub.docker.com, obraz `traefik`

Revision #23
Created 2026-09-28 17:15:45 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:32 UTC by Tomasz Konieczny