1. Versión y redes

version: '3.8'

- Define el formato de Compose que estamos usando (3.8).

networks:

traefik-net:

driver: bridge

name: traefik-net

attachable: true

- Creamos la red **traefik-net** de tipo bridge.
- attachable: true permite que contenedores externos (o comandos manuales) se conecten a ella.
- Todos los servicios la usan para comunicarse internamente.

---

2. Servicio reverse-proxy (Traefik)

reverse-proxy:

image: traefik:v2.5

container_name: traefik

ports:

- "80:80"

- "8080:8080" # Dashboard

volumes:

- /var/run/docker.sock:/var/run/docker.sock

- ./traefik/traefik.yml:/etc/traefik/traefik.yml

labels:

- "traefik.enable=true"

- "traefik.http.routers.dashboard.rule=Host(`traefik.localhost`) && (PathPrefix(`/api`) || PathPrefix(`/dashboard`))"

- "traefik.http.routers.dashboard.service=api@internal"

- "traefik.http.routers.dashboard.entrypoints=web"

networks:

- traefik-net

restart: unless-stopped

1. **Imagen y puertos**:
    - Usa traefik:v2.5.
    - Mapea el host 80→80 para tráfico HTTP y 8080→8080 para el dashboard.

2. **Volúmenes**:
    - Monta el socket Docker para que Traefik descubra contenedores dinámicamente.
    - Monta tu configuración estática en /etc/traefik/traefik.yml.
3. **Labels para el dashboard**:
    - traefik.enable=true: habilita Traefik en este contenedor.
    - Define un router llamado dashboard que responde cuando el host es traefik.localhost y la ruta comienza con /api o /dashboard.
    - Usa el servicio interno api@internal de Traefik para exponer el dashboard.
    - Lo ata al entryPoint web (puerto 80).

---

3. Servicio service1

```
service1:
  build:
    context: ./service1
    dockerfile: Dockerfile
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.service1.entrypoints=web"
    - "traefik.http.routers.service1.rule=PathPrefix(`/service1`)"
    - "traefik.http.routers.service1.middlewares=strip-service1,retry"
    - "traefik.http.middlewares.strip-service1.stripprefix.prefixes=/service1"
    - "traefik.http.middlewares.retry.retry.attempts=5"
    - "traefik.http.services.service1.loadbalancer.server.port=8000"
    - "traefik.http.services.service1.loadbalancer.healthcheck.path=/health"
    - "traefik.http.services.service1.loadbalancer.healthcheck.interval=10s"
  networks:
    - traefik-net
  restart: unless-stopped
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
    interval: 30s
    timeout: 10s
    retries: 3
```

1. **Build**:
    - Construye la imagen a partir de la carpeta ./service1 usando su Dockerfile.

2. **Router**:
    - entrypoints=web asegura que atienda en el puerto 80.
    - rule=PathPrefix(/service1) indica que cualquier URL que empiece con /service1 se enrutará aquí.
3. **Middlewares** (única línea combinada):
    - strip-service1: elimina el prefijo /service1 antes de pasar la petición a FastAPI.
    - retry: reintenta hasta 5 veces si falla la conexión.
4. **Definición de middlewares**:
    - stripPrefix.prefixes=/service1 para especificar qué prefijo quitar.
    - retry.attempts=5 para configurar el middleware de reintentos.
5. **Servicio y healthcheck**:
    - El load balancer interno apunta al puerto 8000 del contenedor.
    - Healthcheck interno en /health cada 10 segundos para determinar si el contenedor está “healthy”.

---

4. Servicio service2

La configuración de service2 es análoga a la de service1, cambiando únicamente los nombres:

```
  service2:
    build:
      context: ./service2
      dockerfile: Dockerfile
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.service2.entrypoints=web"
      - "traefik.http.routers.service2.rule=PathPrefix(`/service2`)"
      - "traefik.http.routers.service2.middlewares=strip-service2,retry"
      - "traefik.http.middlewares.strip-service2.stripprefix.prefixes=/service2"
      - "traefik.http.middlewares.retry.retry.attempts=5"
      - "traefik.http.services.service2.loadbalancer.server.port=8000"
      - "traefik.http.services.service2.loadbalancer.healthcheck.path=/health"
      - "traefik.http.services.service2.loadbalancer.healthcheck.interval=10s"
    networks:
      - traefik-net
    restart: unless-stopped
    healthcheck:
```

test: ["CMD", "curl", "-f", "http://localhost:8000/health"]

interval: 30s

timeout: 10s

retries: 3

¿Qué consigue esta configuración?

1. **Prefijo limpio**: Traefik quita /service1 o /service2, evitando 404 de FastAPI.
2. **Alta disponibilidad**: Middleware retry para reintentos automáticos.
3. **Healthchecks**: Solo expone servicios “healthy” al balanceador.
4. **Separación clara**: Cada microservicio con su propio router, middleware y healthcheck.

Con esto, cuando hagas:

curl -v http://localhost/service1/

Traefik quita /service1 y reenvía a http://service1:8000/, devolviéndote tu mensaje de FastAPI.