ChatGPT

Guia Completo: Configurando o Traefik com Docker e Let's Encrypt (Swarm ou Standalone)

Introdução

Traefik é um proxy reverso moderno e **balanceador de carga** projetado para ambientes nativos em nuvem. Ele facilita o roteamento de tráfego HTTP/S para múltiplos serviços de forma dinâmica, integrando-se com **Docker** e outras plataformas de orquestração. Em cenários de desenvolvimento ou produção com vários containers, o Traefik permite expor todos os serviços através de um único ponto de entrada (porta), encaminhando requisições para cada serviço conforme o **host** ou **caminho** da URL solicitada [1]. Isso elimina a necessidade de lembrar múltiplas portas ou lidar com problemas como *CORS*, pois o Traefik centraliza o acesso HTTP e HTTPS a todos os containers.

Traefik destaca-se por integrar-se automaticamente à sua infraestrutura. No caso do Docker, ele consegue **descobrir novos containers** e configurar rotas automaticamente via *labels*. Ou seja, em vez de escrever manualmente blocos de configuração para cada serviço, basta adicionar labels apropriadas nos containers Docker que o Traefik lê essas configurações e passa a rotear o tráfego para eles [2]. Essa abordagem dinâmica simplifica bastante a implantação de aplicações com muitos microsserviços.

Outro benefício importante é que o Traefik possui **suporte nativo ao Let's Encrypt**. Ele pode obter e renovar automaticamente certificados TLS gratuitos para seus domínios, provendo HTTPS para seus serviços sem configuração manual de certificados. De fato, o Traefik foi projetado para tornar a implementação de HTTPS transparente – ele gerencia os desafios ACME do Let's Encrypt e armazena os certificados obtidos, servindo-os para as respectivas rotas de forma automática.

Para ilustrar o poder dessa combinação: o site *sre.rs* (um blog de DevOps) roda inteiramente em um container Docker, publicado através de um proxy Traefik com provedor Docker, utilizando TLS automático do Let's Encrypt e CI/CD para implantações [3]. Isso mostra como Traefik + Docker + Let's Encrypt formam uma pilha robusta que automatiza a exposição de serviços web de maneira segura e escalável.

Neste guia, aprenderemos passo a passo como **configurar o Traefik com Docker**, seja em modo standalone (Docker Compose ou container individual) ou em um cluster Docker Swarm. Cobriremos a criação de redes internas/externas, configuração de regras de roteamento via labels, integração com o Let's Encrypt para certificados SSL, e cuidados de segurança (como o uso de um *socket proxy* do Docker). Ao final, você terá um ambiente funcional em que poderá adicionar novos containers e ter seu tráfego roteado automaticamente pelo Traefik, com **HTTP->HTTPS** e certificados gerenciados de forma transparente.

Visão Geral do Traefik com Docker

Arquitetura simplificada: Traefik operando como proxy reverso em frente a múltiplos containers. Ele escuta requisições no host (porta 80/443) e as encaminha internamente para os serviços Docker corretos com base no hostname ou path [1] [4].

Para que o Traefik consiga rotear o tráfego para os containers, é necessário que ele **monitore a API do Docker**. Geralmente isso é feito montando o *socket* do Docker dentro do container do Traefik. Dessa forma, o Traefik escuta eventos do Docker (subida de containers, remoções, etc.) e atualiza sua configuração dinamicamente [2]. Cada container que deve ser roteado precisa estar acessível ao Traefik – em termos práticos, **Traefik e os containers de aplicação precisam compartilhar uma rede Docker comum** [5]. Ao ficarem na mesma rede, o Traefik pode acessar os serviços pelo nome do container ou IP interno.

A **configuração do Traefik se divide em duas partes**: estática e dinâmica. A *configuração estática* define como o Traefik em si é executado – por exemplo, em quais portas ele escuta (chamadas de *EntryPoints*), quais provedores de configuração serão usados (por ex., Docker, arquivos, etc.), parâmetros de TLS default, middlewares globais, etc. Essa configuração pode ser feita via arquivo (YAML/TOML) ou parâmetros de linha de comando. Já a *configuração dinâmica* envolve as rotas, serviços e middlewares – isto é, as regras que dizem "o host X vai para tal serviço", "aplique tal redirecionamento", etc. No caso do Docker, a configuração dinâmica vem das **labels** nos containers, que o Traefik interpreta em tempo real [2]. Alternativamente, pode-se usar um provider de arquivo (arquivo de configuração dinâmica separado), mas usar as labels do Docker tende a ser mais simples em ambientes containerizados, pois tudo fica definido junto do *deploy* de cada serviço.

Em resumo, o fluxo de funcionamento é: o Traefik inicia expondo certas portas (ex.: 80 e 443) – esses são seus *entrypoints*. Ele se conecta ao Docker (socket) e descobre containers com determinadas labels. Para cada container habilitado, o Traefik cria roteadores e serviços internos: um *router* casa uma regra (ex.: host ou path) e está vinculado a um *service* que aponta para o container (endereço IP interno e porta). Quando uma requisição chega, o Traefik seleciona o router cuja regra combina com aquele host/path e encaminha para o serviço alvo (basicamente, faz um proxy HTTP até o container destino).

Vamos agora colocar tudo isso em prática, configurando passo a passo.

Preparando o Ambiente e as Redes Docker

Antes de subir o Traefik, certifique-se de ter alguns pré-requisitos atendidos:

- **Docker instalado** (p. ex., Docker Engine ou Docker Desktop) e, opcionalmente, Docker Compose para facilitar a definição em YAML [6].
- **Um domínio público** apontando para o IP do seu servidor (caso vá utilizar Let's Encrypt para HTTPS) [6].
- **Portas 80 e 443 liberadas** para acesso externo (e redirecionadas para seu host Docker, se estiver atrás de um roteador) [7], pois o Traefik usará essas portas para receber HTTP e HTTPS.

Com isso em ordem, o primeiro passo é criar uma **rede Docker dedicada** para o Traefik e os serviços. Recomendamos criar uma rede do tipo bridge (ou overlay, se for Swarm) para isolamento. Por exemplo, vamos criar uma rede chamada `proxy`:

```
docker network create traefik_proxy
```

No Docker Swarm, você criaria uma overlay:

```
docker network create -d overlay --attachable traefik_proxy
```

Essa rede será utilizada tanto pelo container do Traefik quanto pelos containers dos serviços que ele irá proxyar. Assim, todos os serviços acessíveis pelo Traefik devem estar **conectados a essa mesma rede** (no Compose isso se define em `networks`). No arquivo de configuração estática do Traefik, podemos até restringir para que ele só considere containers naquela rede (veremos adiante).

Criando o container do Traefik: Podemos usar **Docker Compose** para facilitar. Crie um diretório (ex: `traefik/`) e dentro dele um arquivo `docker-compose.yml` com conteúdo similar a este:

```
services:
  traefik:
    image: traefik:v3.1    # versão do Traefik
    container_name: traefik
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true   # segurança: remove privilégios extras
    ports:
      - "80:80"    # entryPoint web (HTTP)
      - "443:443"  # entryPoint websecure (HTTPS)
      - "8080:8080"  # (opcional) Dashboard do Traefik
    networks:
      - proxy    # conecta à rede criada para traefik e serviços
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro   # acesso ao Docker
(somente-leitura)
      - ./traefik.yml:/traefik.yml:ro   # arquivo de config estática do
Traefik
      - traefik-certs:/certs   # volume para armazenar certificados
(acme.json)
volumes:
  traefik-certs:
    name: traefik-certs   # volume nomeado para persistir os certificados
networks:
  proxy:
    name: traefik_proxy   # utiliza a rede criada anteriormente
```

No trecho acima, mapeamos as portas 80 e 443 do host para dentro do container (essas serão os pontos de entrada externos). Também mapeamos a porta 8080, que corresponde ao **dashboard web** do Traefik – essa interface é opcional e deve ser protegida (por padrão vem desabilitada ou em modo inseguro, então cuidado ao usá-la aberta). Montamos o arquivo `traefik.yml` (que escreveremos já já) como somente-leitura dentro do container, e montamos o socket Docker em modo *read-only* (`:ro`) para minimizar riscos de segurança. O volume `traefik-certs` servirá para o Traefik guardar dados persistentes, principalmente o arquivo `acme.json` que contém os certificados TLS obtidos do Let's Encrypt [8]. Definimos também a rede `proxy` para o serviço Traefik, garantindo que ele estará na rede correta para falar com os outros containers.

> **Dica:** A opção `exposedByDefault` do provider Docker (que veremos adiante) faz com que nenhum container seja publicado se não tiver labels Traefik explícitas [9]. Isso aumenta a segurança, evitando expor acidentalmente serviços não configurados. No compose acima, definimos `security_opt: no-new-privileges` para o container

Traefik, o que previne escalonamento de privilégios no container (boa prática de segurança). Também montamos o socket Docker em modo somente leitura para que o Traefik não possa modificar nada no host Docker, apenas ler eventos [10].

Feito o compose (ou alternativamente, você poderia usar `docker run` com parâmetros equivalentes [11]), vamos criar agora o arquivo estático de configuração do Traefik.

Configuração Estática do Traefik (traefik.yml)

No mesmo diretório, crie o arquivo `traefik.yml` que definirá os parâmetros estáticos do Traefik. Este arquivo controla os entrypoints, o provedor Docker e o resolver do Let's Encrypt, entre outras configurações globais:

```
api:
  dashboard: true
  insecure: true        # não use "insecure" em produção sem protegê-lo!
entryPoints:
  web:
    address: ":80"
    http:
      redirections:
        entryPoint:
          to: websecure
          scheme: https
  websecure:
    address: ":443"
providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false       # só publica containers com
traefik.enable=true
    network: traefik_proxy        # força uso da rede específica para
comunicação
certificatesResolvers:
  letsencrypt:
    acme:
      email: seu-email@provedor.com
      storage: /certs/acme.json
      caServer: https://acme-staging-v02.api.letsencrypt.org/directory
# CA de teste (staging)
      # caServer: https://acme-v02.api.letsencrypt.org/directory        # CA
produção (descomente para usar produção)
      httpChallenge:
        entryPoint: web
```

Vamos entender as principais diretivas acima:

- **api.dashboard** – habilita o dashboard web (em modo inseguro neste exemplo, ou seja, sem autenticação). Em produção, recomenda-se configurar algum tipo de autenticação ou definir `insecure: false` e acessar o dashboard somente via rede interna segura.
- **entryPoints** – definimos dois entrypoints: `web` na porta 80 (HTTP) e `websecure` na 443 (HTTPS). Também configuramos um redirecionamento automático de HTTP para HTTPS [12] [13], ou seja, qualquer requisição que chegar na porta 80 será redirecionada para a porta 443 com o mesmo host/path, promovendo uso de TLS sempre. (Esse bloco de redirecionamento é opcional, mas altamente recomendado para forçar HTTPS).
- **providers.docker** – habilita o provedor Docker, indicando ao Traefik para ler configurações dos containers Docker em execução. Usamos `endpoint: unix:///var/run/docker.sock` para apontar para o socket local do Docker (conforme montado). Marcamos `exposedByDefault: false` para não publicar containers sem labels explícitas [9]. E com `network: traefik_proxy`, instruímos o Traefik a **sempre usar essa rede** para conectar nos containers (caso um container esteja em múltiplas redes, essa configuração garante que ele tente o IP da rede correta). Essa rede deve ser o nome da rede interna do Traefik (no Compose usamos o alias `frontend` ou `proxy` – ajuste conforme seu caso) [14] [15].
- **certificatesResolvers** – aqui definimos um resolver ACME chamado `letsencrypt` (você pode escolher outro nome). Dentro dele, configuramos o **ACME** do Let's Encrypt:
- `email` – seu email para cadastro na ACME (importante para recuperar chave de conta e notificações de expiração).
- `storage` – o caminho do arquivo onde o Traefik salvará os certificados obtidos. Estamos usando `/certs/acme.json`, que dentro do container é um arquivo no volume que montamos (`traefik-certs`). Assim, mesmo recriando o container, seus certificados persistem.
- `caServer` – especifica a URL da API ACME. Por padrão, o Traefik usaria o endpoint de produção do Let's Encrypt, mas no exemplo acima mostramos o uso do **servidor de staging** (linha marcada) [16]. Recomendamos iniciar sempre testando com o ambiente de staging do Let's Encrypt, pois ele tem limites de rate mais altos e emite certificados não confiáveis (para teste). Após confirmar que tudo funciona, você troca para o servidor de produção (descomentando a linha correta) e remove o arquivo acme.json de staging para que o Traefik obtenha os certificados reais [17] [18].
- `httpChallenge` – escolhemos o desafio do tipo HTTP. Informamos que o entrypoint de desafio é o **web (porta 80)** [19]. Isso significa que, para validar um domínio, o Traefik vai responder na porta 80 com um token de confirmação para o Let's Encrypt. **Importante:** certifique-se que porta 80 do Traefik esteja acessível externamente e que o domínio a ser certificado aponte para o IP correto, caso contrário o desafio HTTP-01 irá falhar. (Alternativamente, o Traefik suporta DNS challenge – não mostrado aqui – ou TLS-ALPN challenge. No exemplo usamos o método simples via HTTP.)

Com o `docker-compose.yml` e o `traefik.yml` prontos, podemos **subir o Traefik**:

```
docker compose up -d
```

(Se estiver usando Swarm com um stack file, seria `docker stack deploy -c docker-compose.yml traefik`*.)*

Isso irá baixar a imagem do Traefik e iniciar o container. Você pode verificar os logs do Traefik para ver se ele subiu corretamente: `docker logs -f traefik`. Caso tudo tenha ocorrido bem, o Traefik estará escutando nas portas definidas. Se você habilitou o dashboard (porta 8080 inseguro), acesse

`http://<seu-servidor>:8080` para ver a interface do Traefik – inicialmente ela mostrará zero routers e zero services, já que ainda não configuramos nenhum serviço para publicar.

Publicando Serviços com o Traefik (Configuração Dinâmica via Labels)

Com o Traefik rodando, vamos **expor containers através dele**. Como mencionado, o Traefik inspeciona os containers Docker em busca de labels específicas. Para que um container seja roteado:

- Ele **deve estar conectado** à mesma rede que o Traefik (ex.: `traefik_proxy`). Sem isso, o Traefik não conseguirá alcançá-lo.
- Deve ter a label `traefik.enable=true` para ser considerado (já que definimos `exposedByDefault: false`) [20].
- Deve ter ao menos uma regra de roteamento, geralmente via label `traefik.http.routers.<nome>.rule=...` definindo host ou path. Por exemplo, usar a regra de **Host** permite direcionar pelo domínio usado na requisição.
- Precisamos também indicar qual porta interna do container corresponde ao serviço web. Podemos fazer isso de duas formas: expondo a porta no Docker (não confundir com publicar, pode ser só `EXPOSE` no Dockerfile ou `ports` no Compose sem mapear para host), e o Traefik pode detectar automaticamente; ou explicitamente via label `traefik.http.services.<nome>.loadbalancer.server.port=<porta_interna>` [21]. É comum usar a label explicitamente para evitar ambiguidades e não precisar expor portas no host.
- Por fim, para habilitar TLS via Traefik/Let's Encrypt, configuramos labels relacionadas a TLS no *router*: `traefik.http.routers.<nome>.tls=true` para indicar que aquele router usa TLS, e `traefik.http.routers.<nome>.tls.certresolver=letsencrypt` para vincular ao resolver ACME configurado (no caso, nomeamos de "letsencrypt") [20]. Assim, se o domínio ainda não tiver um certificado gerado, o Traefik automaticamente irá solicitar um ao Let's Encrypt quando a primeira requisição ocorrer (ou no startup dependendo da versão/config). Também podemos especificar domínios/SANs manualmente via labels (útil para wildcard, etc.), mas em geral não é necessário – o Traefik usará o host da regra como nome do certificado.

Vamos a um **exemplo prático**: suponha que queremos publicar um serviço web simples (poderia ser um whoami ou uma aplicação web qualquer). Usaremos um container de exemplo `traefik/whoami` (retorna informações da requisição). Criaremos um arquivo Compose para este serviço como ilustração:

```
services:
  whoami:
    image: traefik/whoami
    container_name: whoami-test
    networks:
      - proxy
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.whoami.rule=Host(`testesuatraefik.com`)"
      - "traefik.http.routers.whoami.entrypoints=websecure"
      - "traefik.http.routers.whoami.tls=true"
      - "traefik.http.routers.whoami.tls.certresolver=letsencrypt"
      - "traefik.http.services.whoami.loadbalancer.server.port=80"
```

```
networks:
  proxy:
    external: true
    name: traefik_proxy
```

Analisando as labels definidas acima:

- **traefik.enable=true**: habilita este container para o Traefik (senão ele ignoraria).
- **traefik.http.routers.whoami.rule=Host(**`testesuatraefik.com`**)**: define uma regra do tipo Host. Isso significa que só atenderemos tráfego cujo host HTTP seja `testesuatraefik.com` (você substituiria pelo seu domínio). Note que é possível usar múltiplos hosts ou mesmo regras de Path, conforme a sintaxe de regras do Traefik [21]. Aqui escolhemos Host porque é o mais comum para separar serviços por subdomínio.
- **traefik.http.routers.whoami.entrypoints=websecure**: vincula o router `whoami` ao entryPoint `websecure` (ou seja, porta 443). Opcionalmente, poderíamos também atender em `web` (porta 80) se quiséssemos permitir HTTP – nesse caso, poderíamos listar ambos: `entrypoints=web,websecure`. Mas como configuramos redirecionamento global de 80->443, geralmente definimos apenas websecure mesmo.
- **traefik.http.routers.whoami.tls=true**: indica que esse router requer TLS (certificado). Isso faz com que o Traefik sirva o tráfego usando HTTPS. Se um navegador tentar http://testesuatraefik.com, ele será redirecionado para https:// automaticamente pelo redirecionamento que ativamos.
- **traefik.http.routers.whoami.tls.certresolver=letsencrypt**: aponta que o resolver ACME a usar para esse router é o *letsencrypt* (nome que definimos no traefik.yml). Com isso, se ainda não houver cert para este domínio, o Traefik vai executar o desafio ACME e obter o certificado, armazenando em acme.json. Na próxima vez, ele já terá o cert armazenado e apenas o renovará quando necessário.
- **traefik.http.services.whoami.loadbalancer.server.port=80**: como nosso container `traefik/whoami` escuta na porta 80 interna, estamos especificando essa porta. Repare que **não expusemos essa porta 80 no host** (não há `ports:` no compose do whoami). Não precisamos – o Traefik acessa internamente via rede Docker. Essa label garante que ele sabe em qual porta do container conectar [22]. (Se não puséssemos, o Traefik tentaria descobrir sozinho portas *exposed* pelo container, mas é uma boa prática sempre explicitar).

Após subir esse serviço (via `docker compose up -d` no seu arquivo do whoami ou integrado no mesmo compose do Traefik), se tudo estiver correto, ao acessar o domínio configurado no seu navegador, você deve ser atendido pelo serviço interno. Por exemplo: `https://testesuatraefik.com` deveria mostrar a página do whoami. O Traefik terá obtido um certificado do Let's Encrypt para *testesuatraefik.com* automaticamente – você pode verificar no dashboard ou nos logs mensagens do ACME.

Labels adicionais úteis: no exemplo acima, usamos o mínimo necessário. Há várias outras labels que podem ser usadas conforme a necessidade – por exemplo, middlewares como autenticação, compressão, whitelists de IP etc. Um exemplo citado na comunidade é adicionar uma *middleware* de whitelist de IPs por arquivo e referenciá-la na rota [23]. Você também pode definir redirecionamentos específicos, reescrita de path, roteamento por PathPrefix, entre outros. Para manter o guia focado, não entraremos em todos esses detalhes, mas a [documentação oficial do Traefik lista todas as labels disponíveis e regras de roteamento suportadas [2] [21]].

Um ponto a observar: **nome do router e service nas labels** – usamos `whoami` como identificador em `routers.whoami` e `services.whoami`. Você pode usar qualquer identificador alfanumérico em vez de `whoami` (como o nome do seu serviço). Apenas certifique-se de usar o *mesmo* nome em todas as labels relacionadas ao mesmo router/service. No docker-compose acima isso já está consistente (`whoami` em todas). Em Docker Swarm, as labels podem ser aplicadas no nível do serviço e o Traefik costuma prefixar internamente com o nome do stack, mas você pode manter nomes explícitos para evitar confusão.

Traefik em ação: Depois de adicionar alguns serviços com suas labels, o dashboard do Traefik (se habilitado) exibirá os routers ativos, cada um associado a um entrypoint e regra, apontando para um serviço (container) específico, com indicação se TLS está ativo e qual resolver/cert usado. É uma boa forma de visualizar e depurar a configuração.

Separando Serviços em Redes/EntryPoints Internos vs Externos

Em alguns cenários avançados, pode-se desejar que certos serviços fiquem acessíveis **somente em rede interna** (por exemplo, disponíveis apenas via VPN ou rede local), enquanto outros são expostos publicamente. O Traefik permite implementar isso de algumas formas.

Usando entrypoints distintos: Uma estratégia é definir entrypoints separados para tráfego interno e externo, possivelmente em portas diferentes ou usando hostnames diferentes. Por exemplo, você poderia ter no `traefik.yml` entrypoints como:

```
entryPoints:
  web-int:
    address: ":80"
  websecure-int:
    address: ":443"
  web-ext:
    address: ":81"
  websecure-ext:
    address: ":444"
```

Nesse caso, poderíamos expor o Traefik também nas portas 81/444 e configurar o roteador de borda (firewall/roteador) para encaminhar portas 80->81 e 443->444 para o seu host, assim os entrypoints “externos” seriam alcançáveis de fora. Enquanto isso, os entrypoints “internos” usam as portas padrão 80/443, porém como não há port-forwarding dessas portas no roteador para o host (ou são bloqueadas a acessos externos), elas só seriam acessíveis por clientes na LAN ou VPN. Esse é exatamente o arranjo descrito em uma documentação de homelab [24] [25]: utilizar portas não padrão para serviços externos e portas padrão para internos, isolando via DNS/roteamento de modo que usuários externos nem consigam resolver os domínios internos.

No Traefik, ao usar múltiplos entrypoints, você pode escolher **em qual entrypoint cada serviço vai operar**. Isso é feito pela label `...routers.<nome>.entrypoints`. No exemplo da documentação, o autor define que certos serviços usem `entrypoints=websecure-int` (interno) ou `websecure-ext` (externo) conforme o caso [26]. Assim, mesmo que dois serviços tenham domínios diferentes, ele consegue especificar que um só responde no Traefik interno e outro no externo. Naturalmente, os clientes externos não acessariam o internal entrypoint (pois não há rota/porta aberta), então aquele serviço efetivamente fica restrito.

Outra forma de segmentar é através de **DNS e resolução interna**: por exemplo, você poderia usar domínios `.lan` ou subdomínios específicos que só existem no DNS interno, e ainda assim usar o mesmo Traefik. Clientes externos nem resolveriam esses nomes, ou resolveriam para IPs privados. Enquanto isso, Traefik serve todos na mesma porta, mas de fato apenas usuários na rede interna conseguiriam acessar determinados hosts. No entanto, em ambientes complexos, separar entrypoints pode dar mais controle (pode aplicar configurações TLS diferentes, por exemplo certificados internos diferentes, etc.).

Para a maioria dos usuários iniciantes, não é necessário complicar com entrypoints múltiplos. Mas tenha em mente que **é possível isolar serviços**. A recomendação mais simples: se um serviço não deve ser público, não aponte nenhum domínio público para ele e não o documente publicamente. Você pode consumir via IP local ou via DNS interno. E, é claro, não coloque o Traefik em nenhuma rede que seja exposta para esse serviço (ou use autenticação/Middleware no Traefik para proteger). No contexto do Docker, uma prática comum é **colocar serviços sensíveis em networks distintas e não conectá-los ao Traefik**, de modo que eles não fiquem acessíveis externamente de forma alguma (ex: um banco de dados permanece apenas na rede interna do stack da aplicação, sem nenhuma label Traefik).

Em Docker Swarm, como nota, costuma-se criar uma rede overlay única (às vezes chamada `traefik-public` ou similar) que é attachable. Todos os serviços que precisam ser publicados conectam-se a ela e adicionam labels de Traefik. Serviços que não devem ser expostos externamente simplesmente não se conectam a essa rede, ficando apenas em redes privadas de seus respectivos stacks [27] [28]. Assim, Traefik nem sequer vê esses containers.

Resumindo: **Traefik oferece flexibilidade para ambientes com diferentes níveis de exposição**. Você pode rodar uma única instância atendendo tudo e usar DNS/regras para controlar acesso, ou rodar instâncias separadas de Traefik (uma para público, outra para interno), ou usar entrypoints separados em uma instância. A escolha depende da complexidade desejada. Para aprofundamento, a documentação de homelab do XMS Systems detalha como configurar entrypoints internos/externos e associá-los via labels [29] [26].

Executando o Traefik no Docker Swarm

O Docker Swarm permite orquestrar containers em um cluster. Felizmente, **usar Traefik no Swarm** não é muito diferente de usá-lo com Docker Compose. A maioria das configurações que discutimos se aplicam igualmente. Existem, contudo, alguns pontos de atenção:

- **Deploy do Traefik no Swarm:** Você pode definir o Traefik em um arquivo de stack (compose versão 3.x). Geralmente, recomenda-se rodar apenas uma instância do Traefik (replicas: 1) ou rodá-lo em modo global somente em nós manager. Isso porque, ao usar Let's Encrypt, múltiplas instâncias do Traefik poderiam tentar obter certificados simultaneamente e bater nos limites do Let's Encrypt. É possível rodar Traefik em alta disponibilidade compartilhando o arquivo acme.json em um storage comum, ou usando um backend KV (Consul/Etcd) para armazenar certificados, mas isso adiciona complexidade. Para iniciantes no Swarm, mantenha 1 réplica para evitar problemas de sincronização de certificados.

- **Networks no Swarm:** Crie uma rede overlay para o Traefik conforme mencionado (ex: `traefik_proxy` ou `traefik-public`). Marque-a como attachable para que serviços externos ao stack principal consigam conectar-se a ela. Ao definir o serviço Traefik no stack, referencie essa rede (use `external: true` se ela for criada fora do compose). Todos os serviços de outros stacks que precisarem do Traefik devem também referenciar essa rede

externa e ter as labels. O Traefik no Swarm enxerga containers de todos os stacks/namespaces contanto que compartilhem a rede e o provedor Docker esteja ativado. Você pode filtrar por `swarmCluster=true` nas config do Traefik v3 se necessário, mas não entraremos nesse detalhe.

- **Labels em serviços:** No Swarm, as labels do Traefik devem ser colocadas na definição do serviço (no compose under deploy.labels, ou diretamente em labels do serviço se usar versão <3.8). O Traefik vai ler as labels de serviços Swarm do mesmo jeito que faz com containers individuais.

- **Atualizações e rollback:** Quando atualizar a versão do Traefik no Swarm, fique atento para possivelmente não perder o volume de certificados. Use volumes nomeados ou mapeie para algum path em todos os nós (por exemplo, NFS) se quiser alta disponibilidade. Uma abordagem simples é restringir o Traefik a rodar sempre no mesmo nó (com `placement` constraints) e usar um volume local nesse nó para acme.json.

Há um guia excelente mantido pela comunidade que cobre **Traefik no Docker Swarm passo a passo**, incluindo configuração de HTTPS, middlewares, etc. Você pode consultar o *AeonEros Wiki - Traefik Reverse Proxy for Docker Swarm* [30] para se aprofundar nesses detalhes específicos de Swarm. Esse guia aborda desde o *getting started* até TLS via Let's Encrypt no Swarm [31] [32], sendo uma ótima referência para casos de uso mais avançados em produção.

Considerações de Segurança e Melhores Práticas

Ao expor vários serviços através de um proxy, devemos nos atentar a alguns pontos de segurança:

1. Proteção do Docker Socket: Montar o socket do Docker dentro de qualquer container (seja Traefik ou outro) equivale a dar **acesso root** ao host Docker, pois quem acessa o socket pode controlar containers e inclusive escapar para o host. Embora o Traefik em si seja projetado para interagir com o socket de forma controlada, é sempre uma superfície de ataque importante. Uma solução recomendada é usar um **Docker Socket Proxy**, como o projeto Tecnativa/docker-socket-proxy [33]. Esse proxy atua como intermediário, expondo o socket Docker em um endereço TCP interno e **filtrando as requisições** – por exemplo, permitindo apenas operações de leitura (listar containers, eventos) e bloqueando operações de escrita/perigosas. Assim, você pode montar esse proxy no Traefik em vez do socket bruto. O proxy retorna erro 403 Forbidden para chamadas não autorizadas [34], prevenindo que mesmo que alguém comprometa o Traefik, não consiga escalar privilégios facilmente.

Para usar o socket proxy, você rodaria um container separado (`tecnativa/docker-socket-proxy`) conectado na mesma rede do Traefik e montando `/var/run/docker.sock`. Configura algumas variáveis de ambiente no proxy para autorizar somente endpoints GET necessários (containers, infos, etc.) [35]. Então, no Traefik, em vez de montar o socket, você aponta o provider Docker para o endpoint TCP do proxy (por ex.: `tcp://socket-proxy:2375`). Desse modo, Traefik faz chamadas HTTP restritas ao proxy, e o proxy repassa apenas o que é seguro ao Docker. Lembre-se de **nunca expor a porta do socket proxy fora da host** – coloque-o em uma rede *internal* junto com Traefik, sem publish de porta [36]. Documentações como *"How to use Traefik without exposing the Docker socket"* mostram o passo a passo dessa configuração, e o README do projeto também traz recomendações de segurança.

2. Protegendo o Dashboard: Por padrão, o Traefik v3 não expõe o dashboard a menos que você ative (no nosso config ativamos com `insecure: true` temporariamente para facilidade). Em produção, **evite deixar o dashboard acessível publicamente sem proteção**. Você pode ou desativá-lo completamente, ou habilitar algum middleware de autenticação. Por exemplo, pode criar um

middleware de autenticação básica (via arquivo estático ou label) e anexar ao router do dashboard. Outra opção é bindar a porta 8080 somente em localhost em vez de 0.0.0.0, e acessar via túnel SSH quando precisar. O importante é que o dashboard revela informações sobre seus serviços e pode potencialmente ser usado para alterar configurações se alguém obter acesso (no v3 talvez nem permita alterar, mas de todo modo, é sensível).

3. Atualizações e versão do Traefik: A equipe do Traefik (Traefik Labs) está em constante melhoria; verifique a documentação para a versão específica que você usar. A sintaxe de configuração v2 -> v3 mudou levemente em algumas seções. Este guia foca em Traefik 3.x (no Traefik 2.x é muito parecido, com pequenas diferenças nos static/dynamic file keys). Sempre teste mudanças em ambiente de homologação antes de aplicar em produção, especialmente ao lidar com certificados (para não esgotar limite do Let's Encrypt).

4. Middlewares e Rate Limits: Considere utilizar recursos do Traefik como middlewares de segurança: *rate limit* (limitar requisições para evitar brute force), *circuit breakers* ou *retries* para tornar serviços mais resilientes, e *IP whitelist* para áreas administrativas das suas aplicações. Essas configurações podem ser definidas via labels ou arquivo dinâmico. A documentação oficial e o catálogo de middlewares do Traefik têm exemplos prontos.

5. Logging e monitoramento: Por padrão o Traefik faz log no stdout. É útil aumentar o nível de log para *DEBUG* temporariamente se algo não funcionar, pois ele mostra as regras carregadas, etc. Em produção, mantenha em INFO ou WARN para não lotar logs. Você também pode ativar o *access log* do Traefik para registrar todas as requisições roteadas – isso ajuda em auditoria, depuração de erros 404, etc. O access log pode ser rotacionado; o wiki AeonEros tem dicas de log rotation para Traefik [37].

Conclusão e Referências

Configuramos com sucesso um ambiente com Traefik proxy reverso integrando com Docker. Recapitulando, o Traefik ficou escutando nas portas 80/443 do host e encaminhando para serviços docker internos conforme as regras definidas nas labels dos containers. Habilitamos TLS automático via Let's Encrypt – agora, sempre que adicionarmos um novo serviço com um host válido, o Traefik obterá e instalará um certificado automaticamente (facilitando imensamente a adoção de HTTPS em todas as aplicações). Essa abordagem **elimina configurações manuais de virtual hosts** e certificados em cada aplicação; o Traefik centraliza isso de forma declarativa.

Para aprender mais e se aprofundar em cenários avançados, recomendamos as seguintes leituras:

- **Documentação oficial do Traefik:** especialmente a seção de Docker Provider e exemplos de Docker Compose [38], e a referência de configurações (routers, services, middlewares). Há também um guia oficial de *HTTP Routing* no docs.docker.com com outro tutorial de Traefik [39] [5].
- **Guia Traefik no Docker Swarm (AeonEros Wiki):** um guia comunitário completo cobrindo desde conceitos iniciais até HTTPS no Swarm, com exemplos de configuração passo-a-passo [30] [32].
- **Exemplos práticos de Compose:** o repositório *Haxxnet Compose-Examples* traz arquivos de compose prontos para diversas aplicações usando Traefik [23] – é ótimo para ver na prática como ficam as labels de diferentes casos de uso.
- **Curso Real World DevOps (Predmijat):** os materiais do curso (repositório `predmijat/realworlddevopscourse`) incluem configuração do Traefik junto a outras ferramentas (CI/CD, etc.), mostrando um caso de uso do mundo real de DevOps com Traefik integrado [3].

- **Blog Sven van Ginkel:** série *Traefik Essentials* com tutoriais de configuração do Traefik v3, incluindo uso de Cloudflare DNS challenge, wildcard certs, autenticação, etc., disponível no Medium e no site pessoal do autor [40] [41].

Seguindo este guia e explorando as referências, você estará apto a configurar um **ambiente de proxy reverso robusto** com Traefik, Docker e Let's Encrypt – seja para projetos pessoais em um *homelab* ou mesmo em ambientes de produção em escala moderada. Aproveite a facilidade que o Traefik traz para a arquitetura de microsserviços e boa implantação!

Referências Utilizadas: Docker Docs (Traefik Guide) [2] [5], Medium/@svenvanginkel (Traefik v3 + LetsEncrypt) [8] [22], XMS Systems Docs (EntryPoints) [29] [26], Reddit/r/Traefik Comunidade [23], Tecnativa Docker Socket Proxy (GitHub) [33] [34], AeonEros Wiki (Traefik Swarm Guide) [31] [32], entre outros. Cada link citado ao longo do texto fornece mais detalhes sobre os tópicos discutidos – recomendamos sua leitura para aprofundamento. Boa sorte com sua configuração Traefik!

---

[1] [2] [4] [5] [11] [21] [39] HTTP routing with Traefik | Docker Docs
https://docs.docker.com/guides/traefik/

[3] sre.rs
https://sre.rs

[6] [7] [8] [9] [10] [12] [13] [14] [15] [17] [18] [19] [20] [22] [40] [41] Setting up Traefik v3 with LetsEncrypt in Docker | Medium
https://medium.com/@svenvanginkel/setting-up-traefik-v3-in-docker-0c0559a696f1

[16] Traefik Essentials Reverse Proxy with Docker & Let’s Encrypt | Sven van Ginkel
https://svenvg.com/posts/traefik-essentials-reverse-proxy-with-docker-lets-encrypt/

[23] [27] [28] The Ultimate Guide to Setting Up Traefik : r/Traefik
https://www.reddit.com/r/Traefik/comments/1hzjk4d/the_ultimate_guide_to_setting_up_traefik/

[24] [25] [26] [29] Traefik - Entry Points - XMS Docs
https://docs.xmsystems.co.uk/entrypoints/

[30] [31] [32] [37] [38] New 2025 Docker/-Swarm Beginners-Guide for Traefik Reverse Proxy! : r/Traefik
https://www.reddit.com/r/Traefik/comments/1imzn8a/new_2025_dockerswarm_beginnersguide_for_traefik/

[33] [34] [35] [36] GitHub - Tecnativa/docker-socket-proxy: Proxy over your Docker socket to restrict which requests it accepts
https://github.com/Tecnativa/docker-socket-proxy