

Traefik

Traefik, c'est quoi ?

Traefik est un reverse proxy conçu pour Docker. Il reçoit tout le trafic web d'une VM, lit les étiquettes (labels) posées sur les conteneurs, et en déduit seul vers quelle application envoyer chaque requête. Il obtient aussi les certificats HTTPS et les renouvelle automatiquement.

Élément	Rôle
Point d'entrée <code>web</code> (port 80)	Reçoit le HTTP et redirige vers HTTPS ; sert au défi de validation Let's Encrypt
Point d'entrée <code>websecure</code> (port 443)	Reçoit le HTTPS et applique le certificat correspondant au nom demandé
Fournisseur Docker	Surveille les conteneurs et crée les routes à partir de leurs labels
Résolveur de certificats	Demande et renouvelle les certificats Let's Encrypt

Visiteur → `cloud.alpha.nebuleuse.example:443` → **Traefik** → **conteneur Nextcloud**

Concrètement, ajouter une application ne demande jamais de modifier Traefik : il suffit que son conteneur soit sur le réseau `proxy` et porte les bons labels. Traefik le détecte à chaud, sans redémarrage.

Prérequis

1. Le socle est en place : Docker installé, réseau `proxy` créé, ports 80 et 443 ouverts dans le pare-feu.
2. Un enregistrement DNS pointe chaque nom du client vers l'adresse publique de sa VM. Une entrée générique `*.alpha.nebuleuse.example` suffit.
3. Le port 80 est joignable depuis Internet : Let's Encrypt s'en sert pour vérifier que le domaine appartient bien au client.
4. Le domaine est réel. `nebuleuse.example` étant fictif, aucun certificat réel ne peut lui être délivré. Pour reproduire ce TP, remplacer le domaine par un domaine dont on dispose.

Note — Pendant les essais, activer le mode « staging » de Let's Encrypt (variable `traefik_acme_staging`). Les certificats délivrés ne sont pas reconnus par les navigateurs, mais le serveur de test n'impose pas les mêmes limites de demandes que le serveur réel, qui bloque un domaine après quelques échecs répétés.

Arborescence du rôle

```
roles/traefik/  
├─ defaults/main.yml  
├─ handlers/main.yml  
├─ tasks/main.yml  
└─ templates/  
    ├─ docker-compose.yml.j2  
    ├─ logrotate-traefik.j2  
    └─ traefik.yml.j2
```

Sur la VM, le rôle crée :

```
/opt/nebuleuse/traefik/  
├─ docker-compose.yml  
├─ traefik.yml          # configuration statique  
└─ acme/acme.json      # certificats (créé par Traefik, accès réservé à root)  
  
/var/log/traefik/access.log  # journal d'accès, lu par Fail2ban
```

Variables

```
# roles/traefik/defaults/main.yml  
  
# Version de Traefik. Choisir une version récente : avec les versions récentes de Docker,  
# une version trop ancienne de Traefik ne parvient plus à lire l'API Docker.  
traefik_version: "v3.6"  
  
# Dossier de travail sur la VM.  
traefik_dir: /opt/nebuleuse/traefik  
  
# Dossier du journal d'accès (monté dans le conteneur, lu par Fail2ban).
```

```
traefik_log_dir: /var/log/traefik

# Adresse e-mail déclarée auprès de Let's Encrypt (définie dans group_vars/all).
traefik_acme_email: "{{ nebuleuse_acme_email }}"

# true : utilise le serveur de test de Let's Encrypt (certificats non reconnus, sans limites strictes).
traefik_acme_staging: false

# Réseau Docker partagé avec les applications (défini dans group_vars/all).
traefik_network: "{{ docker_proxy_network }}"

# Noms de domaine que les applications doivent joindre à l'intérieur de la VM.
# Voir la section « Joindre le SSO depuis un conteneur ».
traefik_network_aliases:
  - "{{ client_subdomains.sso }}.{{ client_domain }}"
```

La configuration statique

La configuration statique décrit Traefik lui-même : ses ports, ses sources de configuration, ses certificats. Elle n'est lue qu'au démarrage.

```
# roles/traefik/templates/traefik.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.

global:
  # Pas de vérification de mise à jour ni de télémétrie : la VM n'a pas à contacter ces services.
  checkNewVersion: false
  sendAnonymousUsage: false

log:
  level: INFO

# Journal d'accès : une ligne JSON par requête. Fail2ban le lit pour repérer les attaques.
accessLog:
  filePath: /var/log/traefik/access.log
  format: json
```

```
providers:
  docker:
    # Un conteneur n'est publié que s'il porte le label "traefik.enable=true".
    exposedByDefault: false
    # Réseau utilisé pour joindre les conteneurs.
    network: {{ traefik_network }}

entryPoints:
  web:
    address: ":80"
    http:
      # Toute requête HTTP est redirigée vers HTTPS.
      redirections:
        entryPoint:
          to: websecure
          scheme: https
  websecure:
    address: ":443"
    http:
      tls:
        # Tout routeur sur ce point d'entrée reçoit automatiquement un certificat
        # pour le nom de sa règle Host(...).
        certResolver: letsencrypt

certificatesResolvers:
  letsencrypt:
    acme:
      email: {{ traefik_acme_email }}
      # Emplacement des certificats : volume persistant, pour ne pas les redemander à chaque
      redémarrage.
      storage: /acme/acme.json
{% if traefik_acme_staging %}
      # Serveur de test de Let's Encrypt.
      caServer: https://acme-staging-v02.api.letsencrypt.org/directory
{% endif %}
      # Validation par HTTP : Let's Encrypt vérifie le domaine sur le port 80.
      httpChallenge:
        entryPoint: web
```

Note — Cette configuration obtient un certificat par nom de domaine. Pour un certificat unique couvrant tous les sous-domaines (certificat « wildcard »), il faut passer par la validation DNS, qui demande les identifiants de l'API du fournisseur de noms de domaine. C'est plus pratique, mais cela ajoute un secret par client.

Le fichier Docker Compose

```
# roles/traefik/templates/docker-compose.yml.j2
# Fichier g  r   par Ansible : toute modification manuelle sera   cras  e.
services:
  traefik:
    image: traefik:{{ traefik_version }}
    container_name: {{ client_name }}_traefik
    restart: unless-stopped
    # Interdit au processus de gagner de nouveaux privil  ges.
    security_opt:
      - no-new-privileges:true
    # Seul conteneur de la VM    publier des ports vers Internet.
    ports:
      - "80:80"
      - "443:443"
    volumes:
      # Lecture de la liste des conteneurs. L'acc  s est en lecture seule, mais il donne
      # tout de m  me une vue   tendue sur Docker : voir « Pour aller plus loin ».
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - {{ traefik_dir }}/traefik.yml:/etc/traefik/traefik.yml:ro
      - {{ traefik_dir }}/acme:/acme
      - {{ traefik_log_dir }}:/var/log/traefik
    networks:
      {{ traefik_network }}:
        # Noms que les autres conteneurs peuvent joindre directement, sans sortir de la VM.
        aliases:
{% for alias in traefik_network_aliases %}
          - {{ alias }}
{% endfor %}

networks:
```

```
{{ traefik_network }}:
  # Réseau créé par le rôle docker-engine : Compose ne doit pas le recréer.
  external: true
```

Rotation du journal d'accès

Un journal qui grossit sans limite finit par remplir le disque. Docker fait tourner les journaux de ses conteneurs, mais pas les fichiers que Traefik écrit lui-même. Une règle `logrotate` s'en charge.

```
# roles/traefik/templates/logrotate-traefik.j2
{{ traefik_log_dir }}/access.log {
  # Un fichier par jour, sept jours conservés.
  daily
  rotate 7
  compress
  delaycompress
  missingok
  notifempty
  # Copie puis vide le fichier : Traefik continue d'écrire dans le même fichier ouvert.
  copytruncate
}
```

Tâches

```
# roles/traefik/tasks/main.yml

- name: Créer les dossiers de Traefik
  ansible.builtin.file:
    path: "{{ item.path }}"
    state: directory
    mode: "{{ item.mode }}"
  loop:
    - { path: "{{ traefik_dir }}",      mode: "0755" }
    # Le dossier des certificats est réservé à root : il contient les clés privées.
    - { path: "{{ traefik_dir }}/acme", mode: "0700" }
    - { path: "{{ traefik_log_dir }}",  mode: "0755" }
```

```

- name: Déposer la configuration statique de Traefik
  ansible.builtin.template:
    src: traefik.yml.j2
    dest: "{{ traefik_dir }}/traefik.yml"
    mode: "0644"
  # Un changement de configuration statique demande un redémarrage.
  notify: Redémarrer traefik

- name: Déposer le fichier Docker Compose de Traefik
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ traefik_dir }}/docker-compose.yml"
    mode: "0644"

- name: Déposer la règle de rotation du journal d'accès
  ansible.builtin.template:
    src: logrotate-traefik.j2
    dest: /etc/logrotate.d/traefik
    mode: "0644"

- name: Démarrer Traefik
  community.docker.docker_compose_v2:
    project_src: "{{ traefik_dir }}"
    # "present" crée ou met à jour les conteneurs, sans toucher à ceux déjà conformes.
    state: present

- name: Attendre que Traefik écoute sur le port 443
  # Le module s'exécute sur la VM : "127.0.0.1" désigne donc la VM elle-même.
  ansible.builtin.wait_for:
    host: 127.0.0.1
    port: 443
    timeout: 60

```

```

# roles/traefik/handlers/main.yml
- name: Redémarrer traefik
  community.docker.docker_compose_v2:
    project_src: "{{ traefik_dir }}"
    state: restarted

```

Comment une application se déclare

Traefik n'a aucune liste d'applications : chaque conteneur se décrit lui-même avec des labels. Toutes les pages d'application suivent ce modèle, ici pour Nextcloud.

```
# Extrait d'un docker-compose.yml.j2 d'application (exemple : Nextcloud)
labels:
  # Publier ce conteneur (obligatoire, car exposedByDefault vaut false).
  - "traefik.enable=true"
  # Réseau à utiliser si le conteneur est sur plusieurs réseaux.
  - "traefik.docker.network={{ docker_proxy_network }}"
  # Règle de routage : les requêtes pour ce nom vont vers ce conteneur.
  # Le nom du routeur (nextcloud-alpha) doit être unique sur la VM.
  - "traefik.http.routers.nextcloud-{{ client_name }}.rule=Host(`{{
client_subdomains.nextcloud }}.{{ client_domain }}`)\"
  # Point d'entrée HTTPS : le certificat est demandé automatiquement.
  - "traefik.http.routers.nextcloud-{{ client_name }}.entrypoints=websecure"
  # Port d'écoute de l'application À L'INTÉRIEUR du conteneur.
  - "traefik.http.services.nextcloud-{{ client_name }}.loadbalancer.server.port=80"
```

Joindre le SSO depuis un conteneur

Une application doit parfois contacter le SSO de la VM, par exemple pour échanger un code de connexion contre un jeton. Elle le fait avec son adresse publique, `sso.alpha.nebuleuse.example`. Or ce nom pointe vers l'adresse publique de la VM, et un conteneur qui cherche à la joindre fait un détour par l'extérieur : le chemin fonctionne ou non selon la façon dont l'hébergeur attache l'adresse publique à la machine (adresse portée directement par la VM ou traduite par une passerelle).

Le plus fiable est d'éviter ce détour. Le rôle donne à Traefik un **alias** de réseau : sur le réseau `proxy`, le nom `sso.alpha.nebuleuse.example` est résolu vers le conteneur Traefik lui-même. L'application joint donc Traefik directement, sur le port 443, avec le même nom et le même certificat. Le résultat est le même pour l'utilisateur, sans dépendre de l'adresse publique.

```
# Extrait de docker-compose.yml.j2 (généré par le rôle)
networks:
```

```
proxy:
  aliases:
    - sso.alpha.nebuleuse.example
```

Seul le nom du SSO a besoin de cet alias, car c'est le seul que les applications appellent entre elles. La variable `traefik_network_aliases` permet d'en ajouter d'autres si une application doit en joindre une autre.

Vérification

```
# Sur la VM : Traefik est démarré et publie les ports 80 et 443.
docker ps --filter name=traefik --format '{{.Names}}\t{{.Status}}\t{{.Ports}}'

# La redirection HTTP vers HTTPS répond "301".
curl -sI http://127.0.0.1 | head -n 1

# Un nom sans application répond "404" : Traefik écoute, mais n'a aucune route pour lui.
curl -skI https://127.0.0.1 | head -n 1

# Journal de démarrage : aucune erreur de certificat attendue.
docker logs --tail 30 alpha_traefik

# Journal d'accès : le champ ClientHost doit contenir l'adresse du visiteur, pas celle de Docker.
sudo tail -n 3 /var/log/traefik/access.log | jq '{ClientHost, RequestHost, DownstreamStatus}'
```

Une fois une application déployée, le certificat apparaît avec :

```
# Affiche l'émetteur et les dates de validité du certificat de l'application.
echo | openssl s_client -connect cloud.alpha.nebuleuse.example:443 -servername
cloud.alpha.nebuleuse.example 2>/dev/null | openssl x509 -noout -issuer -dates
```

Pour aller plus loin

- **Protéger l'accès au socket Docker.** Même en lecture seule, le socket expose beaucoup d'informations sur la machine. Un intermédiaire comme `docker-socket-proxy` limite Traefik aux seules requêtes nécessaires.

- **Certificats wildcard** par validation DNS, pour ne pas dépendre de l'accès au port 80.
- **En-têtes de sécurité** (HSTS, `X-Content-Type-Options`) ajoutés par un middleware commun à toutes les applications.

Pièges

1. **Limites de Let's Encrypt** : des demandes répétées en échec bloquent temporairement un domaine. Tester avec `traefik_acme_staging: true`, puis repasser à `false` en supprimant le contenu du dossier `acme` pour obtenir les vrais certificats.
2. **DNS pas encore propagé** : le défi échoue et le journal indique que le domaine ne pointe pas vers la VM. Vérifier l'enregistrement avant de redémarrer Traefik.
3. **Application inaccessible, erreur 404** : le label `traefik.enable=true` ou la règle `Host(...)` est absente ou erronée. Le routeur n'existe pas, donc Traefik ne sait pas où envoyer la requête.
4. **Application inaccessible, erreur 502 ou 504** : Traefik trouve la route mais n'atteint pas le conteneur. Causes habituelles : mauvais port dans le label `loadbalancer.server.port`, conteneur absent du réseau `proxy`, ou conteneur sur plusieurs réseaux sans label `traefik.docker.network`.
5. **Traefik ne détecte plus aucun conteneur** : avec les versions récentes de Docker, une version trop ancienne de Traefik n'arrive plus à lire l'API Docker. Le journal mentionne une version d'API trop ancienne. Mettre à jour `traefik_version`.
6. **Adresse du visiteur remplacée par celle de la passerelle Docker** : le journal d'accès montre l'adresse interne de Docker au lieu de celle du visiteur, ce qui rend Fail2ban inefficace. Le cas se rencontre surtout avec les connexions IPv6 ou lorsqu'un intermédiaire est placé devant la VM. Le contrôle se fait avec la dernière commande de vérification, depuis une adresse extérieure.
7. **Noms de routeur en double** : deux conteneurs qui déclarent le même nom de routeur se contredisent, et seul l'un des deux fonctionne. Le nom du client dans le nom du routeur évite les collisions entre projets.
8. **Modification de `traefik.yml` sans effet** : la configuration statique n'est lue qu'au démarrage. Le handler redémarre Traefik quand le fichier change ; les labels, eux, sont pris en compte à chaud.

Page suivante : **Authentik (SSO)**.

Revision #1

Created 2026-10-16 20:23:33 UTC by clement-derouet

Updated 2026-10-16 20:29:37 UTC by clement-derouet