

TP : Ajouter un serveur HTTPS dans un réseau Docker et tester la connectivité réseau

Objectifs du TP :

1. Comprendre les bases du protocole HTTPS et des certificats SSL/TLS.
 2. Déployer un serveur web avec HTTPS dans un conteneur Docker.
 3. Générer un certificat SSL auto-signé pour sécuriser les connexions.
 4. Tester l'accès au serveur HTTPS depuis l'hôte et un autre conteneur.
 5. Comprendre la communication entre conteneurs dans un réseau Docker.
 6. Analyser le trafic réseau et inspecter les logs pour vérifier le bon fonctionnement du serveur.
-

Introduction : HTTPS, TLS et certificats SSL

Qu'est-ce que HTTPS ?

HTTPS (HyperText Transfer Protocol Secure) est une version sécurisée de HTTP qui chiffre les échanges entre un client (navigateur, API, etc.) et un serveur. Il repose sur TLS (Transport Layer Security) pour garantir :

Confidentialité : Les données échangées sont chiffrées, empêchant leur interception.

Intégrité : Les données ne peuvent pas être modifiées en transit.

Authentification : Le client peut vérifier l'identité du serveur via un certificat SSL/TLS.

Le rôle de TLS (Transport Layer Security)

TLS est un protocole cryptographique qui assure la sécurisation des communications. Il remplace SSL (Secure Sockets Layer) qui est obsolète.

À quoi sert un certificat SSL/TLS ?

Un certificat SSL/TLS est un fichier qui permet d'établir une connexion sécurisée en prouvant l'identité d'un serveur. Un certificat valide doit être signé par une autorité de certification (CA).

Dans ce TP, nous allons générer un **certificat auto-signé**, ce qui signifie que nous faisons confiance à notre propre certificat au lieu d'une autorité externe. Cela permet d'activer HTTPS en local, mais les navigateurs afficheront un avertissement indiquant qu'il n'est pas issu d'une CA reconnue.


Étape 1 : Création du réseau Docker

Pourquoi créer un réseau Docker ?

Un réseau Docker permet aux conteneurs de communiquer entre eux sans exposer leurs ports à l'extérieur.

Commande :

```
docker network create --driver bridge mon-reseau-https
```

Objectif :  Isoler les conteneurs dans un réseau Docker pour sécuriser leur communication.

Étape 2 : Déploiement du serveur HTTPS avec Nginx

Création d'un répertoire de travail

```
mkdir -p ~/docker/nginx-https  
cd ~/docker/nginx-https
```

Cela permet d'organiser les fichiers nécessaires (configuration Nginx, certificats, Dockerfile).

Génération d'un certificat SSL auto-signé

Commande :

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \  
-keyout nginx.key -out nginx.crt \  
-subj "/C=FR/ST=Paris/L=Paris/O=MonServeur/CN=localhost"
```

Explications :

- x509 : Génère un certificat auto-signé.
- nodes : Ne chiffre pas la clé privée (sinon, Nginx demanderait un mot de passe à chaque redémarrage).
- days 365 : Durée de validité de 1 an.
- newkey rsa:2048 : Génère une clé RSA de 2048 bits.
- subj "/C=FR/.../CN=localhost" : Définit les informations du certificat.

Cela crée :

- nginx.key** : Clé privée du serveur.
- nginx.crt** : Certificat auto-signé.

Création du fichier de configuration Nginx

Commande :

```
nano nginx.conf
```

Contenu du fichier :

```
events {}

http {
    server {
        listen 443 ssl;
        server_name localhost;

        ssl_certificate /etc/nginx/ssl/nginx.crt;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;

        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers on;

        location / {
            root /usr/share/nginx/html;
            index index.html;
        }
    }
}
```

Objectifs : ☒ Activer HTTPS sur le port 443. ☒ Configurer les certificats SSL. ☒ Servir une page web sécurisée. ☒ Sécuriser la configuration TLS.

Étape 3 : Création du conteneur Docker Nginx avec HTTPS

Création du fichier Dockerfile

Commande :

nano Dockerfile

Contenu :

```
FROM nginx:latest
COPY nginx.conf /etc/nginx/nginx.conf
COPY nginx.crt /etc/nginx/ssl/nginx.crt
COPY nginx.key /etc/nginx/ssl/nginx.key
```

Explications :

FROM nginx:latest : Utilise l'image officielle de Nginx.

COPY : Copie la configuration et les certificats dans le conteneur.

Construction et lancement du conteneur

```
docker build -t mon-nginx-https .
docker run -d -p 443:443 --name mon-serveur-https --network mon-reseau-https mon-nginx-https
```

Objectifs : ☒ Lancer un serveur HTTPS accessible sur le port 443. ☒ Intégrer le conteneur dans le réseau Docker.

Étape 6 : Analyse des logs et surveillance du trafic

Affichage des logs du conteneur

`docker logs mon-serveur-https --follow`

Objectifs : ☒ Surveiller en temps réel les requêtes HTTPS reçues. ☒ Détecter d'éventuelles erreurs de configuration.

Surveillance du trafic HTTPS avec tcpdump

Depuis le conteneur :

```
apt update && apt install -y tcpdump
tcpdump -i eth0 port 443 -w trafic_https.pcap
```

Explications :

`tcpdump -i eth0 port 443` : Capture le trafic HTTPS sur l'interface réseau du conteneur.

`-w trafic_https.pcap` : Enregistre les paquets capturés dans un fichier pour analyse ultérieure.

Pour analyser les paquets capturés :

```
apt install -y wireshark
wireshark trafic_https.pcap
```

Objectifs : ☒ Observer le trafic sécurisé. ☒ Vérifier les connexions HTTPS. ☒ Détecter d'éventuelles anomalies.