## Exercice 4 – Apache, modules et SSL

Informations

**Évaluation** : formative

**Type de travail** : individuel

**Durée** : 3 heures

**Système d’exploitation** : Linux Ubuntu

**Environnement** : Docker

Objectifs

Cet exercice a pour objectifs :

* Lancer un conteneur Apache personnalisé
* Faire une configuration avancée : modules et SSL.

Dans cet exercice, vous allez lancer un conteneur Apache personnalisé, installer des modules Apache et créer un site Web sécurisé.

#### Partie 1 : Installation d’Apache

Dans cette partie, vous allez lancer un serveur Web Apache selon les spécifications données.

Étape 1 : Docker-compose

Nous allons utiliser docker-compose pour se créer un conteneur Apache personnalisé. Voici les caractéristiques de ce que nous voulons accomplir :

* + Remplacer le fichier de configuration httpd.conf par le nôtre.
  + Créer un répertoire /srv/htdocs.
  + Créer un point de montage du répertoire actuel au répertoire /srv/htdocs.
  + Installez les logiciels vim (ou nano) et iproute (pour vérification réseau).

Docker-compose nous permet de sauvegarder nos informations d’exécution de notre conteneur dans un fichier facile à lire (yaml). Docker-compose n’est pas un outil de production, mais parfait pour faire du développement local ou pour faire des tests. Voici des informations sur la syntaxe d’un fichier docker-compose.yml.

version: '3.8'  # si aucune version n'est spécifiée, alors v1. Minimum v2 recommandé

services:  # conteneurs. Identique à docker run

  servicename: # un nom convivial. c'est aussi le nom DNS à l'intérieur du réseau

    image: # facultatif si vous utilisez build:

    command: # facultatif, remplace le CMD par défaut spécifié par l'image

    environment: # facultatif, identique à -e dans l'exécution du docker

    volumes: # facultatif, identique à -v dans l'exécution du docker

servicename2:

volumes: # facultatif, identique à docker volume create

networks: # facultatif, identique à docker network create

Si votre projet à un Dockerfile et un docker-compose.yml, vous pouvez lancer votre projet avec :

git clone github.com/moi/MonLogiciel

docker-compose up

1. Nous allons commencer par reproduire la commande de l’exercice 2. Dans le répertoire Sites, créer un fichier docker-compose.yml avec les instructions suivantes à l’intérieur.

version: "3"

services:

  my-apache-app:

    image: httpd # utilise l’image la plus récente

    ports:

      - '8080:80' # exposer 8080 sur l'hôte et l’envoyé à 80 dans le conteneur

    volumes:

      - .:/usr/local/apache2/htdocs/ # lien avec le répertoire de Apache

Lancez votre conteneur et vérifiez que ça fonctionne en allant au site Web <http://www.monsite.com>. La commande pour lancer le conteur est :

docker-compose up -d

Voici d’autres commandes que vous pouvez utiliser.

docker-compose logs

docker-compose --help

docker-compose ps

docker-compose top

Arrêter votre conteneur.

docker-compose down

1. Nous allons restructurer notre environnement. Dans votre répertoire Sites, créer un répertoire monsite. Dans le répertoire monsite, créer un répertoire conf et y copier le fichier httpd.conf récupéré dans l’exercice 2. Créer également un répertoire html et y déplacer le fichier index.html créer dans l’exercice 2.
2. Nous allons créer une image qui répond à notre demande. Dans le répertoire Sites, créer un fichier Dockerfile avec les informations suivantes :

# Dockerfile

# Creer une image pour notre serveur Apache

FROM httpd:latest

# Utilise image officiel

RUN apt-get update \

&& apt-get install -y vim iproute2

# Install vim et iproute pour ss

RUN mkdir /srv/htdocs

# Ajouter le repertoire pour les documents HTML

EXPOSE 80 443

# Exposer les ports 80 et 443

WORKDIR /usr/local/apache2/conf

# Change dans le repertoire de configuration

COPY ./monsite/conf/httpd.conf httpd.conf

# Remplacer httpd.conf par le mien

Construire votre image.

docker image build -t monhttpd .

Tester votre image.

docker run -d --name my-apache-app -p 8080:80 -v "$PWD"/monsite/html:/srv/htdocs monhttpd

Ouvrez un navigateur et allez au site Web http://www.monsite.com:8080.

Entrez dans votre image et vérifiez si les logiciels sont bien installés.

docker container exec -it my-apache-app bash

ss -tnap

exit

Arrêter votre conteneur et l’enlever.

docker container rm -f my-apache-app

1. Modifier le fichier httpd.conf pour remettre les logs comme ils étaient : mettre en commentaire la ligne des journaux de l’exercice 2. De cette manière, Docker va s’occuper des logs (on les consulte avec docker container logs).

# Mettre en commentaire

# CustomLog "logs/access\_log" combined

1. Nous allons maintenant combiner docker-compose et notre Dockerfile. Modifier votre fichier docker-compose.yml pour qu’il ressemble à ça.

version: "3"

services:

  my-apache-app:

    build: # si nécessaire construire l’image de l’info du répertoire

context: .

image: httpd-custom

    ports:

      - '8080:80' # exposer 8080 sur l'hôte et l’envoyé à 80 dans le conteneur

    volumes:

      - ./monsite/html:/srv/htdocs # lien avec le répertoire de Apache

Lançons le tout :

docker-compose up --build -d

Avez-vous une page Web?

Arrêter votre conteneur et effacer l’image :

docker-compose down

docker image ls

docker image rm httpd-custom

1. Enlever la ligne images de votre fichier docker-compose.yml pour qu’il ressemble à ça.

version: "3"

services:

  my-apache-app:

    build: # si nécessaire construire l’image de l’info du répertoire

context: .

    ports:

      - '8080:80' # exposer 8080 sur l'hôte et l’envoyé à 80 dans le conteneur

    volumes:

      - ./monsite/html:/srv/htdocs # lien avec le répertoire de Apache

Lançons le tout :

docker-compose up -d

Quel nom porte l’image maintenant ?

Ceci est un peu plus «clean», car, maintenant je peux utiliser la commande docker-compose down --rmi local pour arrêter le conteneur et effacer mon image.

Avant d’utiliser cette commande, utiliser votre navigateur pour aller sur votre site.

Vérifier que docker-compose reçoit bien les logs, puis arrêter le conteneur et effacer l’image.

docker-compose logs

docker-compose down --rmi local

#### Partie 2 : Modules dans Apache

httpd est un serveur modulaire. Cela implique que seules les fonctionnalités les plus élémentaires sont incluses dans le serveur principal. Des fonctionnalités étendues sont disponibles via des modules qui peuvent être chargés dans httpd. Par défaut, un ensemble de modules de base est inclus dans le serveur au moment de la compilation.

Dans cette partie, nous allons installer le module php. Nous allons encore utiliser le principe de conteneurs en séparant Apache et PHP en les construisant dans des conteneurs séparés. Il est possible de créer une image personnalisée qui inclut à la fois Apache et PHP. Mais, comme nous voulons suivre une approche plus moderne de séparation des services, nous allons les séparer.

Nous allons également essayer d’optimiser notre système en utilisant des images plus petites. Les images construites avec Alpine Linux sont beaucoup plus petites que celles construites avec Debian ou Ubuntu. Par contre, le gestionnaire de package n’est pas le même, Apline utilise apk au lieu de apt.

Étape 1 : Préparer la structure de répertoire

1. Nous allons faire de petits changements dans notre structure de répertoire. Reproduire la structure ci-dessous.

/Sites/

|--docker-compose.yml

|--monsite/

| |--Dockerfile # pour httpd, le déplacer du répertoire Sites

| |--conf/

| |--httpd.conf

| |--html/

| |--index.html

| |--info.php # nous allons le créer plus loin

|--php/

|-- Dockerfile # pour php, nous allons le créer plus loin

1. Si ce n’est pas déjà fait, déplacer le fichier Dockerfile du répertoire Sites au répertoire monsite.
2. Pour pouvoir vérifier php, créer un fichier info.php dans le répertoire monsite/html contenant le code suivant:

<?php

phpinfo();

?>

Étape 2 : PHP avec Apache

1. Modifier votre fichier docker-compose pour inclure php et tenir compte des modifications que nous avons faites.

version: "3.2"

services:

php:

build: './php/'

networks:

- backend

volumes:

- ./monsite/html:/srv/htdocs

apache:

build: './monsite/'

depends\_on:

- php

networks:

- frontend

- backend

ports:

- "8080:80"

volumes:

- ./monsite/html:/srv/htdocs

networks:

frontend:

backend:

Vous pouvez voir que nous utilisons deux réseaux et que nous les nommons. Porter également attention à la dépendance de httpd par rapport à php.

1. Modifiez maintenant le fichier Dockerfile sous monsite pour créer l’image de httpd.

# Dockerfile

FROM httpd:alpine

# Utilise une petite image

RUN apk update; \

apk upgrade;

# Mets a jour le serveur

RUN mkdir /srv/htdocs

# Ajouter le repertoire pour les documents HTML

EXPOSE 80 443

# Exposer les ports 80 et 443

WORKDIR /usr/local/apache2/conf

# Change dans le repertoire de configuration

COPY ./conf/httpd.conf httpd.conf

# Remplacer httpd.conf par le mien

Pourquoi la commande COPY a-t-elle changé ?

1. Créer le Dockerfile pour php dans le répertoire php.

# Dockerfile

FROM php:fpm-alpine

# Utilise une petite image

RUN apk update; \

apk upgrade;

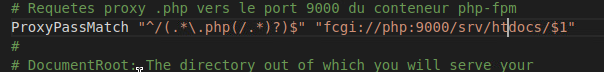
# Mets a jour le serveur

1. Modifier le fichier httpd.conf.

# Ajouter juste avant les commentaires de DocumentRoot

# Requêtes proxy .php vers le port 9000 du conteneur php-fpm

ProxyPassMatch “^/(.\*\.php(/.\*)?)$” “fcgi://php:9000/srv/htdocs/$1”



LoadModule deflate\_module modules/mod\_deflate.so

LoadModule xm2enc\_module modules/mod\_xml2enc.so

LoadModule proxy\_module modules/mod\_proxy.so

LoadModule proxy\_fcgi\_module modules/mod\_proxy\_fcgi.so

# Sous <Directory "/srv/htdocs"> modifier AllowOverride

AllowOverride All

1. À partir du répertoire Sites, exécuter la commande de docker-compose pour lancer le tout en arrière-plan.

docker-compose up --build -d

docker-compose ps

docker-compose logsAllez à la page du site sur votre navigateur. Puis, allez à la page http://www.monsite.com:8080/info.php

Voyez-vous la page d’informations de PHP ?

1. Vous pouvez arrêter votre site Web.

#### Partie 3 : Site Web sécurisé

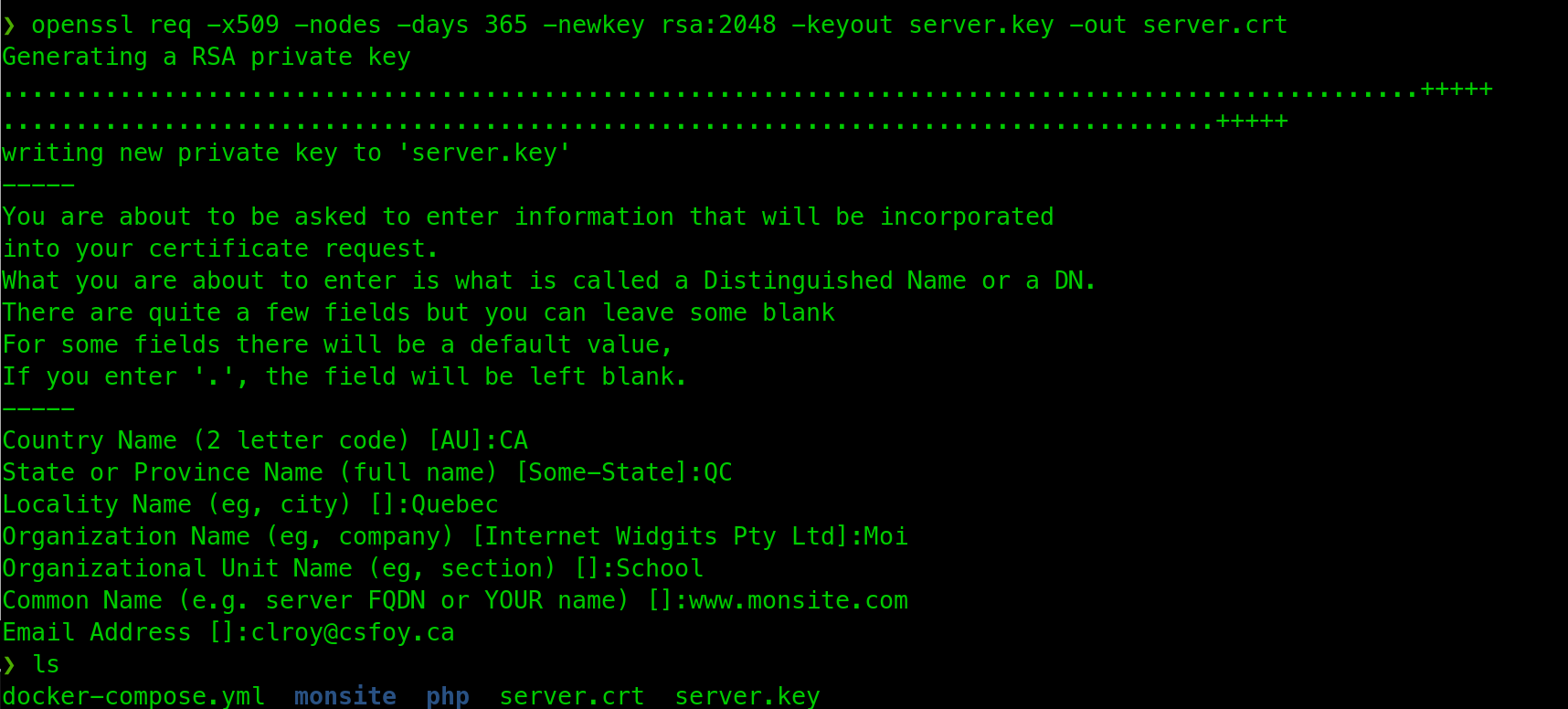
Dans cette partie, nous allons ajouter le chiffrement TLS (encore appelé SSL) à notre site Web. Nous allons commencer par nous générer un certificat, puis nous allons l’activer dans notre serveur httpd.

Étape 1 : Générer un certificat.

1. Pour générer un certificat, nous allons utiliser openssl. La commande est la suivante (sous Windows, vous devez installer openssl) :

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out server.crt

N’oubliez pas de bien inscrire le nom de votre site.



Si vous avez plusieurs sites, vous devriez utiliser des noms de clé et de certificat différents pour chacun de vos sites.

Déplacer ou copier votre clé et certificat dans le répertoire monsite/certs.

Étape 2 : Générer un certificat.

1. Aller au site Web <https://hub.docker.com/_/httpd> et regarder sous la section SSL/HTTPS pour configurer votre serveur avec SSL.
2. Modifier votre fichier docker-compose.yml pour exposer le port 443.
3. Lancer (utiliser l’option --build) et vérifier que votre serveur est bien lancé et utilise maintenant https. (Utiliser https://www.monsite.com/index.html, car nous n’avons pas encore de serveur virtuel).

Est-ce que votre site Web fonctionne avec SSL ?

Le certificat fonctionne, mais il a des « droits » qui ne sont pas bien configurés (plutôt des configurations manquantes).

1. Dans vos configurations, vous avez inclus le fichier conf/extra/httpd-ssl.conf. Ouvrez le fichier et regardez les informations à l’intérieur. Vous pouvez ouvrir celui que vous avez récupéré dans l’exercice 2 ou vous pouvez ouvrir un shell sur le conteneur httpd.

**Note** : Si vous ouvrez un shell, Alpine Linux n’utilise pas bash, mais sh comme shell.

Dans le fichier httpd-ssl.conf, vous avez une section (conteneur) <VirtualHost \_default\_:443> qui définit les informations sur votre connexion sécurisée. Dans cette section vous avez les paramètres suivants :

DocumentRoot "/usr/local/apache2/htdocs"

ServerName www.example.com:443

ServerAdmin you@example.com

Ce sont les paramètres pour les connexions sécurisés, on doit donc les configurer à notre serveur.

1. Copier le fichier httpd-ssl.conf que vous avez récupéré dans l’exercice 2 dans le répertoire Sites/monsite/conf. Modifiez les informations suivantes :

DocumentRoot "/srv/htdocs"

ServerName www.monsite.com:443

ServerAdmin *votre\_courriel* # Cette ligne est optionnelle

Ajouter à la suite, dans le conteneur <VirtualHost \_default\_:443>, les informations suivantes :

#

# DirectoryIndex: sets the file that Apache will serve if a directory

# is requested.

#

<IfModule dir\_module>

DirectoryIndex index.php index.html

</IfModule>

1. Modifier le fichier httpd.conf pour ajouter sous <Directory "/srv/htdocs"> les informations suivantes :

DirectoryIndex index.php index.html

1. Modifier le Dockerfile pour tenir compte de notre fichier httpd-ssl.conf. Renommer le fichier info.php à index.php.
2. Relancer vos conteneurs.

Est-ce que votre site Web fonctionne avec SSL maintenant (utiliser https://www.monsite.com ou https://www.monsite.com/index.php, car nous n’avons pas encore de serveur virtuel)?

1. Arrêter vos conteneurs.

Pour vérification

Remettre une capture d’écran de votre Linux avec Firefox ouvert sur la page sécurisée de votre serveur Web à https://www.monsite.com ou https://www.monsite.com/index.php.

Compétences développées

|  |  |
| --- | --- |
| **00SJ** – Effectuer le déploiement de serveurs intranet. | **00SJ # 1** – Analyser le projet de déploiement.  **00SJ # 2** – Monter les serveurs intranet.  **00SJ # 3** – Installer les services intranet.  **00SJ # 5** – Participer à la mise en service des serveurs intranet. |

**Note** : les compétences sont développées en partie.

Références

<https://httpd.apache.org/docs/2.4/>

<https://cwiki.apache.org/confluence/display/httpd/DistrosDefaultLayout>

<https://hub.docker.com/_/httpd>

<https://docs.docker.com/compose/compose-file/>

<https://docs.docker.com/engine/reference/builder/>

<https://www.sslshopper.com/article-how-to-create-and-install-an-apache-self-signed-certificate.html>