

Partie Pratique

- I. a) Makefile
- II. a) Dockerfile
- II. b) Docker Compose

I. a) Makefile

Objectif

Cet exercice est l'occasion de (re)voir la compilation C, ainsi que la création d'un Makefile afin de simplifier cette compilation.

Code à compiler

Le code de l'exercice est à récupéré ici [LordAzyks/I-a-Makefile](https://github.com/LordAzyks/I-a-Makefile).

```
git clone https://github.com/LordAzyks/I-a-Makefile
```

Tâches à réaliser

1. Compilation

A l'aide de l'outil gcc (ou d'un équivalent), réussir à compiler les sources du projet en un fichier exécutable.

Attention, il est nécessaire de bien observer les sources.

Démontrer la correcte compilation en exécutant le fichier compilé.

```
./result.o 26
```

2. Makefile

Afin de se simplifier la vie pour les prochaines fois qu'une compilation sera nécessaire, réaliser un Makefile afin de réaliser les 3 actions suivantes :

build

```
make build
```

Doit produire le même fichier compiler qu'a l'étape 1.

run

```
make run ???
```

Doit permettre de compiler et d'exécuter le resultat.

Attention, il faut passer un argument au fichier compiler, comment faire ?

clean

```
make clean
```

Doit supprimer le fichier compilé.

II. a) Dockerfile

Objectif

L'objectif de cet exercice est d'installer et de se (re)familiariser avec Docker au travers de la réalisation d'un Dockerfile.

Installation de Docker

Etant dépendant de votre OS, je vous laisse suivre la procédure d'installation adéquat : [Get Docker](#)

Pour ceux sous Linux, il n'est pas nécessaire d'installer Docker Desktop, et je conseil de simplement suivre l'installation du package Docker de votre distribution. Il peut également être judicieux d'installer Docker Compose de suite.

Tâches à réaliser

Que ce soit à l'aide de vos anciens cours, de la [documentation officiel de Docker](#), cette exercice à pour but de réaliser les premiers pas sur Docker en autonomie.

1. Création d'un Dockerfile

De part la création d'un Dockerfile, nous allons créer notre propre image Docker.

Cette image doit :

- Exposer une réponse HTTP en localhost

Cependant, cette image peut :

- Donner n'importe quelle réponse
- Se baser sur une image Docker de son choix, une image de base type linux
- Utiliser n'importe quel solution de Web Server
- Exposer le port de son choix
- Etre beaucoup trop alambiqué et peu pratique, peut importe c'est un exercice

Vous êtes libre, soyez créatifs, et n'ayez pas la même image que votre voisin, que chacun apprenne de sa propre expérience.

2. Docker run (et sûrement du debug)

Votre image est créée ? Il est temps de nous la présenter en action.

II. b) Docker Compose

Objectif

L'objectif de cet exercice est de revenir sur le déploiement de plusieurs container d'un seul coup à l'aide de Docker Compose.

Taches à réaliser

Lors de cette exercice, nous avons pour objectif de décrire une stack applicative web à l'aide d'un docker-compose.yml. Cette stack devra comprendre à minima :

- Un site web frontend de votre choix
- Une backend ou API appelé par le site web
- Une base de donnée appelée par l'API
- Un reverse proxy seul point d'entrée du network

1. Frontend

Cela sera surement la base de votre stack, il existe de nombreuses application open-source fournissant un moyen d'installation Docker par docker compose, soit créatifs, hébergez une application de votre choix.

Voici deux liens utiles pour rechercher des applications à self-host :

AlternativeTo

Vous permet de chercher des alternatives à vos sites ou logiciels préférés. Pensez à rechercher une solution open-source et self-host.

awesome-selfhosted

-> [version web](#)

Awesome-Selfhosted est une liste de solutions gratuites et self-hosted.

Il y a de nombreuses catégories, n'hésitez pas à passer du temps à rechercher ce qui vous intéresse.

Pensez à chercher la mention "Docker" pour ne pas avoir de mauvaise surprise (sauf si vous désirez refaire toute la partie Dockerfile pour un projet open-source ...)

2. Backend

Le backend de votre solution va sûrement dépendre de votre frontend de choix.

3. Database

Pour la base de données, suivant la solution frontend choisi, il est parfois possible de choisir quel moteur de base de données utiliser, n'hésitez pas à choisir celle avec laquelle vous avez le plus d'habitude.

4. Reverse Proxy

Le reverse proxy permet de recevoir l'ensemble des appels à un nom de domain donné, `localhost` dans notre cas.

Lors de cet exercice, je ne veux voir que les ports du reverse proxy ouvert. L'ensemble des communications entre frontend, le backend et l'API doivent passer par le network docker ou par le reverse proxy.

L'implementation d'un reverse proxy est primordiale lors du déploiements de nombreuses applications sur le même serveur host et permet donc la distinction des services par sous-domaines par exemple.

Pour le reverse proxy, voir la [documentation nginx](#) par exemple.