TP5 (Bonus) – Conteneurisation

Dû le 30 avril à 23h00

1 Introduction

Comme vous l'avez vu en cours, Docker est un puissant système de conteneurisation. Lors de ce TP, vous aurez à écrire un dockerfile qui servira à déployer une application Scala/Java, roulant sur Ubuntu, utilisant quelques dépendances. Le système va identifier les caractères d'un PDF et va les imprimer à STDOUT. Il utilise 'tesseractOCR' for la reconnaissance des caractères. Grâce à Docker, le système pourra rouler sur n'importe quel OS.

2 Installation de Docker

Vous devrez installer Docker sur votre ordinateur. Pour une machine "Windows" ou "Mac", vous pouvez suivre les instructions à `https://www.docker.com/products/docker-desktop`. Notez cependant que pour l'installer directement sur Windows, il faut Windows Pro... Si vous n'avez pas Windows Pro, vous pouvez l'installer sur une VM.

Pour Ubuntu, vous pouvez suivre ceci : `https://docs.docker.com/engine/install/ubuntu/`. Pour les autres versions de Linux, les instructions sont disponibles sur le web.

2.0.1 Installation de machine virtuelle (si vous avez bésoin)

Si vous n'avez pas de VM, installer VirtualBox à partir d'ici `https://www.virtualbox.org/wiki/Downloads`. Ensuite, téléchargez Ubuntu 18.04 ici : `https://ubuntu.com/#download`. Pendant que Ubuntu télécharge, créez une nouvelle VM dans VirtualBox en cliquant sur le bouton bleu "New", ensuite sélectionnez le type "Linux" et la version "Debian 64-bit". Vous pouvez faire "next". Ensuite, allouez de la mémoire pour la VM, peut être quelque chose comme 2000MB. Faites "next". Sur cet écran, créez un nouveau disque dur virtuel (gardez toutes les options standard pour ce disque, mais donnez-lui 16GB d'espace de storage). Une fois la VM crée, faites "start", et on vous demandera de sélectionner un ISO. Attendez que Ubuntu termine de télécharger, et sélectionnez-le. Ensuite, faites "start". Procédez ensuite à l'installation normale d'un ordinateur Ubuntu, puis, passez aux instructions pour installer Docker sur Ubuntu ci-bas.

3 Guide de Démarrage

Dans un dossier clonez le repo pour le TP :

```
$ git clone https://github.com/IFT2245/TP-docker.git
```

Dans le repo, vous verrez deux fichiers, "Makefile" et "Dockerfile" et un repertoire "pdfs". Ce sont les deux fichiers que vous devrez modifier pour ce TP.

Pour commencer, vous pouvez rouler :

```
$ make clone
```

Ça va cloner le repo qu'on a besoin pour l'exécution du système (`https://github.com/Velythyl/OS-docker-TP-Scala.git`). Ultimement, le contenu de ce repo sera copié dans votre Docker, et ensuite vous pourrez exécuter le système sur n'importe quel PDF de votre choix en le "montant" dans votre "Docker container" à l'exécution.

4 Tâche 1 : Dockerfile

Le Dockerfile est une recette pour créer une image Docker. Pour une référence complète sur les Dockerfile, voir : `https://docs.docker.com/engine/reference/builder/`.

Le code du repo que vous avez cloné avec "make clone" est assez simple : il lit tous les fichiers PDF du dossier "pdfs" (il assume que ce sont des PDFs, et que ces PDFs sont bien formés, etc), et il output leur contenu textuel à la console. Les textes sont séparés par "\n\n\t\n\n".

4.1 Les éléments de votre Dockerfile

Voici les éléments que vous devrez inclure dans votre Dockerfile pour que tout fonctionne correctement (en anglais, pour pouvoir vous donner de petits indices sur des mots clés de Docker... ;)).

- Commencer "FROM" l'image openjdk12 qui contient java : `https://hub.docker.com/r/adoptopenjdk/openjdk12`
- Avec "RUN" installer (avec apt) zip, unzip, and tesseract-ocr dans votre image
- Ainsi, installer 'sbt' avec :
  ```
  $ curl -s "https://get.sdkman.io" | bash
  $ source "/root/.sdkman/bin/sdkman-init.sh"
  $ sdk install sbt
  ```
- Avec "COPY" copier les contenus de la repertoire 'OS-docker-TP-Scala' a un répertoire dans votre image.
- Finallement, avec "CMD", assurez-vous que la commande qui est exécutée par défaut lorsque l'image est exécutée est :
  ```
  $ sbt --error run
  ```
  Mais notez que cette commande devra être roulée à partir du fichier où le code du repo est dans votre image. Vous aller peut être vouloir utiliser une commande "WORKDIR" pour contrôler le dossier courant du Docker.

Vous pouvez tester que votre Dockerfile construit bien une image Docker avec

```
$ make build
```

5 Tâche 2 : Run !

Vous devez modifier la partie ‘run’ dans le Makefile pour que ça roule le "Docker image" qui a été construit avec ‘build’.

Dans la commande exécutée par ‘run’, vous devez monter le dossier "pdfs" local de l’hôte du Docker dans le dossier "pdfs" approprié dans le Docker (il écrasera donc celui déjà fourni dans le code sur GitHub, qui en ce moment ne contient qu’un PDF nommé "fct.pdf"). Important ! N’utilisez pas de chemins absolus. Notre machine de correction n’a surement pas la même hiérarchie de fichier que votre machine... Notez cependant que les chemins relatifs ne sont pas supportés par "Docker run", mais une solution est de faire "$(pwd)" au lieu du "." du "./".

Donc, faire :

```
$ make run
```

devrait faire rouler Tesseract-OCR (par l’entremise du code Scala) sur les PDFs qui sont dans le dossier local "pdfs" (pour l’hôte, donc PAS ceux qui sont dans le repo `https://github.com/Velythyl/OS-docker-TP-Scala.git`).

Vous devriez voir :

“J’adore la classe des systémes d’exploitation\n1”

imprimé sur votre terminal (après d’autres prints créés par SBT ou Scala, etc.) si vous utilisez le PDF qui est inclus avec le repo TP-docker que vous avez cloné au début de ce TP.

6 Remise et Correction

Vous devez remettre les deux fichiers, le Dockerfile (appelé "Dockerfile", sans extension), et le fichier “Makefile” (sans extension). Ces deux fichiers doivent être dans un .zip nommé "PrénomNomdefamille1_PrénomNomdefamille2" et remis avec ce lien dropbox.

Ce TP est sur 0 points, avec 5 points bonus sur la session si la correction automatique fonctionne.

La correction sera automatique : si, en appelant

```
$ make clean
$ make clone
$ make build
$ make run
```

avec un dossier contenant des PDF de notre choix, on arrive au résultat attendu, vous aurez tous les points. Le but d’un Docker est la reproductibilité, et c’est ce qu’on vérifie.