

DÉPLOIEMENT D'UN SERVICE AVEC DOCKER

ACTIVITÉ 3 – CONSTRUIRE UNE IMAGE AVEC UN DOCKERFILE ET LA PUBLIER SUR DOCKER HUB (LE REGISTRE PUBLIC DE DOCKER)

Lien :

- Lien vers une formation sur Docker complète mais très simple : <https://blog.microlinux.fr/formation-docker/>

Les documents suivants viennent en complément :

- récapitulatif des commandes Docker ;
- les Instructions Dockerfile.

SOMMAIRE

A. Préalable : Création d'un compte sur le Docker Hub.....	2
B. Le fichier Dockerfile.....	3
1. Principe de fonctionnement.....	3
2. Création du fichier Dockerfile.....	4
C. Construction d'une image.....	6
D. Publication d'une image sur le docker Hub (registre public).....	7
1. Un petit rappel sur les différents types d'images.....	7
2. Déploiement d'une image sur le Docker Hub (registre public).....	7
E. Modification d'une image : principes et exemple.....	9
F. Construction, publication et déploiement d'une nouvelle image en autonomie.....	11

Dès la deuxième activité, nous avons créé une image personnalisée. Pour ce faire nous avons :

- lancé un conteneur en mode interactif ;
- fait des manipulations (mise à jour et installation du serveur SSH) ;
- réalisé un « commit ».

Ce mode de fonctionnement s'avère lourd, compliqué à maintenir et à faire évoluer. Comme nous pouvons le constater sur le « Docker hub », les images Docker sont généralement construites à partir d'un Dockerfile qui est un fichier d'instructions contenant toutes les informations pour les créer : des commandes à exécuter pour installer un logiciel, des variables d'environnement, des méta-données (comme le mainteneur du Dockerfile), des copies de fichiers, etc.

L'objectif est de :

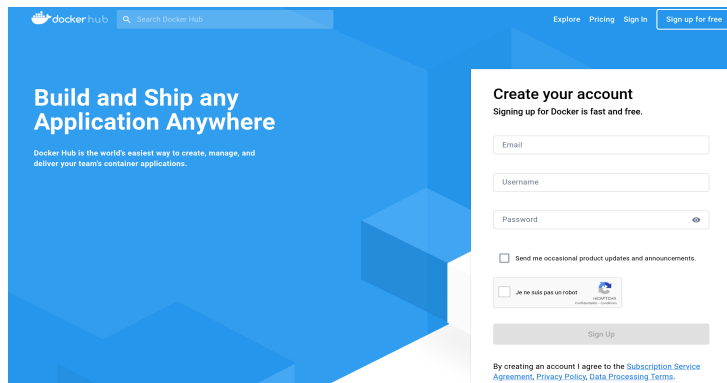
- créer une image très simple similaire à celle que nous avons créée dans la troisième activité (la distribution Ubuntu avec le service SSH) et de la publier sur le registre public (Docker Hub) ;
- modifier cette image et en publier une nouvelle version ;
- construire, publier et déployer une nouvelle image en autonomie.

Vous devez faire toutes les manipulations, notamment celles précédées par : 

A. PRÉALABLE : CRÉATION D'UN COMPTE SUR LE DOCKER HUB

Pour publier une image sur le registre public, il est nécessaire de posséder un compte. Docker propose des comptes gratuits ou payants. Nous nous contenterons du compte gratuit qui permet un nombre de dépôts publics illimités et un dépôt privé.

La procédure de création d'un compte sur <https://hub.docker.com> est très simple :



Cliquez sur "Sign up for free" et suivez les instructions pour créer un compte.

Le « username » sera ensuite obligatoirement utilisé pour préfixer le nom de l'image déployé.

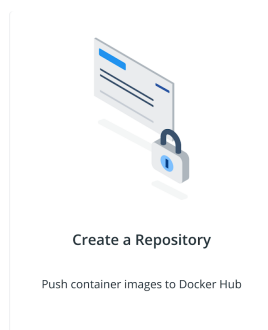
Il suffit ensuite de valider le compte via le mail reçu et de se connecter.

Un dépôt (*repository*) est un ensemble d'images avec le même nom, mais potentiellement des étiquettes (tag) différentes.


i Il n'est pas obligatoire de configurer le dépôt avant d'envoyer la première image : **il sera créé automatiquement sur le Docker Hub public**. Dans la page du dépôt, vous pourrez ensuite ajouter une description, voir et supprimer les tags, etc.

Welcome to Docker Hub

Here are a few things to get you started.





Create Repository

 **aporaf** ▼

Visibility

Using 0 of 1 private repositories. [Get more](#)

☒ **Public** 
Public repositories appear in Docker Hub search results

☐ **Private** 
Only you can view private repositories

Pro tip

You may push a new image to this repository using the CLI:

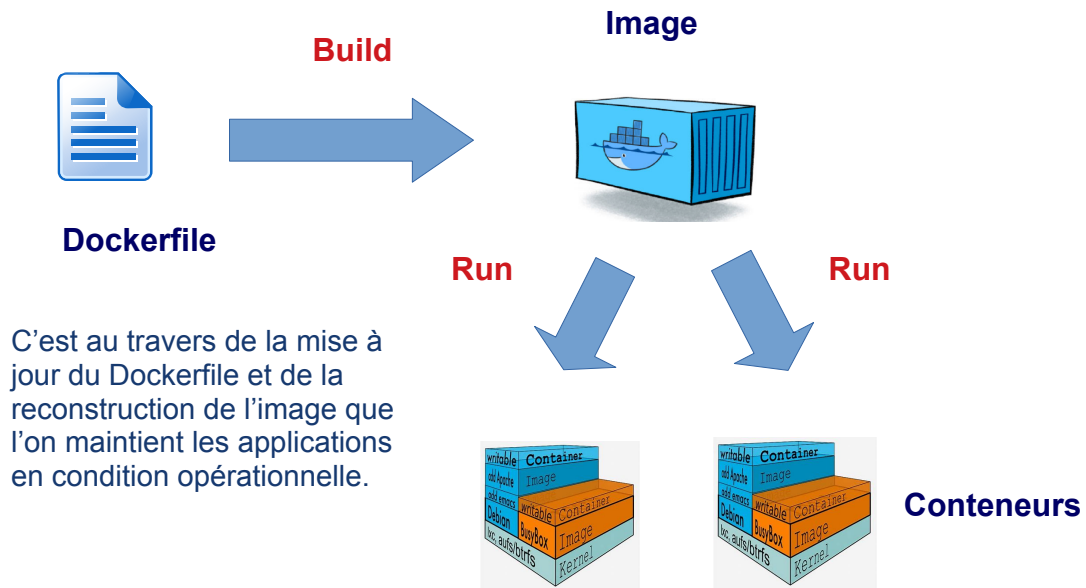
```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired image repository tag.

i Il est possible de gérer un dépôt Docker avec comme source un dépôt tel que Gitlab ou Github. La plateforme Docker se charge alors automatiquement de la compilation et de la génération de l'image Docker avec une mise à jour du dépôt dès que le code source déposé sur Gitlab ou Github est modifié.

B. LE FICHIER DOCKERFILE

1. PRINCIPE DE FONCTIONNEMENT



Docker construit automatiquement les images en lisant, dans l'ordre, les instructions d'un fichier Dockerfile (appelé par défaut « dockerfile »). La commande permettant de construire l'image a la structure suivante : **docker build [-t <nom_image>[:<tag>]] <dossier_dockerfile>**

Lorsque vous exécutez cette commande, le répertoire de travail est appelé **contexte de construction**. Par défaut, le fichier « dockerfile » est supposé se trouver dedans.

Étapes :

- Créer un répertoire pour le contexte de construction : **mkdir -p ~/contexte/ubuntu-ssh.**
- Créer et éditer dedans un fichier appelé « dockerfile »

*Vous pouvez utiliser un éditeur en ligne de commande (vim, nano) mais pour plus de confort et d'aide dans le débogage, il est conseillé de travailler avec **VSCo**de ou **VSCodium** :*

```
Fichier  Edition  Sélection  Affichage  Atteindre  Exécuter  Terminal  Aide
EXPLORATEUR  ...  dockerfile X
CONTEXTE [SSH: 192.168.6...
  ubuntu-ssh
    dockerfile
ubuntu-ssh > dockerfile
1  FROM ubuntu:22.04
2
3  LABEL Mainteneur "Apollonie Raffalli"
4  LABEL description "Ubuntu TLS avec le serveur SSH activé"
5
6
7  # Installation des logiciels
8  RUN apt update \
9      && apt -y --no-install-recommends install \
10     openssh-server \
11     tzdata \
12     locales \
13     && rm -rf /var/lib/apt/lists/* \
14     && apt clean
--
```

2. CRÉATION DU FICHIER DOCKERFILE

Préalables

Nous allons créer une image basée sur celle de la dernière version TLS d'Ubuntu (Ubuntu 22.04), y installer et configurer le service SSH.

☑ Supprimer le conteneur créé lors de la première activité.

Vous trouverez à cette adresse https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ les bonnes pratiques pour la construction d'une image. À noter également que, généralement, les images disponibles sur le registre public sont accompagnées du fichier Dockerfile qui a permis de les générer. Ce qui suit ne constitue qu'un exemple simplifié de ce qu'il est possible de faire.

Le fichier

Les indentations opérées ci-dessous ne sont pas obligatoires et le symbole « \ » permet de poursuivre une entrée à la ligne suivante. Ce formatage contribue à une meilleure lisibilité du fichier.

```
1 FROM ubuntu:22.04
2
3 LABEL Mainteneur "Prénom Nom [@mail ou URL]"
4 LABEL description "Ubuntu TLS avec le serveur SSH activé"
5
6 # Installation des logiciels
7 RUN apt update \
8     && apt -y --no-install-recommends install \
9         openssh-server \
10        tzdata \
11        locales \
12    && rm -rf /var/lib/apt/lists/* \
13    && apt clean
14
15 # Configuration de la Timezone Europe/Paris et des locales
16 ENV LANG=fr_FR.UTF-8
17
18 RUN rm /etc/localtime \
19     && ln -s /usr/share/zoneinfo/Europe/Paris /etc/localtime \
20     && dpkg-reconfigure --frontend noninteractive tzdata \
21     && sed -i 's/# fr_FR.UTF-8 UTF-8/fr_FR.UTF-8 UTF-8/' /etc/locale.gen \
22     && echo 'LANG=${LANG}'>/etc/default/locale \
23     && dpkg-reconfigure --frontend=noninteractive locales \
24     && update-locale LANG=${LANG}
25
26 # Création de l'utilisateur qui pourra se connecter au serveur SSH
27 RUN useradd --create-home --shell /bin/bash userssh \
28     && echo "userssh:mdpuser" | chpasswd
29
30
31 RUN service ssh start
32
33 EXPOSE 22
34
35 CMD ["/usr/sbin/sshd", "-D"]
```


Les explications des mots clés sont fournies sur la page suivante et détaillées dans le document « instructions_dockerfile ».

FROM : Un fichier Dockerfile commence toujours par la commande FROM, qui permet de définir une image de base à partir de laquelle la nouvelle image sera construite. Dans notre exemple, l'image de base est une Ubuntu, version 22.04. Si cette image n'est pas présente sur la machine hôte, elle sera téléchargée à partir des registres officiels Docker.

LABEL : fournit des couples clé/valeur qui serviront de métadonnées décrivant l'image lorsque celle-ci sera diffusée et utilisée. Cette technique permet de communiquer des informations aux utilisateurs de l'image.

ENV : permet de définir des variables d'environnement exploitables lors de la compilation du Dockerfile et dans les futurs conteneurs (ici pour configurer la date et les locales). Comme nous l'avons déjà vu, les valeurs par défaut définies peuvent ensuite être modifiées via le paramètre « -e » de la commande « docker run ».

RUN : permet d'exécuter une ou plusieurs commandes à l'intérieur de l'image. Ici, nous exécutons plusieurs commandes « chaînées » par « && » : mise à jour de la liste des paquets et du système, installation de paquets et nettoyage du gestionnaire de paquets afin que l'image soit un peu plus légère ainsi que dans un second RUN, la mise à jour de la date et des locales, puis la création d'un utilisateur avec l'affectation de son mot de passe et enfin le démarrage du service.


 Il est possible et conseillé d'exécuter plusieurs commandes sur une même instruction RUN pour limiter le nombre de layer (chaque commande dans le fichier Dockerfile génère une couche – layer – qui vient se superposer à la couche précédente) et donc la taille de l'image. Nous aurions pu ne faire qu'une seule commande RUN (mais cela nuit à la lisibilité du Dockerfile).

Remarque : la commande « sed » permet de remplacer une expression par une autre. Ici, il s'agit d'enlever le « # » pour dé-commenter « fr_FR.UTF-8 UTF-8 » dans le fichier « /etc/locale.gen ». On remplace « # fr_FR.UTF-8 UTF-8 » par « fr_FR.UTF-8 UTF-8/ »

EXPOSE : indique le ou les ports sur lesquels un conteneur écoute les connexions (on utilise en règle général le port « connu » pour l'application). Pour un accès externe, les utilisateurs devront exécuter « docker run » avec l'option « -p » pour mapper le port.

 Nous pouvons considérer ici que la définition de l'image est réalisée : nous avons un service ssh qui pourra potentiellement être utilisé sur le port 22 « exposé ».

CMD : ce mot clé définit le comportement par défaut au lancement d'un conteneur alors que tout ce qui précédait a permis de définir l'image ==> Nous décidons ici de lancer le service SSH au premier plan au lancement du conteneur (nous n'aurons pas besoin de le faire avec la commande « docker run »). Ici la syntaxe passe un tableau au format JSON : CMD ["executable", "param1", "param2"...]. Les éléments sont séparés par une virgule et chacun d'eux doit être entre guillemets. À noter qu'il est possible (mais moins conseillé) d'utiliser une autre syntaxe plus simple sous forme de « ligne de commande » : « /usr/bin/sshd -D ».

 S'il y a plusieurs CMD, seule la dernière commande sera exécutée et la commande et les arguments fournis par CMD peuvent être remplacés depuis la ligne de commande lorsque le conteneur docker est lancé : on parle de « surcharge ». Si par exemple on veut seulement faire un « ls » : *docker run <image> ls* avec la commande « ls » qui remplacera le contenu de CMD.

C. CONSTRUCTION D'UNE IMAGE

La commande qui permet de construire une image est la suivante :

```
docker build -t <docker-id>/ubuntu-ssh:1.0 ~/contexte/ubuntu-ssh
```

Remplacez <docker-id> par votre ID (login) de compte Docker Hub créé précédemment (même si le nom de l'image pourra être modifié avant d'être publié sur le Docker Hub).

« /root/contexte/ubuntu-ssh » est le dossier qui contient le dockerfile qui va construire l'image.

```
user@servDockerAR:~$ docker build -t aporaf/ubuntu-ssh:1.0 ~/contexte/ubuntu-ssh/
```

```
...
[+] Building 109.6s (9/9) FINISHED
...
=> [1/5] FROM
docker.io/library/ubuntu:22.04@sha256:ec050c32e4a6085b423d36ecd025c0d3ff00c38ab93a3d71a4
60ff1c44fa6d77
8.2s
...
=> [2/5] RUN apt update      && apt -y --no-install-recommends install      openssh-server
tzdata      locales      && rm -rf /var/lib/apt/lists/*      && apt clean
74.0s
=> [3/5] RUN rm /etc/localtime      && ln -s /usr/share/zoneinfo/Europe/Paris
/etc/localtime      && dpkg-reconfigure --frontend noninteractive tzdata      && sed -i 's/#
fr_FR.UTF-8 UTF-8/fr_FR.U 8.6s
=> [4/5] RUN useradd --create-home --shell /bin/bash userssh      && echo
"userssh:mdpuser" | chpasswd
6.6s
=> [5/5] RUN service ssh start
5.2s
=> exporting to image
5.5s
=> => exporting layers
5.5s
=> => writing image
sha256:c008dc3c19bd4ce3c6f861b42f38fbbf2e62a16449796aa257ed188e6fd891a7
0.0s
=> => naming to docker.io/aporaf/ubuntu-ssh:1.0
```

```
user@servDockerAR:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aporaf/ubuntu-ssh	1.0	c008dc3c19bd	39 minutes ago	107MB

Lancement d'un conteneur associé à cette image (nous n'avons pas à démarrer le service SSH, car nous avons prévu de le faire dans le Dockerfile).

```
user@servDockerAR:~$ docker run -d -p 192.168.60.111:2222:22 --name servssh
aporaf/ubuntu-ssh:1.0
```


```
ca042351eda5c8d0394f401b436e02d5748874e8823b31845e5ac057b265e1d6
```

Test du fonctionnement, en se connectant avec le login/mot de passe « userssh/mdpuser ».

```
user@servDockerAR:~$ ssh userssh@192.168.60.111 -p 2222
```

```
The authenticity of host '[192.168.60.111]:2222 ([192.168.60.111]:2222)' can't
be established.
```

```
...
userssh@ca042351eda5:~$
```

 Les commandes « locale » et « date » doivent aussi montrer que les configurations réalisées dans le dockerfile ont bien été prises en compte.

D. PUBLICATION D'UNE IMAGE SUR LE DOCKER HUB (REGISTRE PUBLIC)

1. UN PETIT RAPPEL SUR LES DIFFÉRENTS TYPES D'IMAGES

Sur le Docker Hub, se côtoient plusieurs types d'images dont les :

- **images communautaires** déposées par n'importe qui sur le registre (du moment que l'on possède un compte) qui peuvent contenir des failles de sécurité intentionnelles, des virus ou n'importe quel type d'attaque. Le nom de ce type d'image est préfixé de l'identifiant du propriétaire de l'image (séparé du nom de l'image elle-même par un symbole /) ;
- **images officielles** publiées par l'équipe de Docker amenant ainsi le niveau de sécurité le plus haut. Leur nom n'est pas préfixé et leur liste est maintenue sur le dépôt GitHub nommé `docker-library/official-images` (<https://github.com/docker-library/official-images>, puis descendre dans le répertoire nommé `library`) ;
- **images certifiées** provenant d'éditeurs commerciaux vérifiés par Docker. La société s'engage sur le contenu des images et leur performance dans une infrastructure certifiée.

Dans la mesure du possible, il vaut mieux utiliser une image certifiée ou officielle plutôt qu'une image communautaire, mais nous n'avons parfois pas le choix. Dans ce cas, d'autres critères sont à prendre en compte :

- le nombre d'étoiles ;
- une description plus ou moins complète de l'image ;
- s'il y a un dépôt Github comme source de l'image ⇒ dans ce cas, Docker a créé automatiquement l'image à partir du code source disponible, et en particulier du fichier Dockerfile ⇒ l'image disponible n'a pas pu être modifiée et il est possible de consulter le Dockerfile qui donne des informations précises sur le contenu de l'image.


2. DÉPLOIEMENT D'UNE IMAGE SUR LE DOCKER HUB (REGISTRE PUBLIC)

Préalable sur le nom de l'image

```
user@servDockerAR:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aporaf/ubuntu-ssh	1.0	c008dc3c19bd	39 minutes ago	107MB

Le nom de l'image est obligatoirement préfixé par le **nom de votre compte sur le Docker Hub**. Ce préfixe est **suivi du caractère "/"** puis de **votre nom de dépôt** (ici « ubuntu-ssh » ; si ce dernier n'est pas encore créé, il le sera automatiquement lors de la publication de l'image) et enfin **la version de l'image (le tag)**. Si la version de l'image est omise, le tag "latest" sera appliqué.

 Il est possible de modifier le nom de l'image avec la commande : **docker tag c00 login/nouveau_nom:nouveau_tag** où c00 représente les 3 premiers caractères de l'identifiant de l'image. Vous pouvez aussi utiliser le nom du dépôt (repository) à la place : **docker tag <ancien nom de l'image> login/nouveau_nom:nouveau_tag**.

```
user@servDockerAR:~$ docker tag c00 aporaf/serveurssh:1.1
```

```
user@servDockerAR:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aporaf/serveurssh	1.1	c008dc3c19bd	25 minutes ago	107MB
aporaf/ubuntu-ssh	1.0	c008dc3c19bd	25 minutes ago	107MB

Les deux tags désignent la même unique image ⇒ vous pouvez ensuite supprimer l'image avec le nom qui ne convient pas :

```
user@servDockerAR:~$ docker rmi aporaf/serveurssh:1.1
```

```
Untagged: aporaf/serveurssh:1.1
```

Pour envoyer l'image, il est nécessaire de s'authentifier en console :

🔗 user@servDockerAR:~\$ docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

Username: aporaf

Password:

WARNING! Your password will be stored unencrypted in `/root/.docker/config.json`.

Configure a credential helper to remove this warning. See

<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded

L'envoi de l'image (par défaut sur le Docker Hub public) peut alors être réalisé :

🔗 user@servDockerAR:~\$ docker push aporaf/ubuntu-ssh:1.0

The push refers to repository [docker.io/aporaf/ubuntu-ssh]

41908566974b: Pushed

...

701979165e5c: Pushed

bce45ce613d3: Mounted from library/ubuntu

1.0: digest:

sha256:3bd9c727493c1e19bd7670a260186d4ca50c72a5703362ffb6261f1bd2d81917 size: 1366

Nous pouvons voir sur l'image ci-dessous qu'un dépôt « ubuntu-ssh » a été automatiquement créé :

 aporaf

Filter by repository name...

Create Repository +

REPOSITORY	DESCRIPTION	LAST MODIFIED
 aporaf / ubuntu-ssh	This repository does not have a description...	3 minutes ago

**aporaf/ubuntu-ssh:1.0**
DIGEST: sha256:3bd9c727493c1e19bd7670a260186d4ca50c72a5703362ffb6261f1bd2d81917

OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE
linux/amd64	37.29 MB	2 minutes ago by aporaf	Image

En cliquant sur le nom de l'image, nous voyons les détails des « layers » :

Image Layers Vulnerabilities

IMAGE LAYERS

1	ARG RELEASE	0 B Command
2	ARG LAUNCHPAD_BUILD_ARCH	0 B ARG RELEASE
3	LABEL org.opencontainers.image.ref.name=ubuntu	0 B
4	LABEL org.opencontainers.image.version=22.04	0 B
5	ADD file ... in /	28.17 MB
6	CMD ["/bin/bash"]	0 B
7	LABEL Mainteneur=Apollonie Raffalli	0 B
8	LABEL description=Ubuntu TLS avec le	0 B
9	RUN /bin/sh -c apt update	8.07 MB
10	ENV LANG=fr_FR.UTF-8	0 B
11	RUN /bin/sh -c rm /etc/localtime	1.05 MB
12	RUN /bin/sh -c useradd --create-home	4.25 KB
13	RUN /bin/sh -c service ssh	113 B
14	EXPOSE map[22/tcp:{}]	0 B
15	CMD ["/usr/sbin/sshd" "-D"]	0 B

L'image est maintenant « sauvegardée » sur le registre officiel et peut ainsi être utilisé par n'importe qui et à partir de n'importe quel système.

E. MODIFICATION D'UNE IMAGE : PRINCIPES ET EXEMPLE

L'objectif est de montrer que l'on peut facilement modifier (et donc maintenir) une image via un fichier « dockerfile » mais également d'illustrer les principes des variables d'environnement et de l'adjonction d'un script à l'image.

Nous allons modifier l'image qui permettra également de se connecter en ssh avec l'utilisateur « root » et un mot de passe par défaut « rootAdmin » ou un mot de passe défini dans la commande « docker run » via une variable d'environnement « MDP ».

Cela nécessite de modifier le Dockerfile et d'ajouter un script à l'image.

1. Écriture du contenu d'un script « script_entry.sh » (volontairement simplifié ici) qui, pour pouvoir être intégré à l'image, doit se trouver dans le dossier contexte « ubuntu-ssh » :

```
#!/bin/bash
# Attribution du mot de passe de root (rootAdmin par défaut défini dans le
dockerfile)
echo root:$MDP | chpasswd
# Démarrage au premier plan du service
/usr/sbin/sshd -D
```

Après avoir **ajouté le script à l'image** (mot clé **COPY**), ce dernier **sera exécuté** par défaut à **chaque lancement d'un conteneur** (mot clé **CMD**). Le script fixe notamment le mot de passe de root selon le contenu de la variable MDP définie dans le Dockerfile (rootAdmin par défaut) ou passé dans une variable d'environnement via la commande « docker run ».

2. Modification du Dockerfile

```
30 # Pour permettre également à root de se connecter
31 RUN sed -ri 's/^#?PermitRootLogin\s+.*?/PermitRootLogin yes/'
32 /etc/ssh/sshd_config
33
34 ENV MDP=rootAdmin
35
36 COPY script_entry.sh /script_entry.sh
37 RUN chmod +x /script_entry.sh
38
39 RUN service ssh start
40
41 EXPOSE 22
42
43 CMD ["/script_entry.sh"]
```

 Il est rappelé que tout ce qui précède « CMD » n'est exécuté qu'une seule fois pour constituer l'image.

RUN modifie le fichier etc/ssh/sshd_config de manière à ce que « root » puisse se connecter en SSH.

ENV définit la variable MDP que l'on utilise dans le script pour attribuer le mot de passe à root. Ici, si la variable d'environnement « MDP » n'est pas utilisée dans la commande *docker run*, root se connectera avec le mot de passe « rootAdmin »

COPY permet de copier des fichiers/dossiers depuis l'hôte Docker vers l'image. Ces derniers doivent se trouver dans le contexte de construction c'est-à-dire dans le même dossier que le fichier Dockerfile. Ici, COPY copie le fichier script.sh dans l'image sous le nom « script_entry.sh ». Il peut également être utilisé pour importer les sources d'un projet (application, base de données, etc) ou des fichiers de configuration. Le droit d'exécution est ensuite donné au script.

Toutes les couches d'une image (layers) sont mises dans le cache. Lorsque l'on lance la construction d'une image, le système cherche dans le cache des couches qu'il peut réutiliser pour optimiser le temps de construction de l'image. Ainsi, seules les lignes concernées par la modification seront ré-exécutées.

3. Reconstruction de l'image (il est possible de lui donner un autre nom)

La sortie de la commande montre que jusqu'à l'étape 6/8, le système utilise son cache.

user@servDockerAR:~\$ docker build -t aporaf/ubuntu-ssh:1.1 ~/contexte/ubuntu-ssh/

```
[+] Building 8.8s (13/13) FINISHED                                docker:default
=> [internal] load build definition from dockerfile                0.1s
=> => transferring dockerfile: 1.25kB                             0.0s
=> [internal] load .dockerignore                                  0.1s
=> => transferring context: 2B                                      0.0s
=> [internal] load metadata for docker.io/library/ubuntu:22.04    0.4s
=> [1/8] FROM docker.io/library/ubuntu:22.04@sha256:ec050c32e4a6085b423d 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 229B                                   0.0s
=> CACHED [2/8] RUN apt update && apt -y --no-install-recommends inst 0.0s
=> CACHED [3/8] RUN rm /etc/localtime && ln -s /usr/share/zoneinfo/Eu 0.0s
=> CACHED [4/8] RUN useradd --create-home --shell /bin/bash userssh & 0.0s
=> CACHED [5/8] RUN sed -ri 's/^#?PermitRootLogin\s+.*?PermitRootLogin y 0.0s
=> [6/8] COPY script_entry.sh /script_entry.sh                   0.1s
=> [7/8] RUN chmod +x /script_entry.sh                           1.8s
=> [8/8] RUN service ssh start                                    4.9s
=> exporting to image                                             1.3s
=> => exporting layers                                           1.3s
=> => writing image sha256:a5d89b062e20106ccaf1a8f584bd08e5f33e9fa94070c 0.0s
=> => naming to docker.io/aporaf/ubuntu-ssh:1.1
```

4. Suppression du premier conteneur : docker rm -f servssh

5. Lancement d'un conteneur à partir de cette nouvelle image :

a) En gardant la valeur rootAdmin pour MDP :

 **docker run -d -p 192.168.60.111:2222:22 --name servssh aporaf/ubuntu-ssh:1.1**

Test : ssh root@192.168.60.111 -p 2222 ⇒ authentification avec rootAdmin

b) En modifiant le mot de passe de root

 **docker run -d -p 192.168.60.111:2222:22 --name servssh -e MDP=nouveauMDP aporaf/ubuntu-ssh:1.1**

Test : ssh root@192.168.60.111 -p 2222 ⇒ authentification avec nouveauMDP

6. Envoi de la nouvelle image sur le Docker Hub

 **user@servDockerAR:~\$ docker push aporaf/ubuntu-ssh:1.1**

```
The push refers to repository [docker.io/aporaf/ubuntu-ssh]
a9fe3dd9aa39: Pushed
...
28fda4c35f69: Pushed
7f128cfed18a: Layer already exists
bce45ce613d3: Layer already exists
1.1: digest:
sha256:90c6eea546b450a4efbcb6ac69099c318b7bc6f560c6bc87ea87822e8f6bd087 size:
1988
```



Nous constatons là aussi que seules les couches supplémentaires sont envoyées.

F. CONSTRUCTION, PUBLICATION ET DÉPLOIEMENT D'UNE NOUVELLE IMAGE EN AUTONOMIE

L'objectif est de créer une image très simple (login/debian12-apache2:1.0) permettant de déployer des pages Web statiques via le logiciel Apache puis, une fois opérationnelle, la publier sur le registre officiel.

Contraintes particulières :

- l'image doit être basée sur la version 12 de Debian et non sur l'image « httpd » ;
- le serveur Web écoutera sur le port 80 et doit être accessible sur le port 8008 ;
- la zone Europe/paris et les locales « françaises » doivent être gérées ;
- une page Web personnalisée doit être intégrée à l'image.

Aide complémentaire :

- il n'est pas nécessaire de lancer le service apache 2 (comme nous l'avons fait pour ssh) car il est lancé automatiquement après l'installation ;
- le mot clé CMD (CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]) définit le comportement par défaut au lancement d'un conteneur : ici le service apache2 sera démarré par défaut au premier plan (nous n'aurons pas besoin de le faire avec la commande « docker run »).

Vous noterez ci-dessous les étapes préalables effectuées.

 Créer le fichier dockerfile.

 Créer l'image correspondante.

 Lancer le conteneur

 Tester l'image

 Publier l'image sur le Docker Hub et vérifier de sa disponibilité en ligne.