


|  |  |
|--|--|
|  <p>Se former autrement<br/>HONORIS UNITED UNIVERSITIES</p> | <h1>Applications web Distribuées</h1> <p>Année universitaire 2024/2025</p> <p>UP WEB</p> |
| <p><b>Atelier :</b></p> <h2>Micro Service avec Docker</h2>   |  |

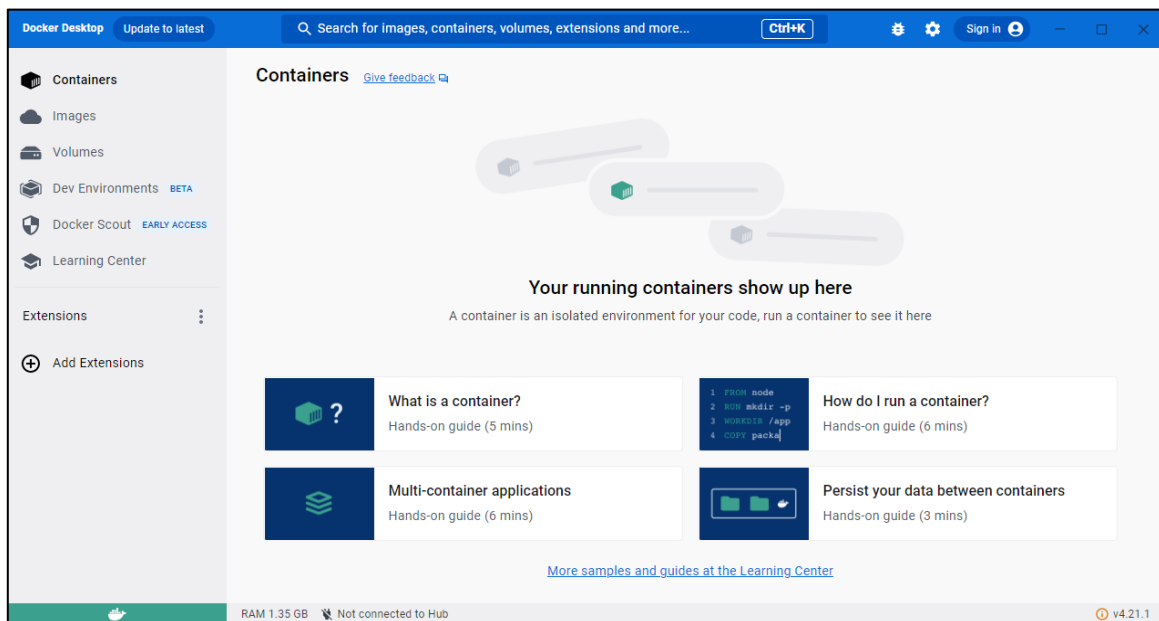
## Objectifs

- Déployer un Micro service Spring Boot avec Docker.
- Découvrir comment héberger un Micro service sur un conteneur docker hub.
- Utiliser l'outil « [Play With Docker](#) » pour tester l'image en ligne.

## Partie I : Dockerisation d'une application MS sur Docker.

### 1. Installation Docker

Avant de déployer l'application sur Docker, assurez-vous d'avoir installé Docker. Dans cet exemple, nous avons utilisé Docker Desktop pour Windows.



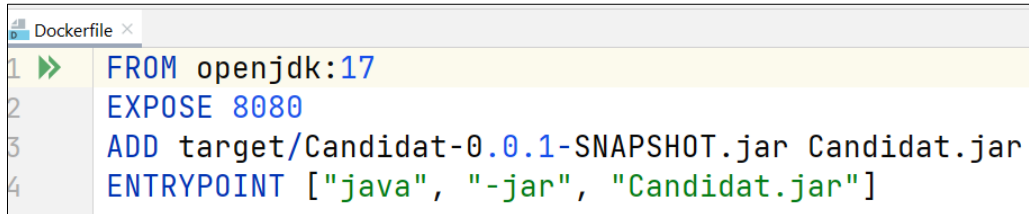
### 2. Déployer une application Spring Boot avec Docker

#### a. Création du fichier Dockerfile

- Une image Docker représente un élément important du travail avec le moteur Docker.
- Dans la structure de projet ci-dessous, nous avons créé un "Dockerfile". Docker lit les commandes / instructions de "Dockerfile" et construit l'image.

Nous commençons tout d'abord par la création d'un fichier « **dockerfile** » sous la racine du projet.

Ce fichier contient les commandes permettant de créer l'image docker à partir de notre application locale :



```
1 FROM openjdk:17
2 EXPOSE 8080
3 ADD target/Candidat-0.0.1-SNAPSHOT.jar Candidat.jar
4 ENTRYPOINT ["java", "-jar", "Candidat.jar"]
```

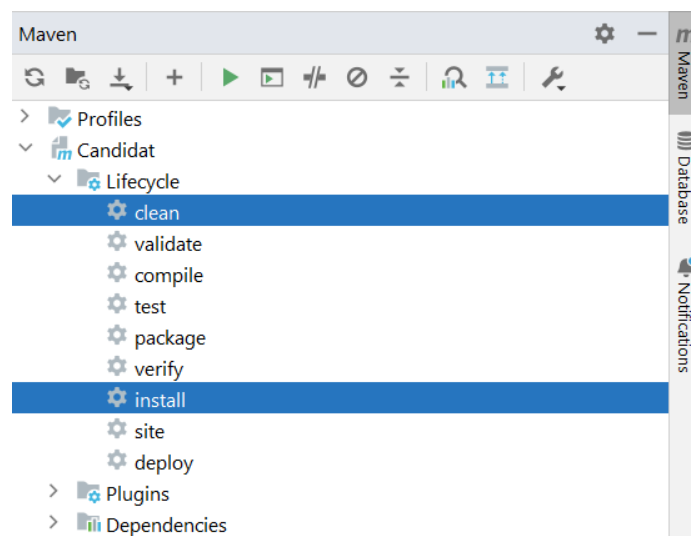
- **FROM** : Représente la première propriété dans le fichier Dockerfile. Cette commande va créer un calque à partir de l'image Docker. Dans notre cas, nous avons utilisé openjdk:17, ce qui signifie que cette application fonctionnera sous Java 17.
- **EXPOSE** : Exposer le port pour le noeud final . Dans cet exemple, nous avons utilisé 8082 en tant que port d'exécution de l'application locale.
- **ADD** : Cette commande permet de définir la source et la destination.
- **ENTRYPOINT** : C'est semblable à CMD, où le fichier de commande / jar sera exécuté.

## b. Génération du fichier .jar avec mvn install

Maintenant, exécuter la commande pour construire l'image et déployez-la sur Docker.

Avant d'exécuter la commande Docker, nous devons créer le **fichier .jar** qui encapsule l'application entière (le code compilé de l'application, ainsi que les dépendances et les ressources nécessaires).

Nous devons donc utiliser les commandes maven « **clean puis install** » pour créer ce fichier **.jar** sous le dossier **target**.



### c. Création de l'image Docker en utilisant ce fichier .jar

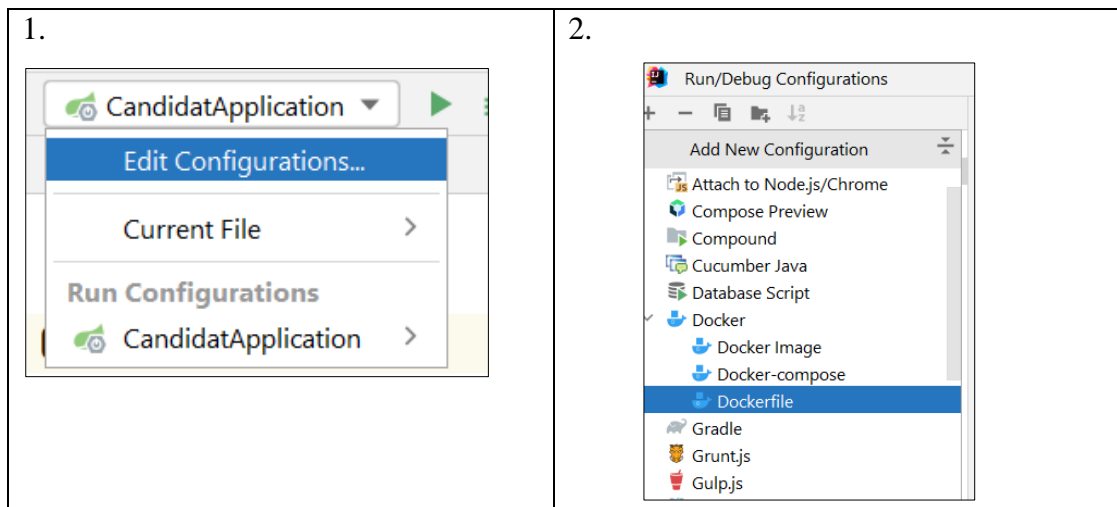
Ce fichier .jar généré va être maintenant copié dans l'image Docker et utilisé comme point d'entrée pour l'exécution de l'application lorsque le conteneur démarre.

Vous pouvez ainsi utiliser le terminal pour générer l'image. Dans ce cas lancer le cmd dans le répertoire qui contient le fichier Dockerfile, puis taper cette commande.

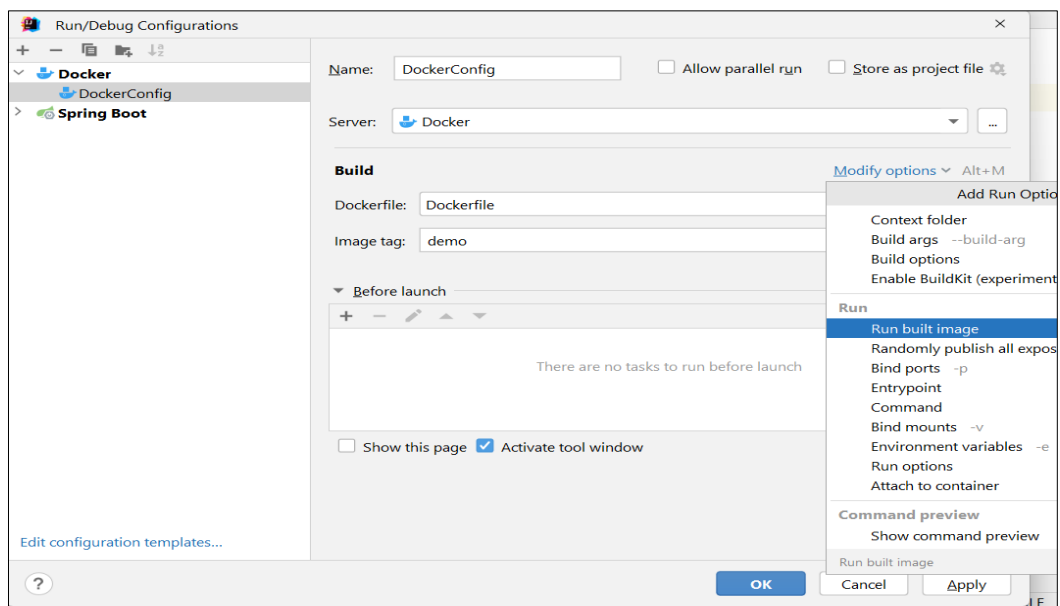
```
docker build -t demo .
```

Ou bien à travers la configuration suivante de l'IDE IntelliJ :

- i. Créer une nouvelle configuration de type Docker, en choisissant **Dockerfile** :



- ii. Nommer cette configuration, donner un nom à l'image docker à générer (**demo** dans notre cas), puis décocher l'option « **Run built image** » afin de ne pas exécuter automatiquement cette image et donc empêcher la d'une instance docker (container) :

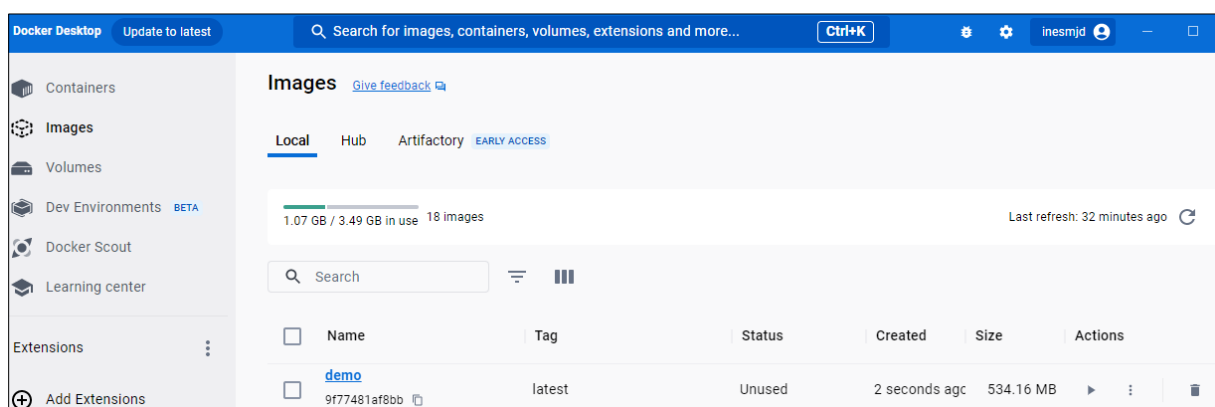


iii. Le résultat de l'exécution sera comme suit :

```
Build Log Dashboard
Step 1/4 : FROM openjdk:17
---> 5e28ba2b4cdb
Step 2/4 : EXPOSE 8080
---> Running in 545cca4409d1
Removing intermediate container 545cca4409d1
---> d88ebd184a89
Step 3/4 : ADD target/Candidat-0.0.1-SNAPSHOT.jar Candidat.jar
---> a8bca84a7d2e
Step 4/4 : ENTRYPOINT ["java", "-jar", "Candidat.jar"]
---> Running in 6670015fa272
Removing intermediate container 6670015fa272
---> 9f77481af8bb

Successfully built 9f77481af8bb
Successfully tagged demo:latest
'demo Dockerfile: Dockerfile' has been deployed successfully.
```

iv. Vérifier la création de l'image sur **Docker desktop** → **Images** → **Local** :



#### d. Dockerisation (conteneurisation) locale à travers Docker Desktop :

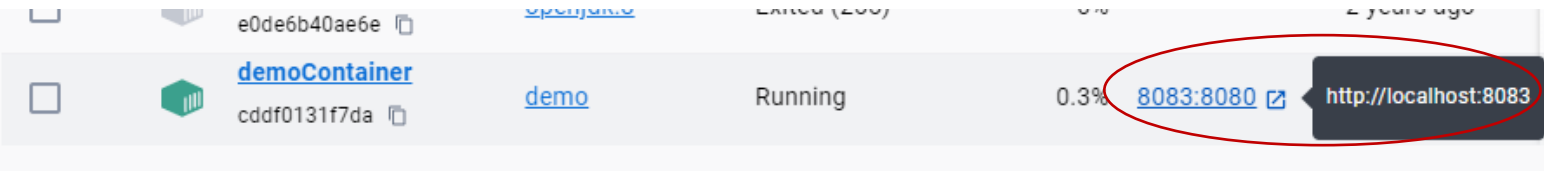
i. Vous pouvez utiliser le terminal pour générer l'image en se pointant sur le répertoire qui contient le fichier Dockerfile, puis taper cette commande.

```
docker run -p 8083 :8080 --name demoContainer demo
```

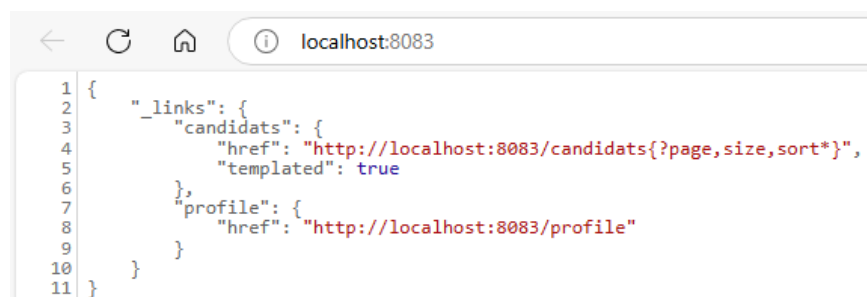
- **8083** : le port de conteneurisation sur lequel l'application s'exécutera sur docker
- **8080** : le port d'exécution de l'application local
- **demoContainer** : le nom de l'instance exécutable (container) à générer et qui va être exécutée sur le port 8083

- **demo** : le nom de l'image docker créé dans l'étape précédente

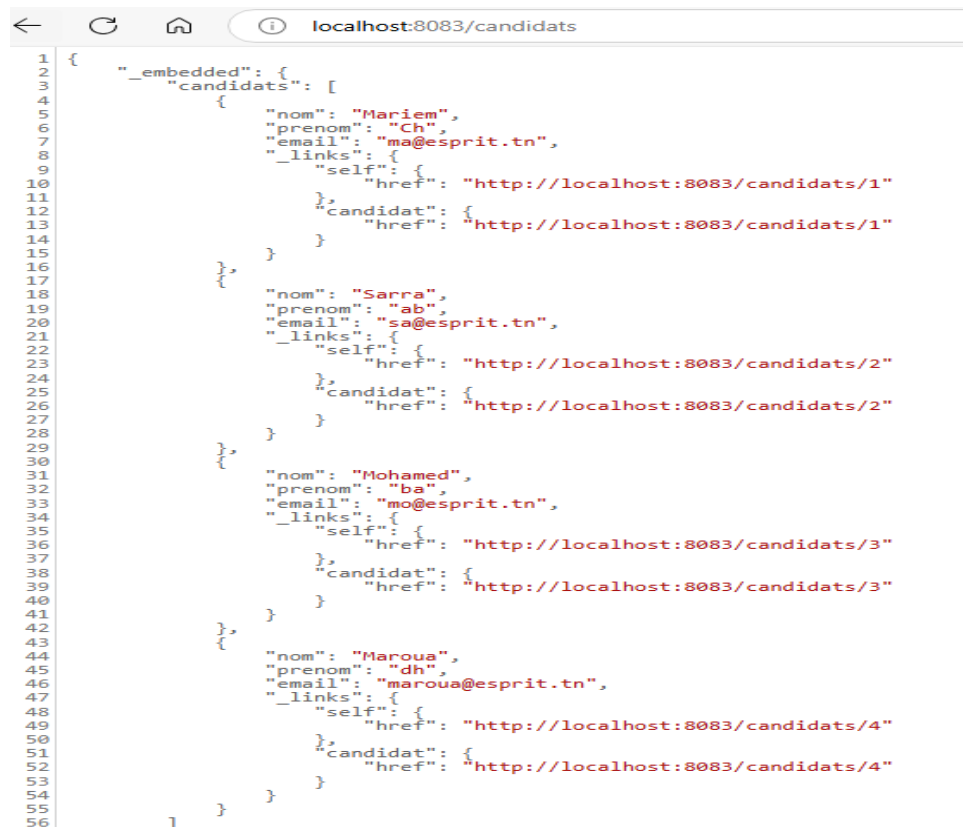
La ligne suivante sera ajoutée sous **Docker desktop → Containers**



En cliquant maintenant sur le lien ci-dessus « **8083 :8080** » vous allez être redirigé vers la page web suivante :



Il suffit d'ajouter le path **/candidats** à la fin de cet URL afin d'afficher la liste des candidats :

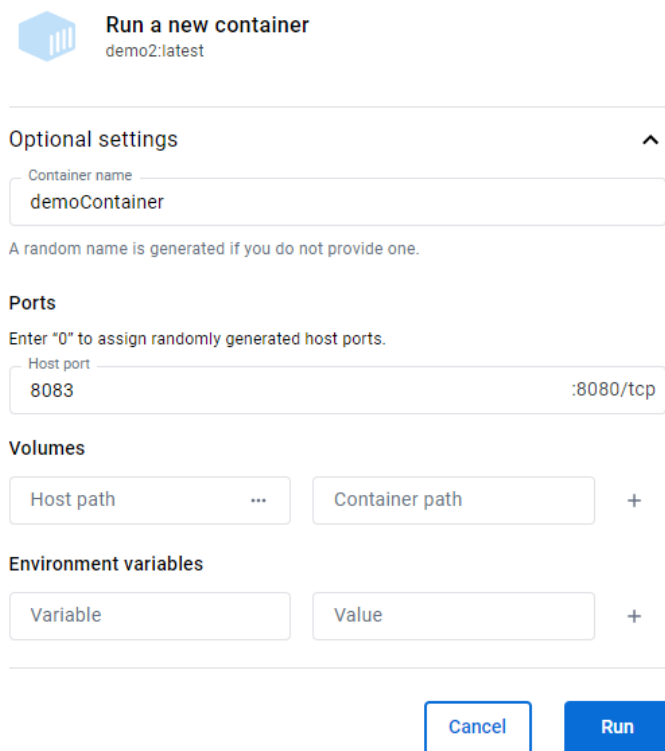


Tous les autres ressources REST sont maintenant accessibles à partir de cette instance Docker via le port **8083**.

ii. On peut aussi l'outil Docker desktop :

**Docker desktop → Images → Local →** cliquer sur l'action **run** devant l'image docker « demo » déjà créée afin de l'exécuter.

Fixer maintenant les paramètres de conteneurisation tel que le nom de l'instance à créer et le port d'exécution sur docker :



**Run a new container**  
demo2:latest

**Optional settings** ^

Container name  
demoContainer

A random name is generated if you do not provide one.

**Ports**  
Enter "0" to assign randomly generated host ports.

Host port  
8083 :8080/tcp

**Volumes**

Host path ... Container path +

**Environment variables**

Variable Value +

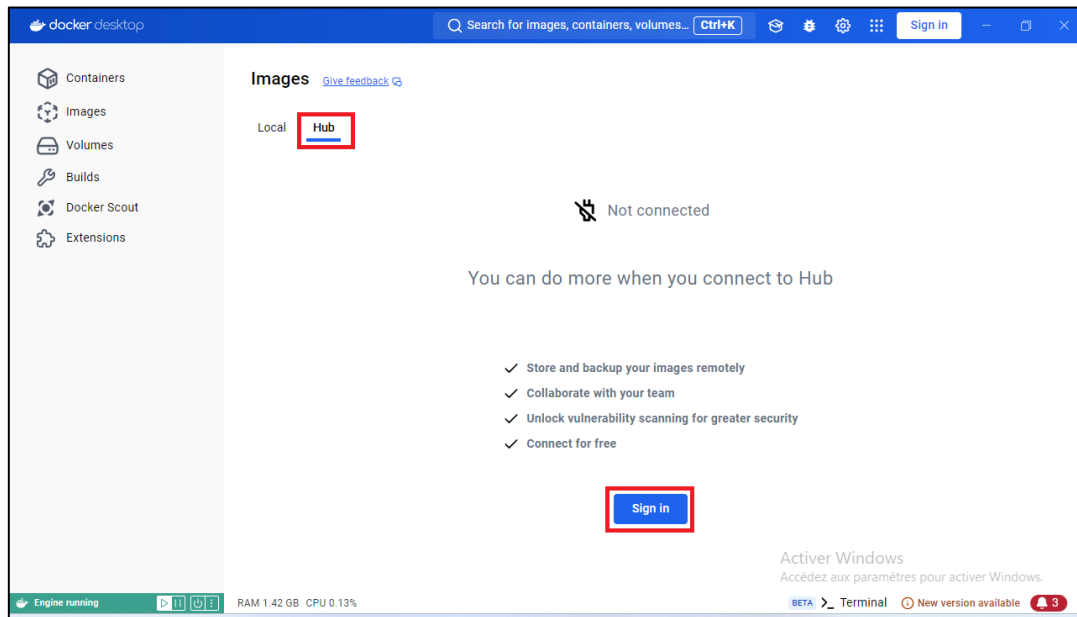
Cancel Run

Et l'instance exécutable (container) « **demoContainer** » sera générée et exécutée sous le port 8083.

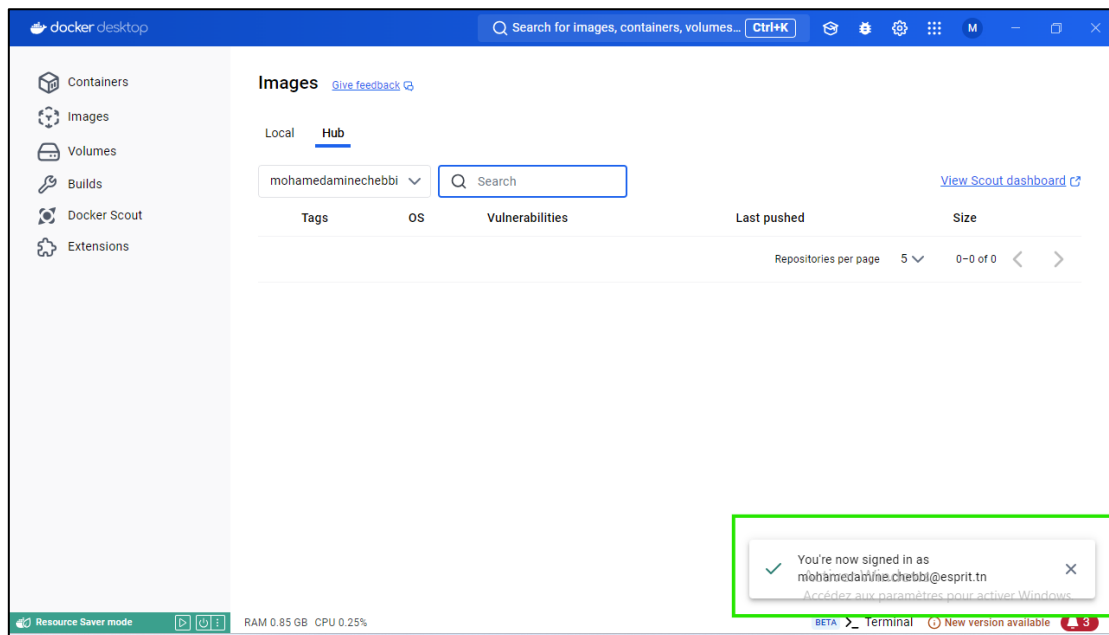
## Partie II : Dockerisation avec docker Hub

Dans cette partie nous allons voir comment partager une image sur docker hub, et comment utiliser l'outil [PWD](#) pour la en ligne.

Afin de se connecter au « **docker hub** », lancer docker desktop puis cliquer sur le lien « **hub** », par la suite cliquer sur « **sign in** », puis taper vos coordonnées (Username + Password).

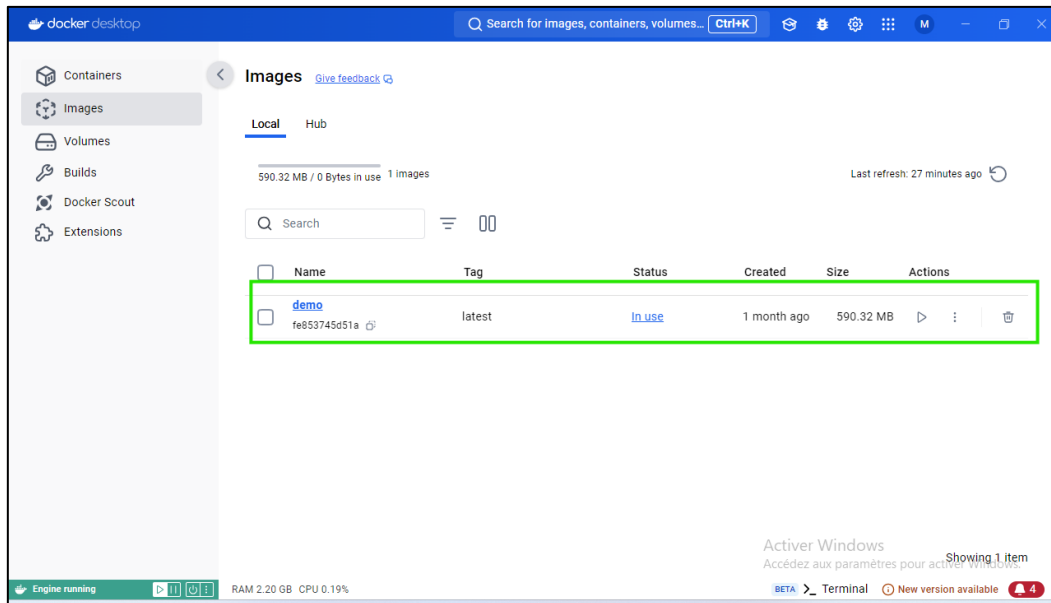


Si la connexion est établie, il faut avoir cette fenêtre :



Maintenant vous allez télécharger (**push**) une image vers votre **Docker hub**.

L'image qu'on va utiliser pour cette étape est appelée « **demo** » (même image qu'on a utilisé dans la première partie en local).



Assurez-vous que vous êtes connecté sur **Docker hub**. Pour vérifier vous pouvez utiliser un terminal cmd en tapant la commande « **docker login** ».

```
Administrateur : C:\Windows\
C:\Users\Mohamed Amine\Downloads\CandidatProject (2)\CandidatProject>docker login
Authenticating with existing credentials...
Login Succeeded
C:\Users\Mohamed Amine\Downloads\CandidatProject (2)\CandidatProject>
```

Maintenant pour envoyer l'image sur le compte **docker hub** tapez la commande :

**docker push demo**

```
Administrateur : C:\Windows\
C:\Users\Mohamed Amine\Downloads\CandidatProject (2)\CandidatProject>docker login
Authenticating with existing credentials...
Login Succeeded

C:\Users\Mohamed Amine\Downloads\CandidatProject (2)\CandidatProject>docker push demo
Using default tag: latest
The push refers to repository [docker.io/library/demo]
88ea280d9ddd: Pushed
6b5aaff44254: Pushed
6b5aaff44254: Pushing [=====>] 133MB/209.2MB
53a0b163e995: Pushed
9b55156abf26: Pushed
9b55156abf26: Pushing [=====>] 130.5MB/152MB
293d5db30c9f: Pushed
9c742cd6c7a5: Pushed
1.0.0: digest: sha256:8a72576cd88c896ce3e219ce7d5a6ddb6f5abea2ed49561d8ab71e55cffe8f8bc size: 2007
```



**En cas où vous recevrez un message d'erreur**, c'est que vous devez « taguer » votre image avant de la pousser « push » sur docker hub :

**i. Taguer l'image docker générée :**

Le tagging est une étape importante pour pousser une image vers Docker Hub. Il permet d'identifier l'image et de spécifier où elle doit être stockée dans Docker Hub.

Lancer la commande suivante pour faire le push de votre image vers un repository sous votre espace docker Hub :

```
docker tag <image_name> <username>/<repository_name>:<tag>
```

où :

- **<image\_name>** : le nom de l'image Docker
- **<username>** : Votre nom d'utilisateur Docker Hub
- **<repository\_name>** : Le nom du dépôt sur Docker Hub, qui peut avoir généralement le même nom de l'image docker
- **<tag>** : Le tag que vous voulez donner à l'image (ex. latest ou v1.0.0, etc).

Dans notre cas, la commande à passer est la suivante :

```
docker tag demo mohamedaminechebbi/demo:1.0.0
```

**ii. Pousser l'image taguée vers le docker hub :**

Une fois l'image correctement taguée, vous pouvez la pousser vers Docker Hub avec la commande suivante :

```
docker push <username>/<repository_name>:<tag>
```

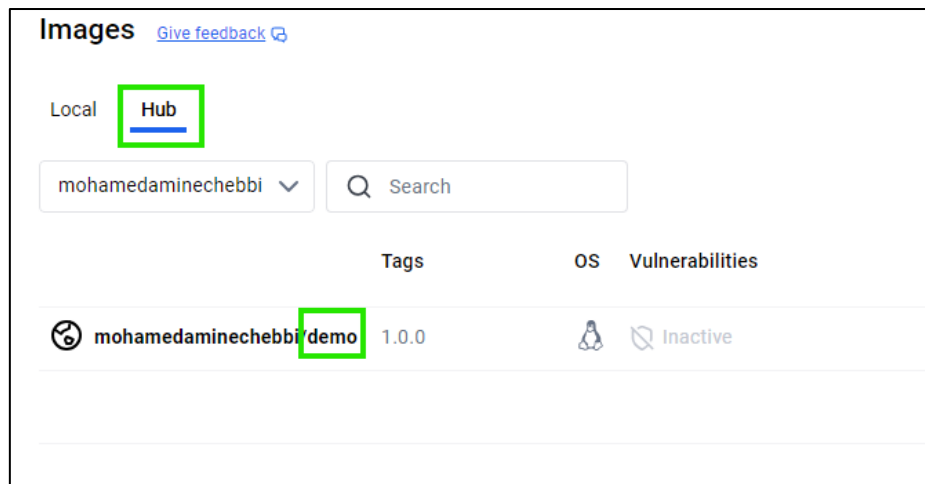
Dans notre cas, on lancera la commande suivante :

```
docker push mohamedaminechebbi/demo:1.0.0
```

Docker va commencer à envoyer (**push**) votre image vers Docker Hub.

NB : Cela peut prendre un certain temps en fonction de la taille de l'image et de votre connexion Internet.

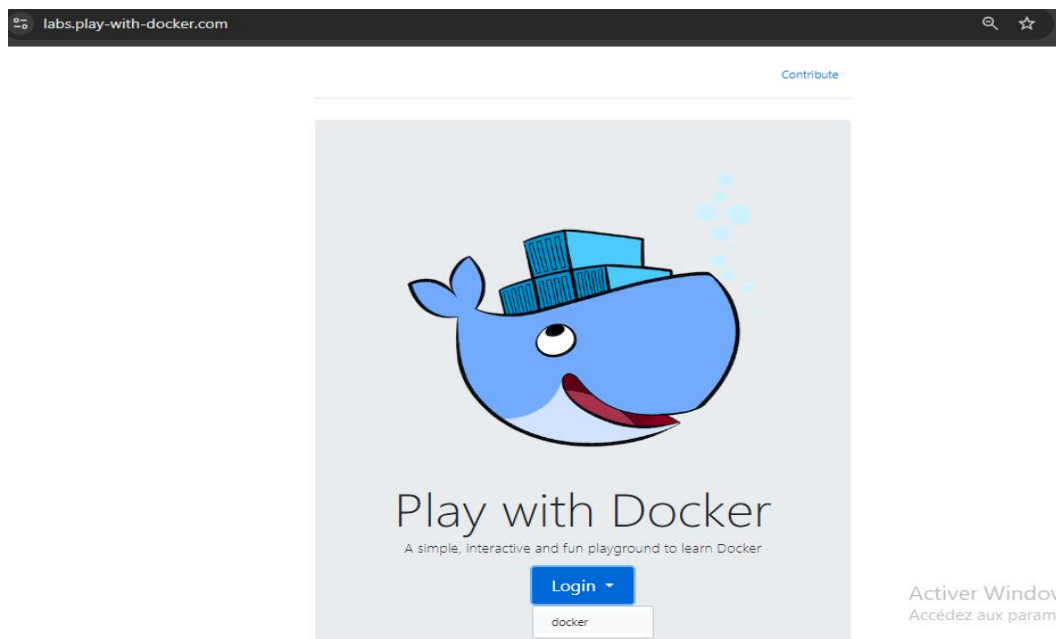
L'image est maintenant sur « **docker hub** » :



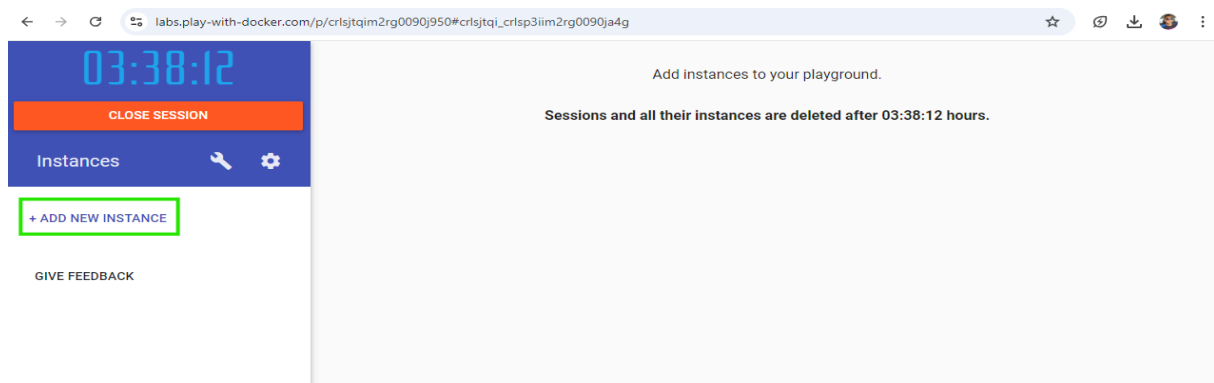
Pour tester l'image en ligne on va utiliser la plateforme « **Play with docker (PWD)** ».

Cliquer sur l'adresse [PWD](#) pour accéder à la plateforme, puis cliquer sur « **login** » puis choisir « **docker** ».

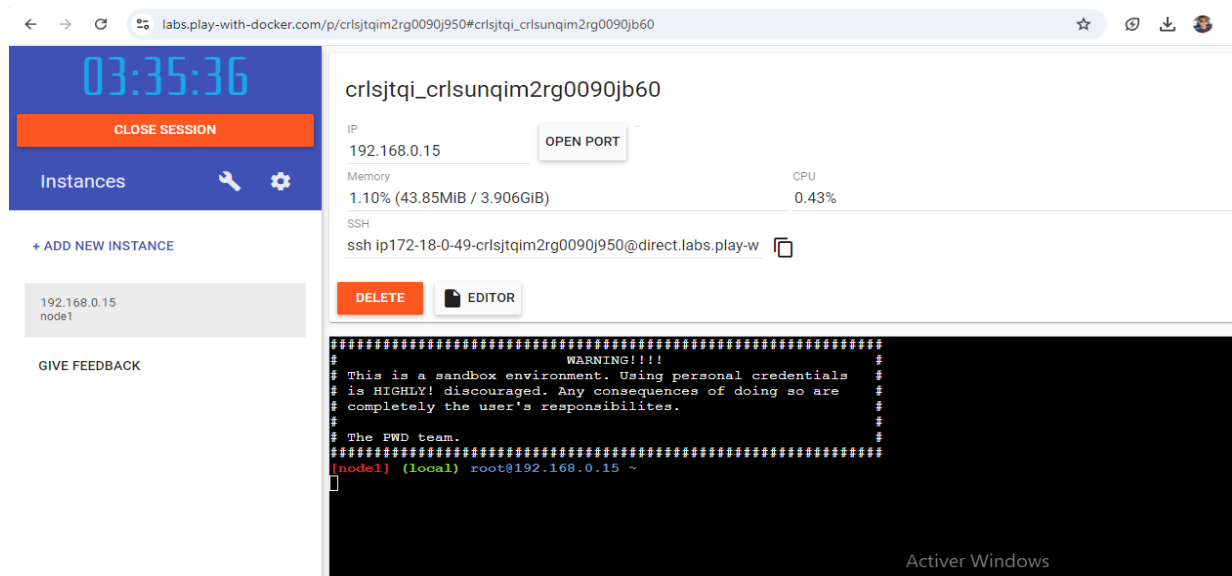
**NB** : Vous devez utiliser les mêmes paramètres de connexion à « **docker hub** ».



Ensuite, cliquer sur « **+ADD NEW INSTANCE** ».



Un terminal sera affiché juste en bas à droite du navigateur.

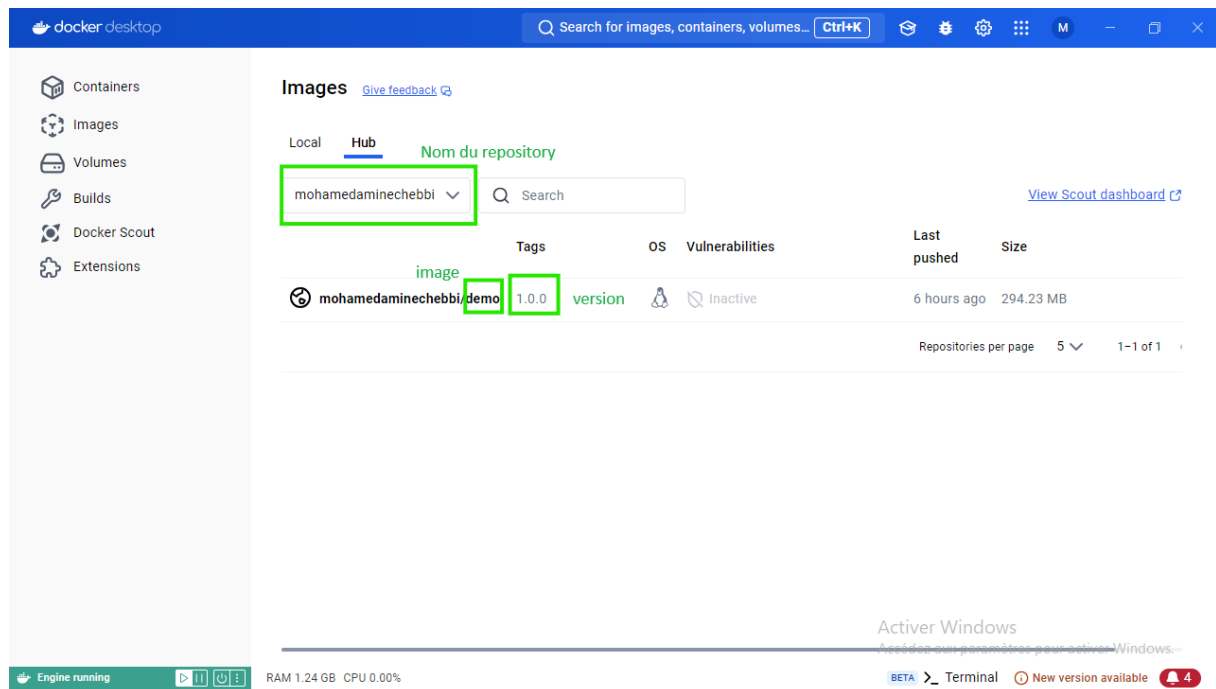


Dans le terminal qui s'affichera, écrire la commande suivante :

**Docker run -p 8082 :8080 -d votreRepository/image :version**

**NB :**

- **votreRepository** : est le nom du repository dans docker hub.
- **image** : est le nom de l'image sur docker hub.
- **version** : est la version de l'image dans docker hub.



Donc la commande finale devient :

```
Docker run -p 8082:8080 -d mohamedaminechebbi/demo:1.0.0
```

```
$ docker run -p 8082:8082 -d mohamedaminechebbi/demo:1.0.0
Unable to find image 'mohamedaminechebbi/demo:1.0.0' locally
1.0.0: Pulling from mohamedaminechebbi/demo
001c52e26ad5: Pull complete
d9d4b9b6e964: Pull complete
2068746827ec: Pull complete
9daef329d350: Pull complete
d85151f15b66: Pull complete
52a8c426d30b: Pull complete
8754a66e0050: Pull complete
1787106a4104: Pull complete
Digest: sha256:8a72576cd88c896ce3e219ce7d5a6ddb6f5abea2ed49561d8ab71e55cffeef8bc
Status: Downloaded newer image for mohamedaminechebbi/demo:1.0.0
401d0db0ab272cbb432620af83bdfed2bf6227e5c1a1b49c7e70427a1f1c59e9
[node1] (local) root@192.168.0.15 ~
$
```

Pour vérifier que l'image a été téléchargée sur l'environnement du test « [PWD](#) » tapez :  
« **docker image ls** »

```
[node1] (local) root@192.168.0.15 ~
$ docker image ls
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
mohamedaminechebbi/demo  1.0.0      fe853745d51a  5 weeks ago  590MB
[node1] (local) root@192.168.0.15 ~
$
```

En fin pour tester le micro-service à partir de l'instance docker-hub, cliquer sur le port qui s'affiche dans la page web :

### crsljtqi\_crslsunqim2rg0090jb60

IP

192.168.0.15

OPEN PORT

8082

Memory

32.61% (1.274GiB / 3.906GiB)

CPU

0.95%

SSH

ssh ip172-18-0-49-crsljtqim2rg0090j950@direct.labs.play-w

DELETE

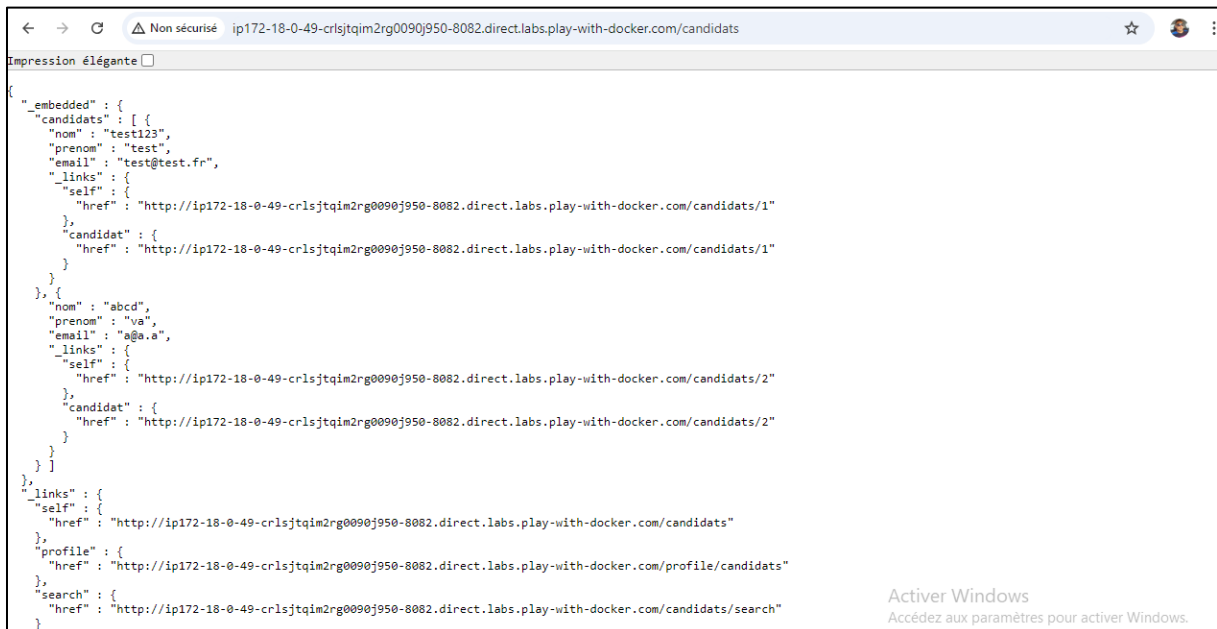
EDITOR

Le micro-service candidat sera affiché :



Ajouter **/candidats** à l'adresse web pour voir la liste des candidats

<http://ip172-18-0-49-crlsjtqm2rg0090j950-8082.direct.labs.play-with-docker.com/candidats>



Pour chercher un candidat par id ajouter l'identifiant du candidat à la fin de l'URL

