**Docker**

Docker permet de créer des conteneurs applicatifs à partir d’images.

**Pour créer un conteneur :**

docker container run --publish <port:port> -d --name <nom\_conteneur> <image>

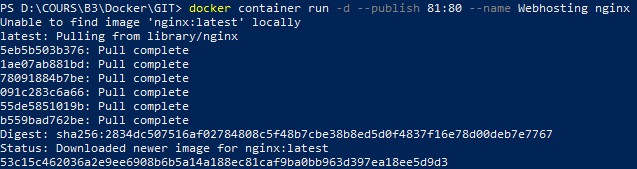
container run : permet de créer le conteneur

--publish : permet de déterminer le port utilisé par le conteneur

-d ou –detach : permet de détacher l’invite de commande du conteneur, pour ne pas le figer

--name : permet de nommer le conteneur

<image> : l’image utilisée par le conteneur. Si elle n’est pas contenue en local, elle va être téléchargée



Ici mon container a bien été créé. On peut voir le status des conteneurs avec la commande suivante :

docker ps



**Pour arrêter un conteneur :**

Docker stop <Id ou nom>

Exemple : docker stop 53





**Pour lancer un conteneur :**

docker start <Id ou nom>

Exemple : docker start 53





**Pour supprimer un conteneur :**

docker rm <ID ou nom>

Le conteneur doit être stoppé au préalable

Exemple : docker rm 53





**Pour afficher les images stockées localement :**

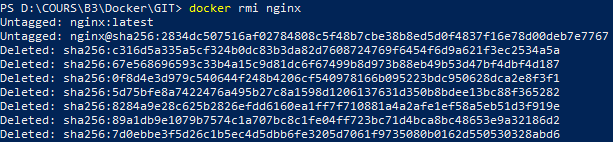
docker images



**Pour supprimer une image :**

docker rmi <ID ou nom>

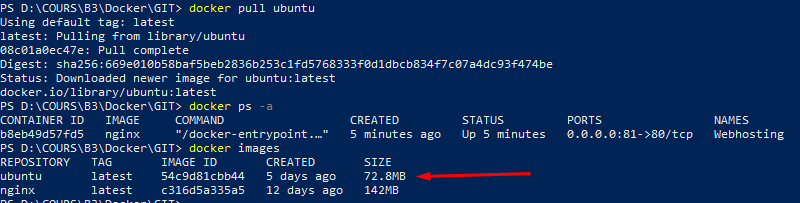
Exemple : docker rmi nginx



**Pour télécharger une image sans installer de conteneur :**

docker pull <image>

Exemple : docker pull ubuntu

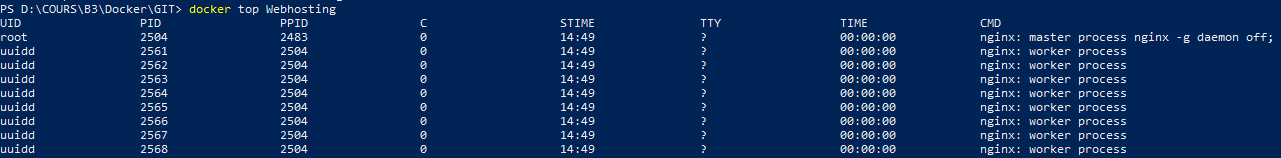


L’image est bien téléchargée mais le conteneur n’est pas créé.

**Commande TOP :**

docker top <ID ou Nom>

Exemple : docker top Webhosting



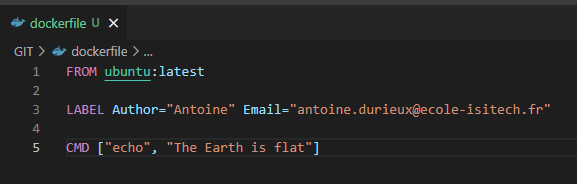
Permet d’afficher les processus actifs pour ce conteneur

**DOCKERFILE**

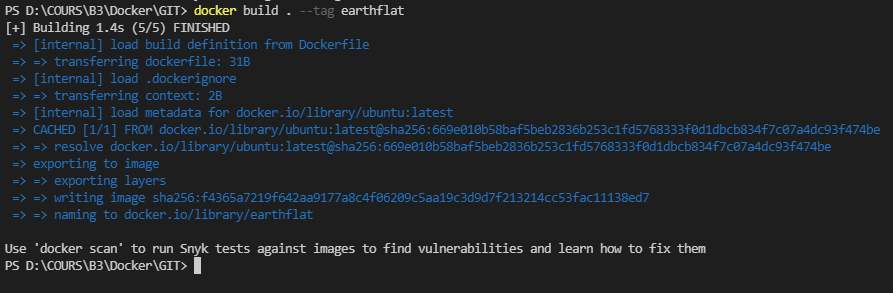
Un Dockerfile est un fichier que l’on va créer nous permettant de générer une image. L’intérêt du dockerfile est d’automatiser la création d’image, puis passer des commandes.

Nous pouvons créer un dockerfile à partir de visual studio Code.

Dans notre exemple, nous allons créer un dockerfile pour générer une image pour ensuite créer un conteneur ubuntu qui nous affichera le message « The Earth is flat ».



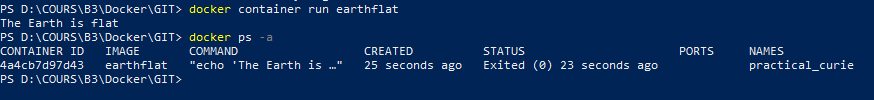
Une fois notre dockerfile prêt, nous allons effectuer un docker build afin de générer notre image.



Notre docker build s’est terminé sans erreur. Nous avons utilisé l’argument « --tag » afin de nommer notre image générée. Avec la commande « docker images », nous pouvons retrouver l’image que nous venons de générer.



Nous allons maintenant construire un conteneur à partir de notre image. Pour cela, nous allons utiliser la commande « docker container run »



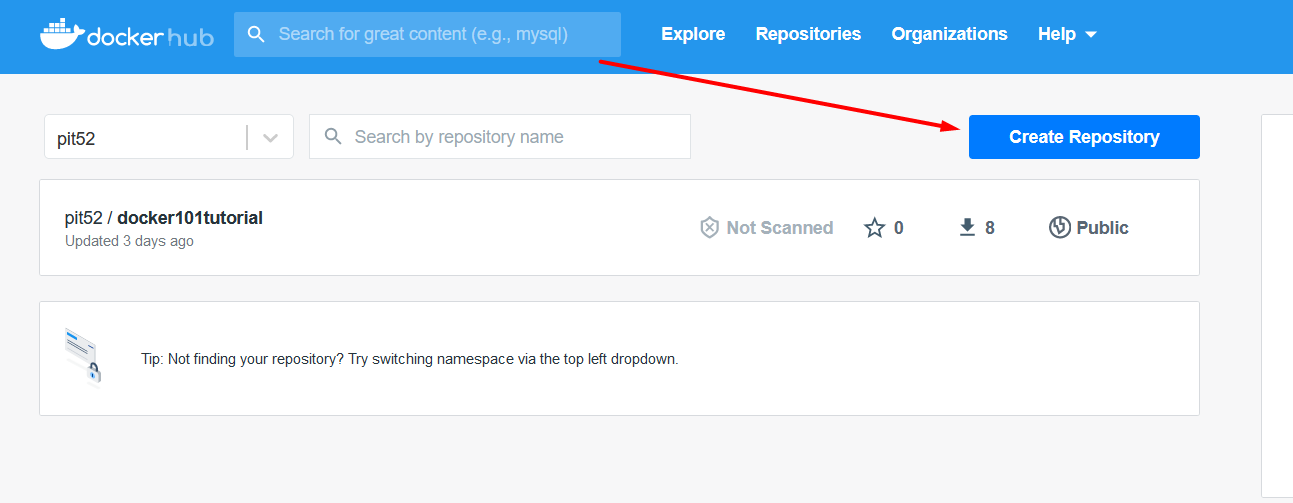
Nous pouvons voir que le texte « The Earth is flat » est bien affiché et que le conteneur est bien créé.

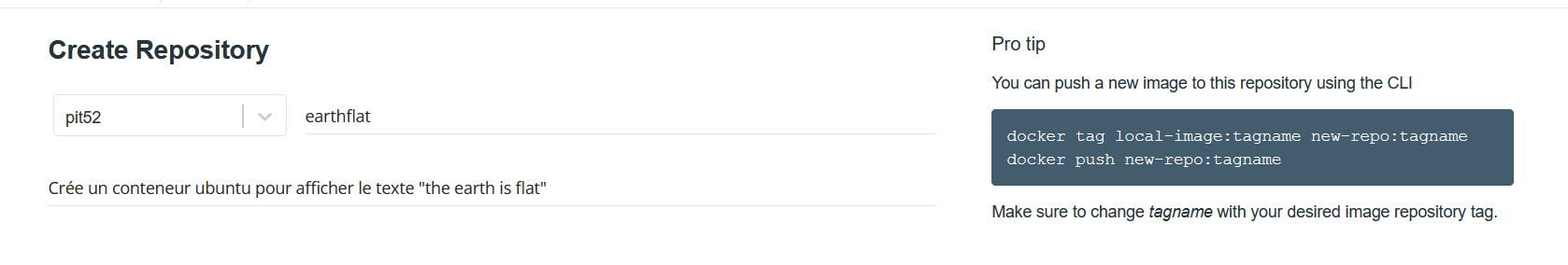
**Push d’une image générée sur le docker hub**

Une fois que notre image a été générée à partir de notre dockerfile, nous avons la possibilité de la push vers le docker hub. Voici les différentes étapes pour effectuer cela.

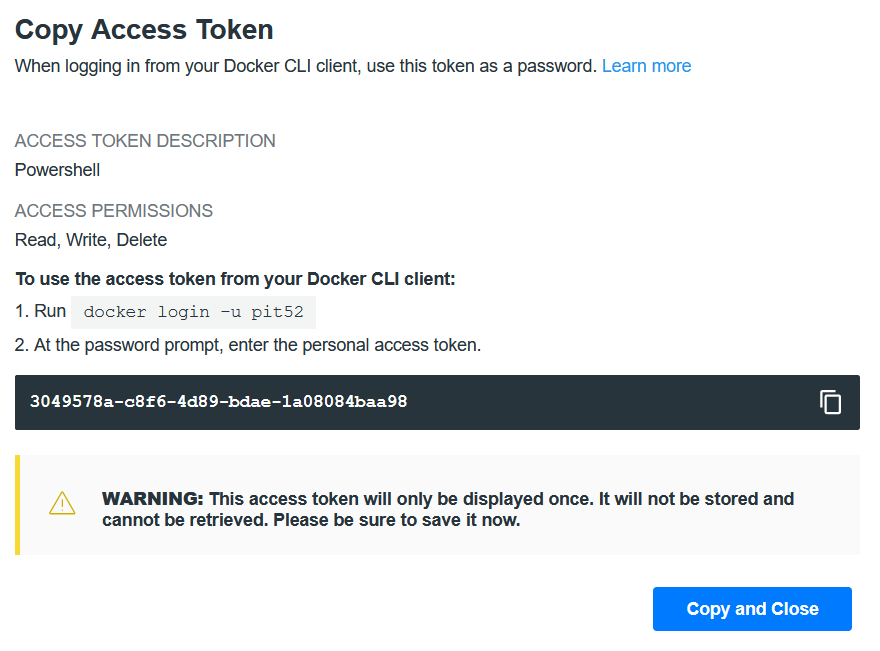
Dans un premier temps, nous allons nous créer un compte sur <https://hub.docker.com/>

Une fois cela fait, il faut créer un Repository

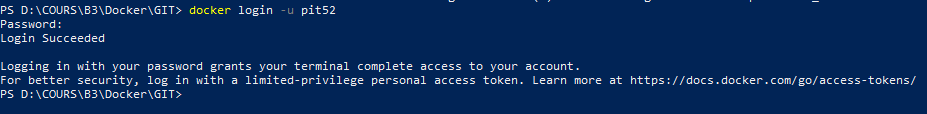


Dedans, nous allons mettre un nom et une description à notre repository

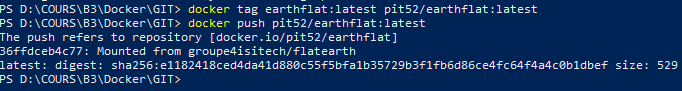
Il faut ensuite générer un access token afin d’autoriser notre powershell à push notre image sur notre repository de Docker Hub.



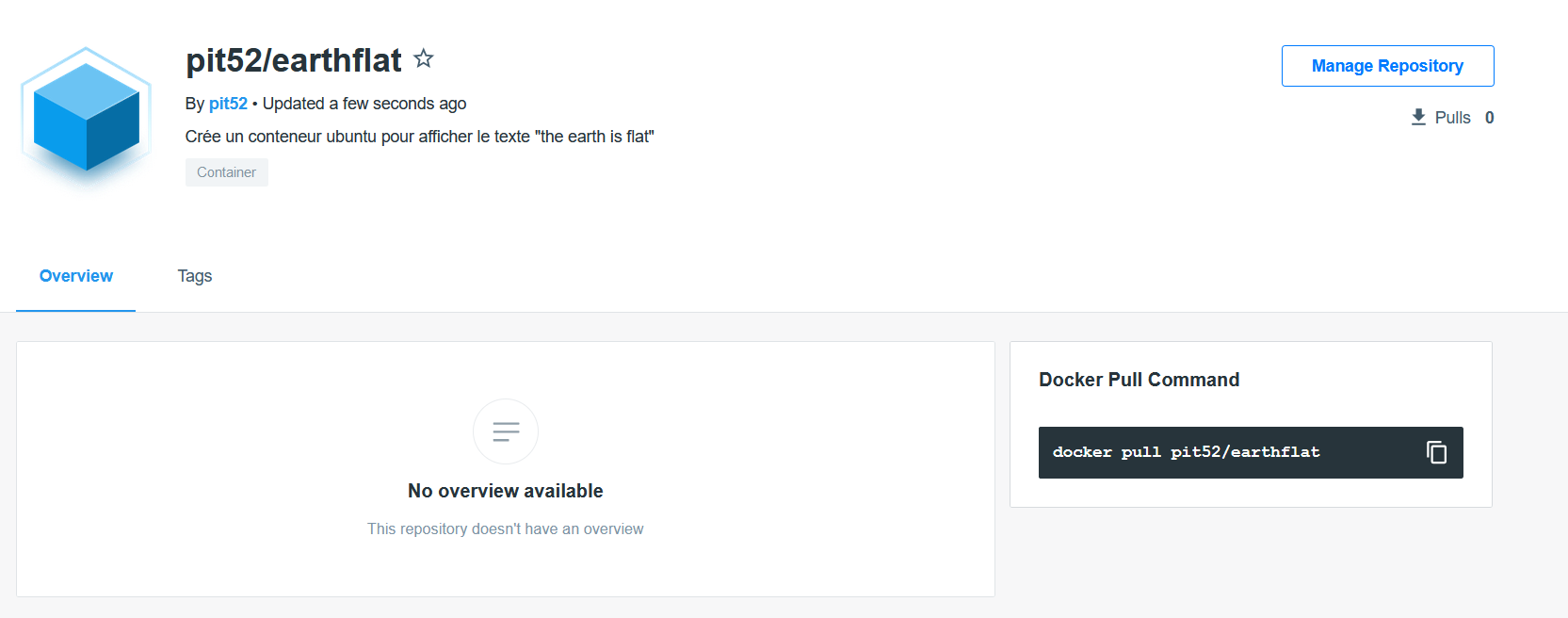
Nous allons ensuite exécuter la commande fournie dans notre powershell.



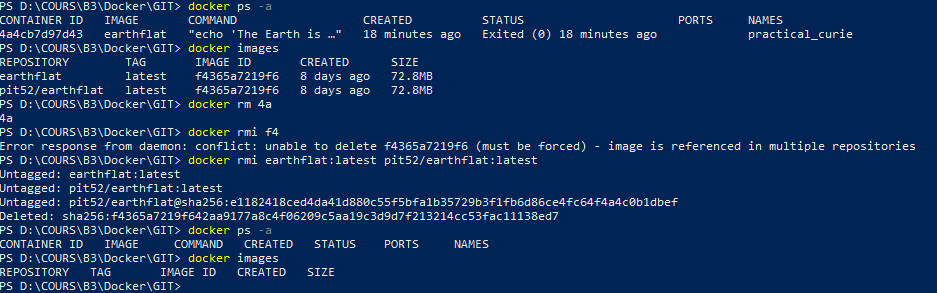
Nous avons bien réussi à nous logger. Nous allons maintenant push notre image.



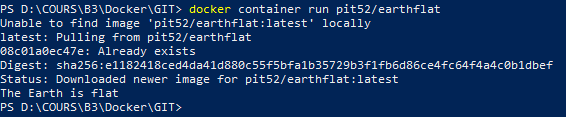
Nous pouvons voir que notre push a bien fonctionné : <https://hub.docker.com/r/pit52/earthflat>



Nous allons maintenant le tester. Pour cela nous allons supprimer tous les conteneurs et toutes les images que nous avons localement.



Nous allons maintenant créer un conteneur à partir de notre image que nous avons push sur docker hub.



Nous pouvons voir que l’image n’existe pas en local, et qu’elle a bien été pull depuis le docker hub. Le container a bien été créé.

**Instructions d’un dockerfile**

FROM : Image à partir de laquelle nous allons partir

MAINTAINER : Auteur du dockerfile

ENV : Permet de définir des variables d’environnement dans notre dockerfile

LABEL : Ajout de métadonnées

VOLUME : Crée un point de montage dans notre container

RUN : Permet de lancer des commandes pour construire l’image

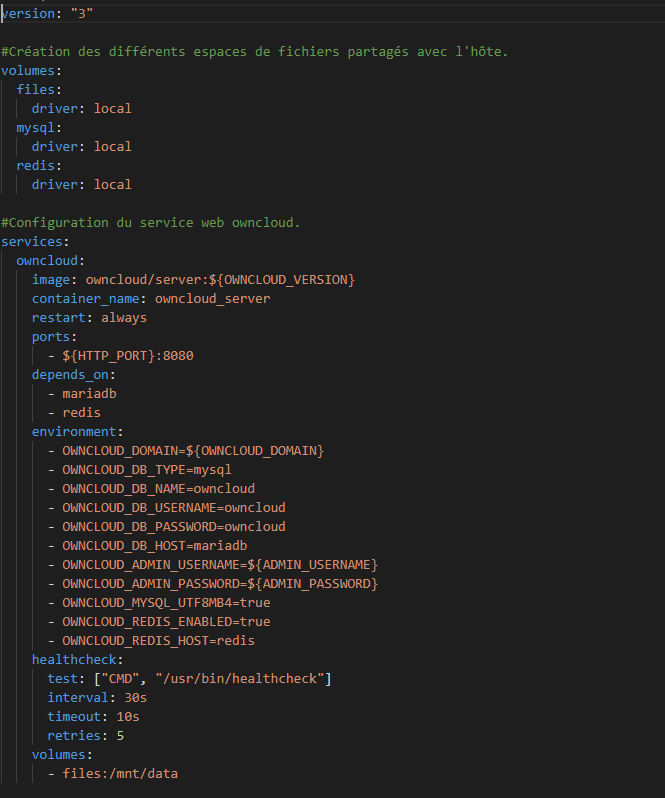
EXPOSE : Permet de définir un ou des ports sur lesquels va écouter notre conteneur

CMD : Permet d’exécuter une commande au démarrage du conteneur

**DOCKER COMPOSE**

La commande docker-compose permet de créer des conteneurs qui vont fonctionner ensemble à partir d’un fichier .yml

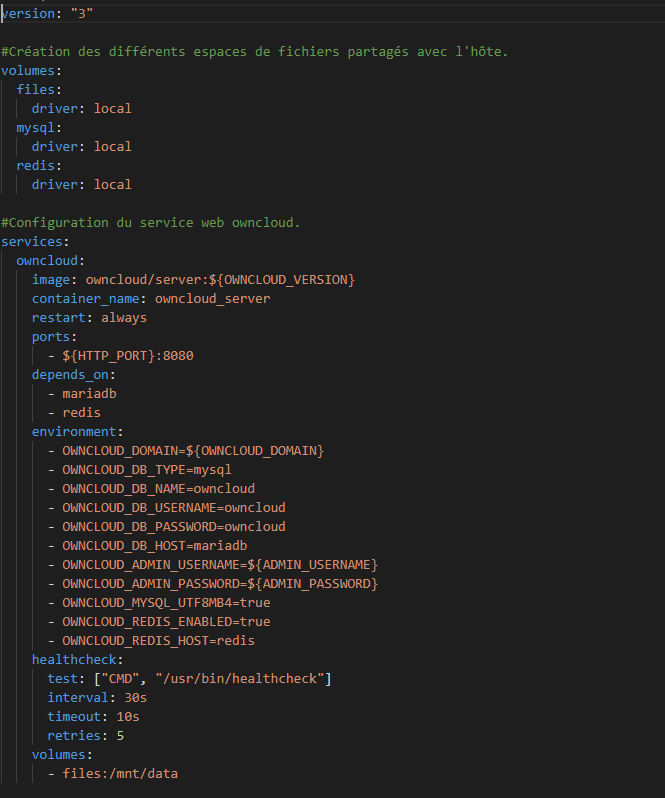
Exemple d’un fichier .yml :



Les différentes instructions d’un fichier .yml seront spécifiées dans notre projet.

**PROJET OWNCLOUD**

Tout d’abord, nous avons commencé par créer notre fichier .yml. Qui est le suivant.



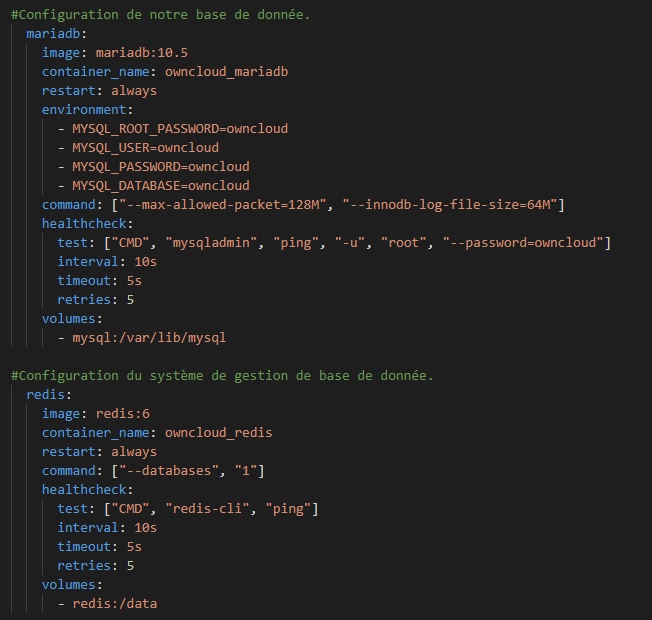


Image : récupère l’image du container que l’on souhaite utiliser. (Owncloud, mariadb, redis)

Container\_name : nommage du container.

Restart : Définit à quel moment démarre le service.

Ports : Mappage du port.

Depends\_on : De quel autre container ce service dépend.

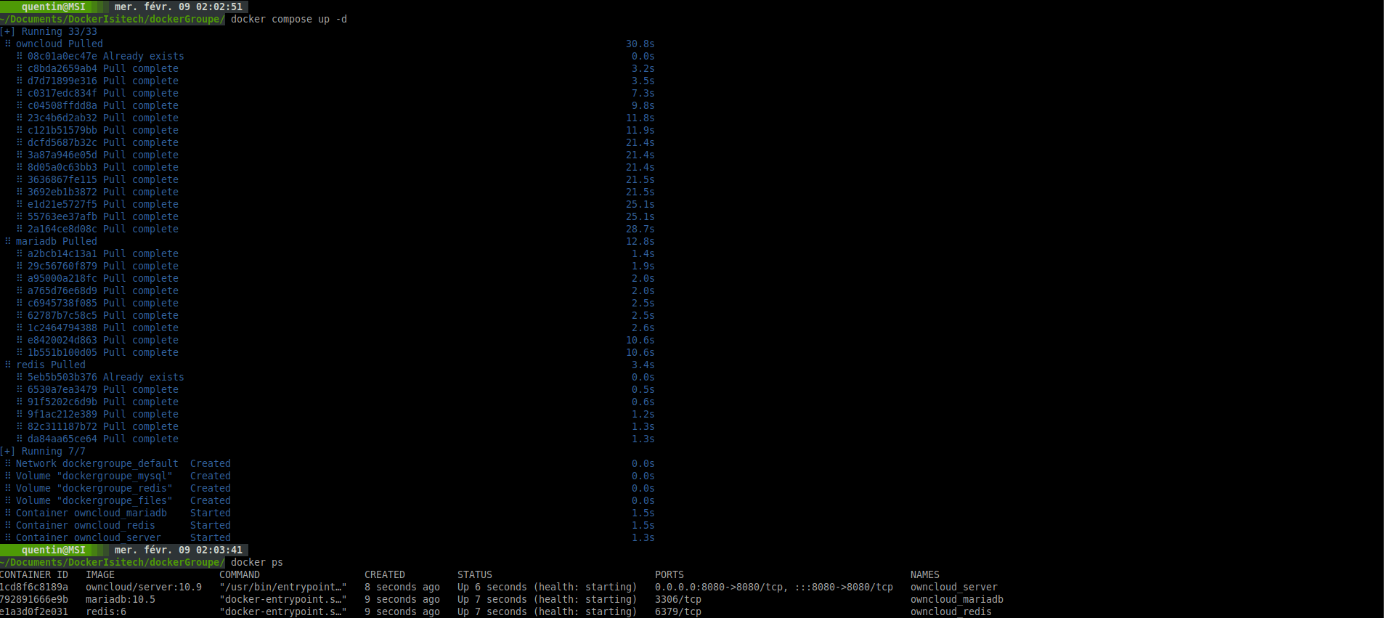
Environment : définition des variables d’environnement.

Healthcheck : Check de l’état du container.

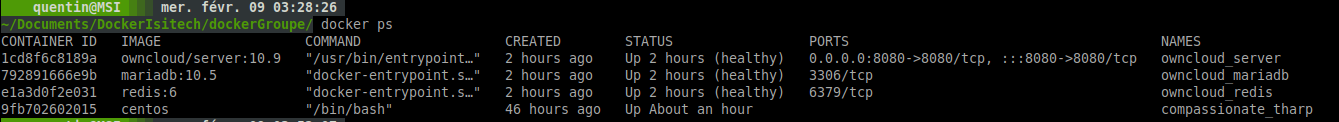
Volumes : Définition de l’emplacement du volume de fichier partagé.

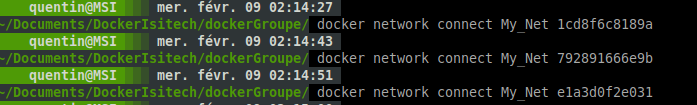
Command : Permet de rajouter des arguments lors de la création du container

Là nous retrouvons la commande docker compose up -d.

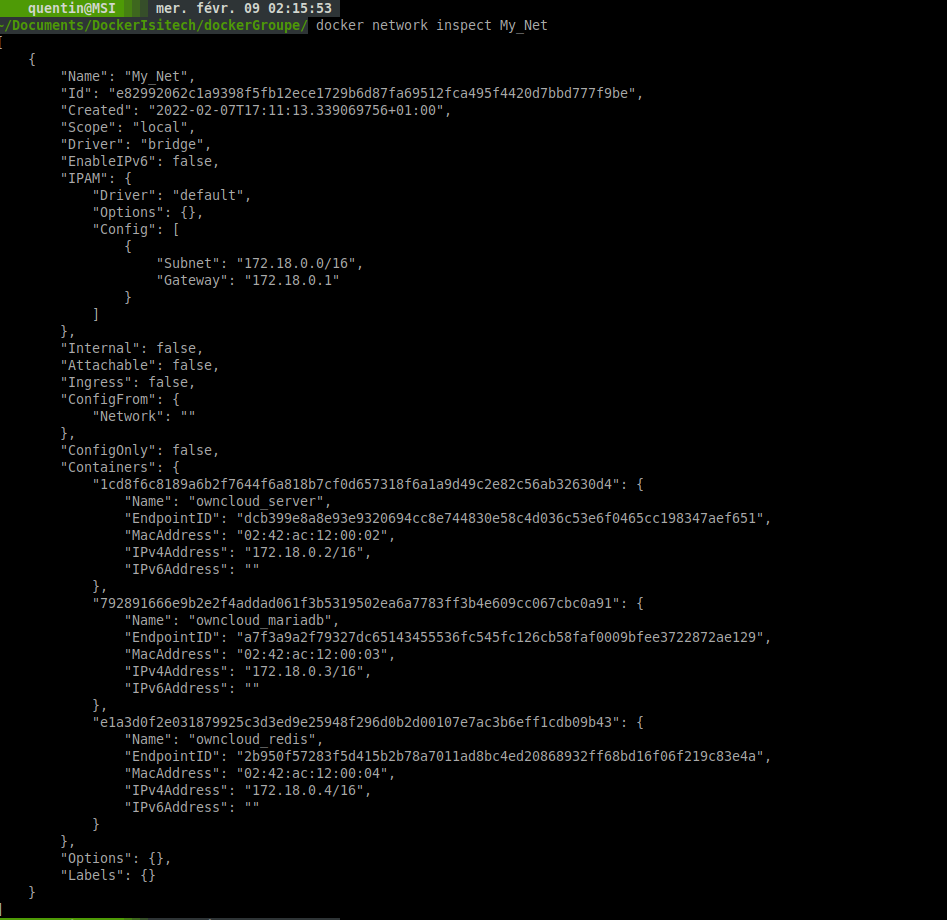
Cette commande une foi lancée, nous exécute notre fichier .yml et donc crée nos différents container avec les paramètres précédemment établis.

On peut voir dans la capture d’écran suivante que nos containers ont bien été créés.

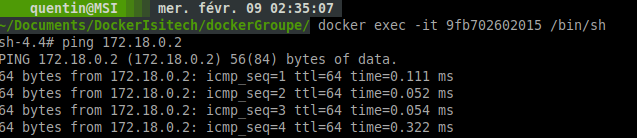


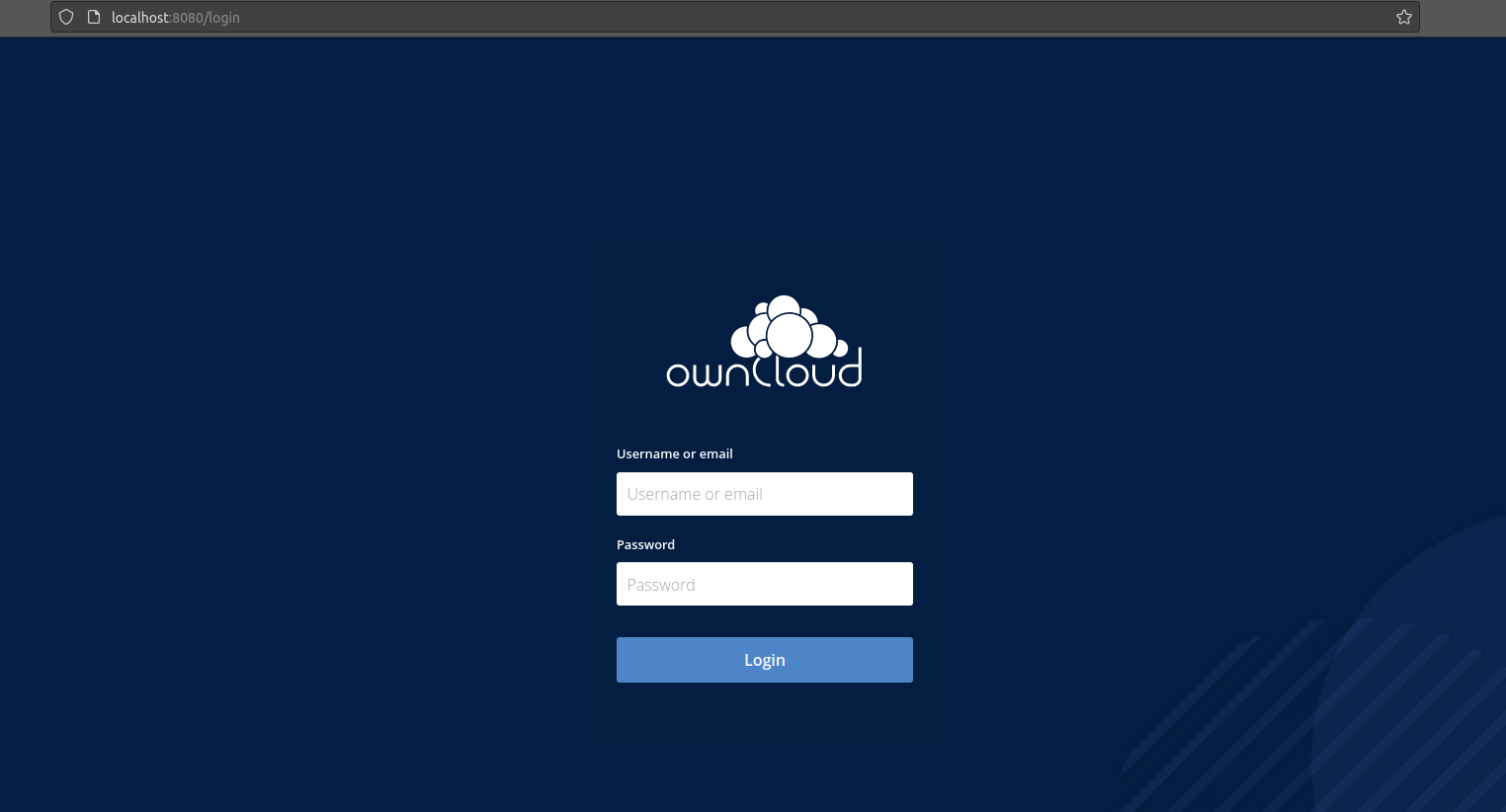
Nous avons connectés nos containers à un réseau virtuel commun précédemment créés via la commande : docker network create My\_Net

Voici le détail du réseau virtuel My\_Net que nous avons créé.



En apportant un container CentOS dans le réseau virtuel nous avons pu constater qu’il était possible de ping les autres machines.



Enfin nous avons pu constater que notre owncloud fonctionne avec succès car nous arrivons à accéder à la page web.

Pour essayer, nous avons créé des utilisateurs via l’interface web.

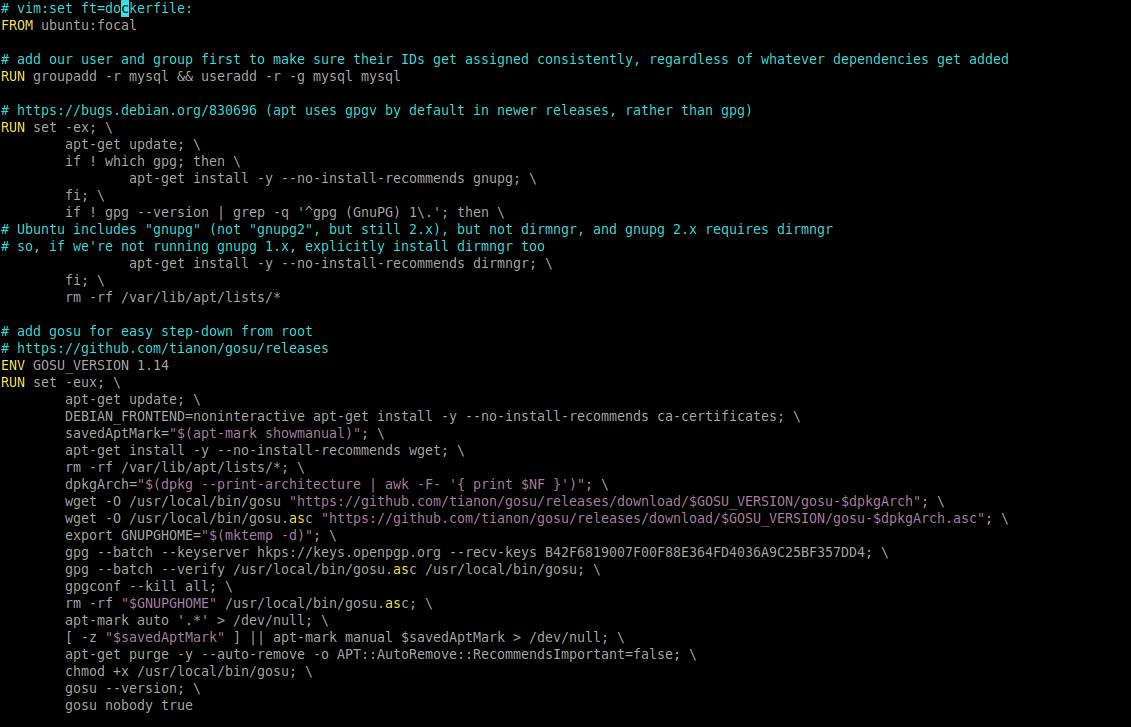


**Projet Final Groupe 4**

Tout d’abord nous avons tenté de créer nos différentes images à partir de Dockerfile.

Notre owncloud étant composé d’une partie base de données et d’une partie Web, nous avons commencé par créer l’image de notre base donnée SQL.

Voici le dockerfile :

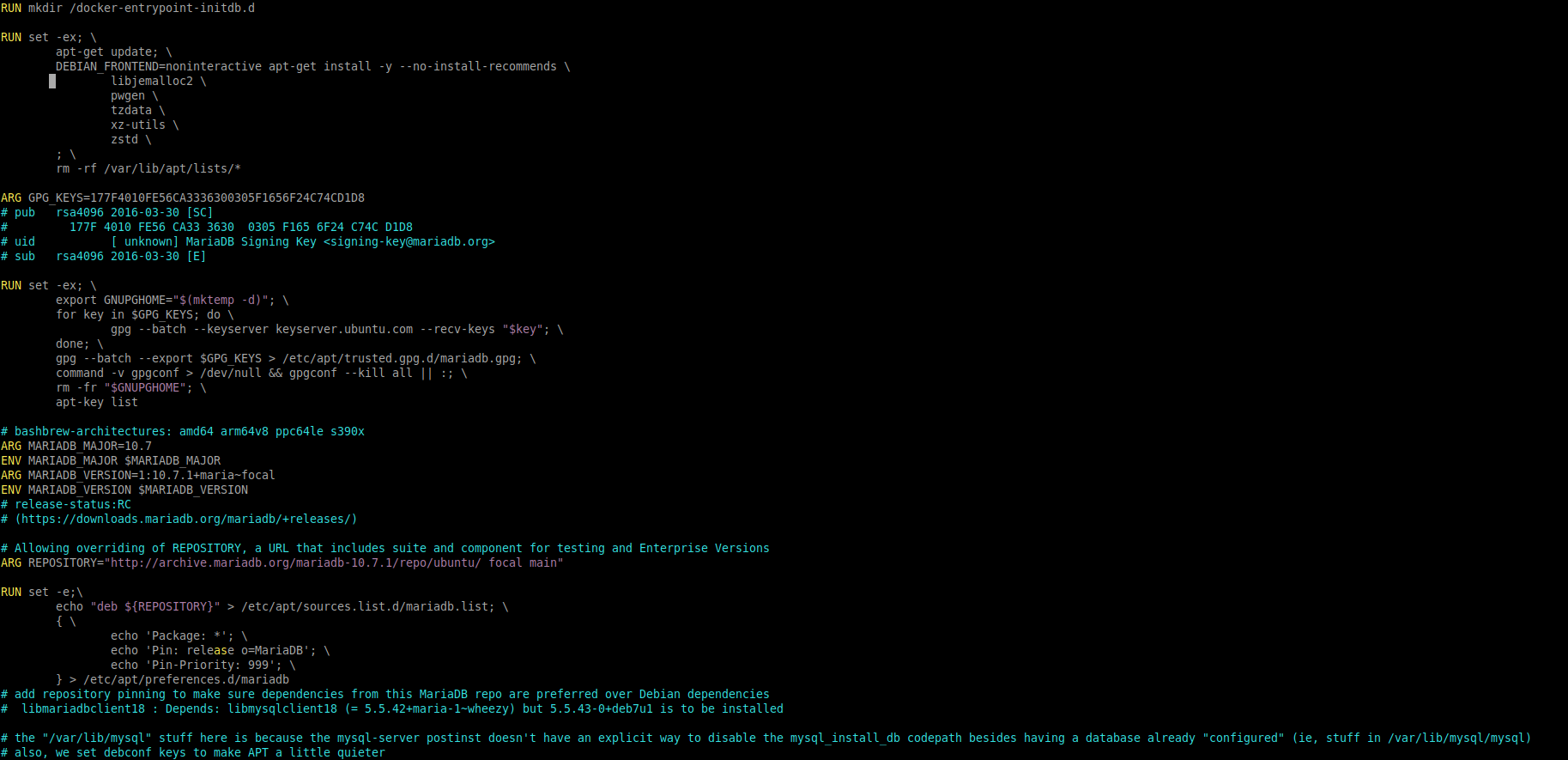


FROM: Ici on part de l’image d’un Ubuntu.

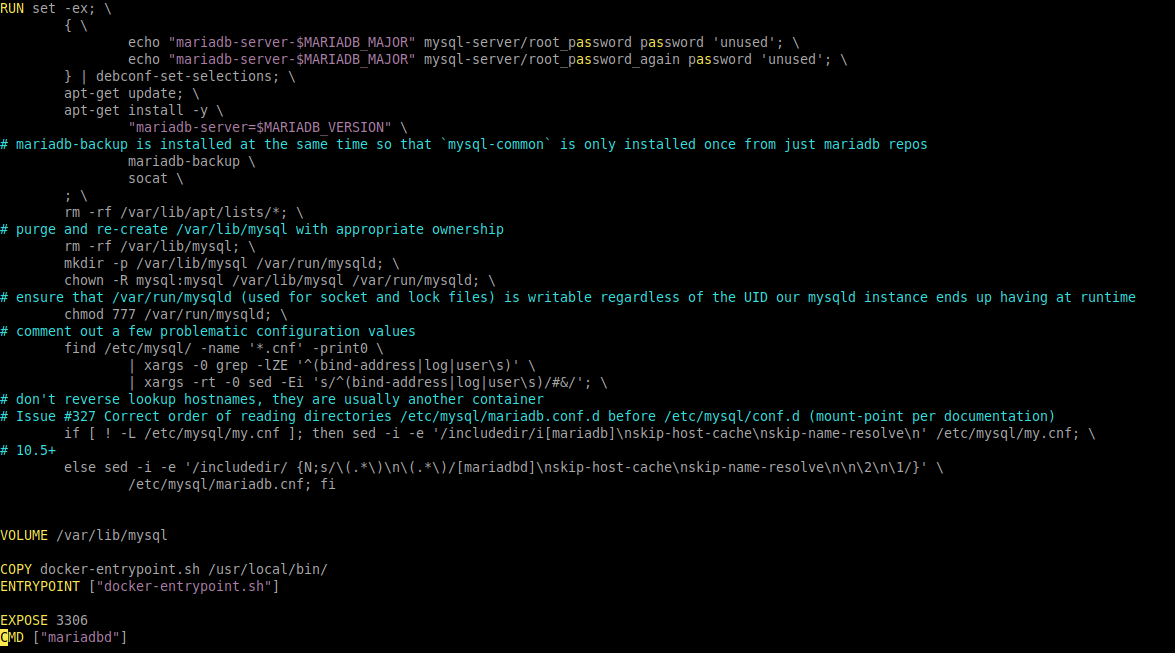
RUN : On créer le groupe mysql et l’utilisateur mysql qui viendra dans ce groupe.

RUN : ici on update et on réalise l’installation de gpg et de dirmngr. (ce qui permet de générer des clefs de chifffrement et de transmettre des information chiffrées) Le rm -rf nous permet de supprimer tout ce qu’il se trouve dans le chemin indiqué.

RUN : Installation et utilisation du wget afin d’aller chercher le packet nécessaire et de les installer.

RUN : Création du répertoire docker-entrypoint-initdb.d

RUN : Installation des ressources nécessaires au fonctionnement de la base de données

RUN : Ajout de la source pour installer les packet mariadb

RUN : définition du mot de passe root de la base de données. Création des répertoires nécessaire au bon fonctionnement de la base de données, mise en place des bonnes permissions.

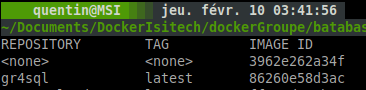
VOLUME : Chemin du volume.

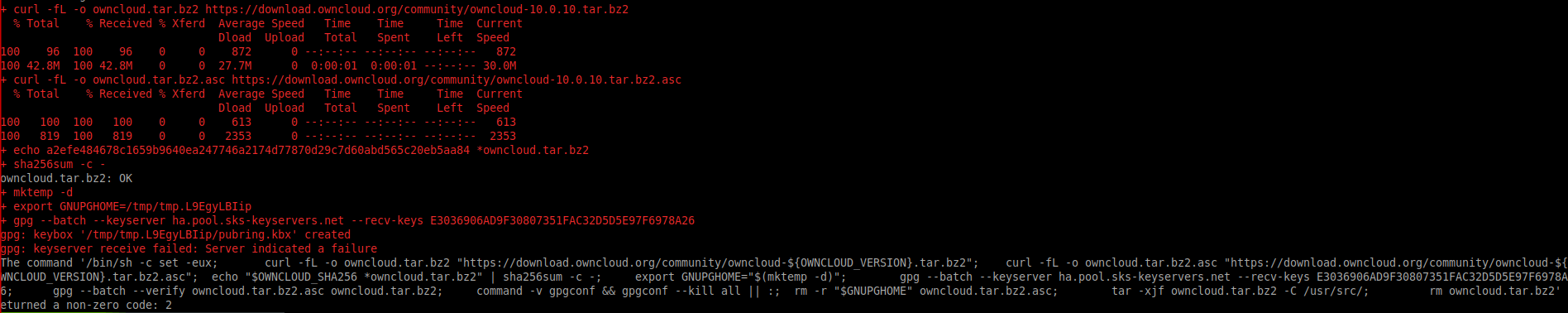
COPY : copie du script à exécuter. (Ce scripte définit le comportement du container une foi exécuté)

ENTRYPOINT : Lancement du script.

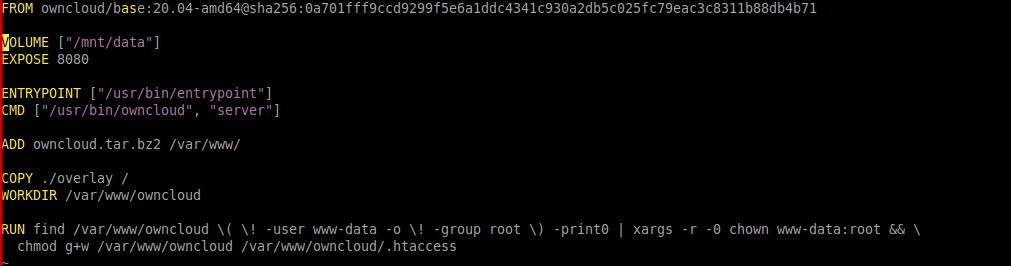
EXPOSE : ouverture du port commun d’une base de données SQL (3306)

CMD : démarre le service une foi l’installation terminé.

Nous avons créé notre image correspondante.

Ensuite lors de la création de l’image de owncloud, notre dockerfiles a échoué. On le constate sur la capture suivante.

Nous n’avons donc pas eu d’autres choix que d’utiliser le dockerfiles suivant.



FROM : Va chercher l’image de owncloud déjà prête sur le docker hub.

VOLUME : Chemin du volume.

EXPOSE : ouverture du port 8080 pour l’accès au service.

ENTRYPOINT : Récupération des paramètres définissant le comportement du container.

CMD : Lancement de owncloud

ADD : Copier le fichier indiqué dans le répertoire /var/www/

COPY : Copie l’overlay dans à la racine.

WORKDIR : indique le répertoire ou sont effectué les actions.

RUN : définition des droits pour le bon fonctionnement du service owncloud.

Création de l’image à partir du dockerfile. 

Pour conclure, devant la complexité de la création d’image pour mettre en place un owncloud. Nous avons dû utiliser des dockerfile disponibles sur le GitHub mais nous nous sommes efforcés de comprendre leur fonctionnement.