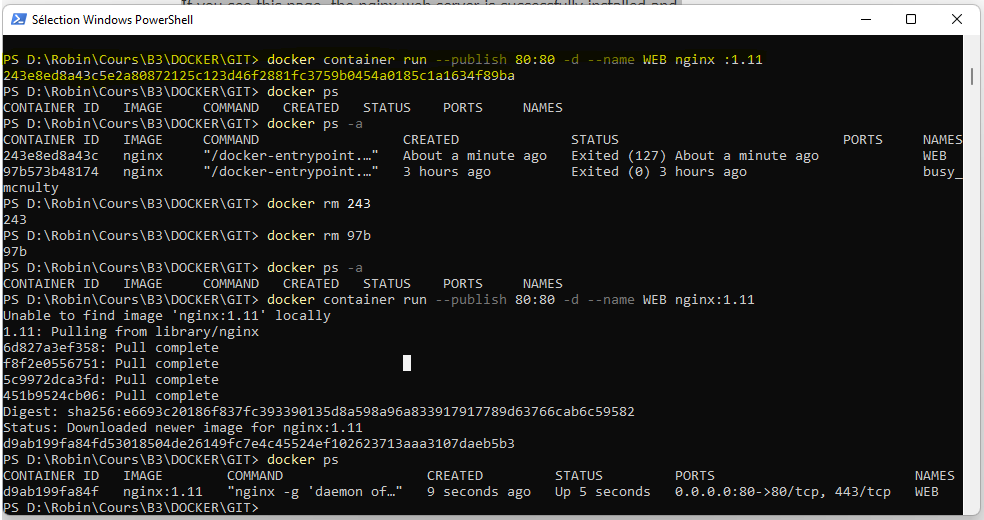
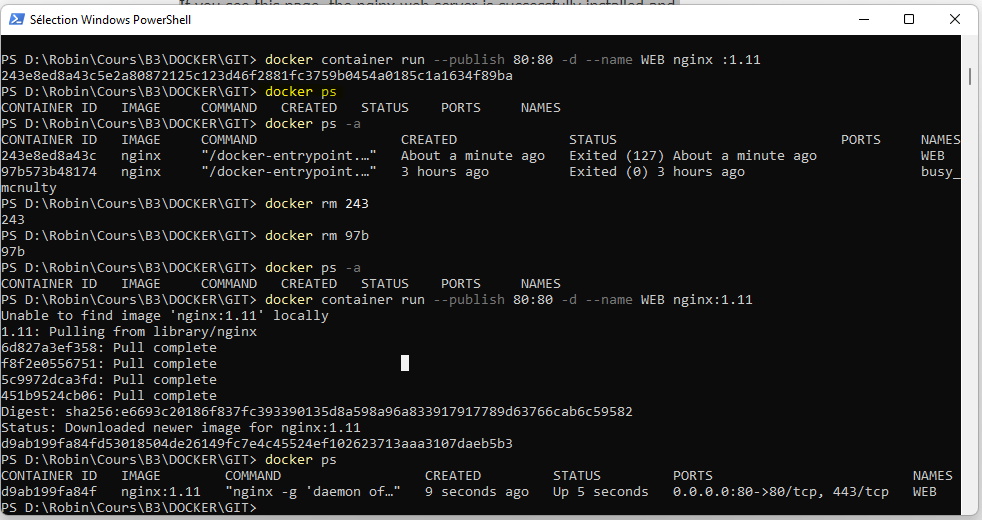
Compte rendu semaine passé sur le docker.

1. Commande découverte docker.

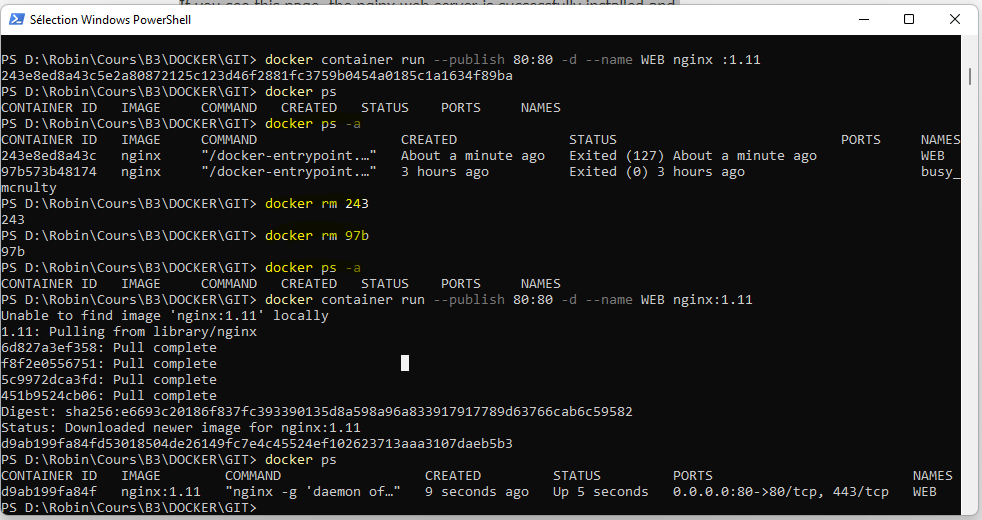
Création de mon container docker container run --publish 80:80 -d --name WEB nginx :1.11



Par la suite je souhaite visualiser mes containers existants. Avec la commande Docker ps

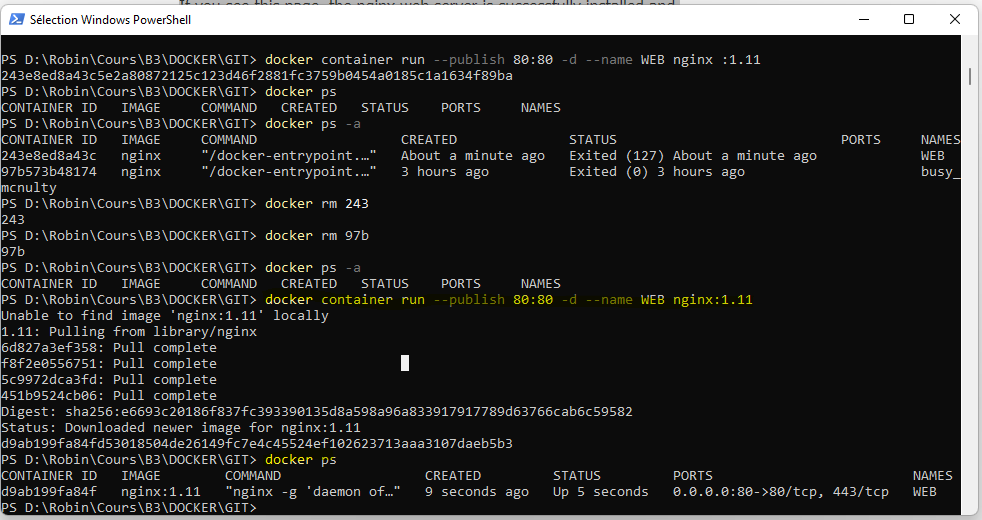


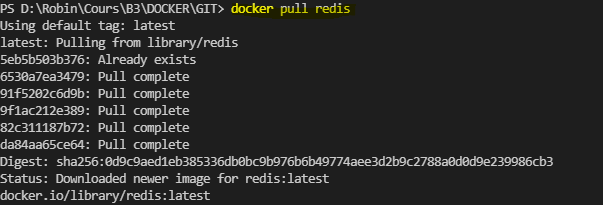
Je n’avais aucun résultat alors j’ai fait un docker ps -a celui-ci m’affiche bien deux containers un créé au préalable et celui créé à l’instant. Par la suite je les ai donc supprimés. Avec la commande docker rm et les 3 premier caractère du CONTAINER ID.

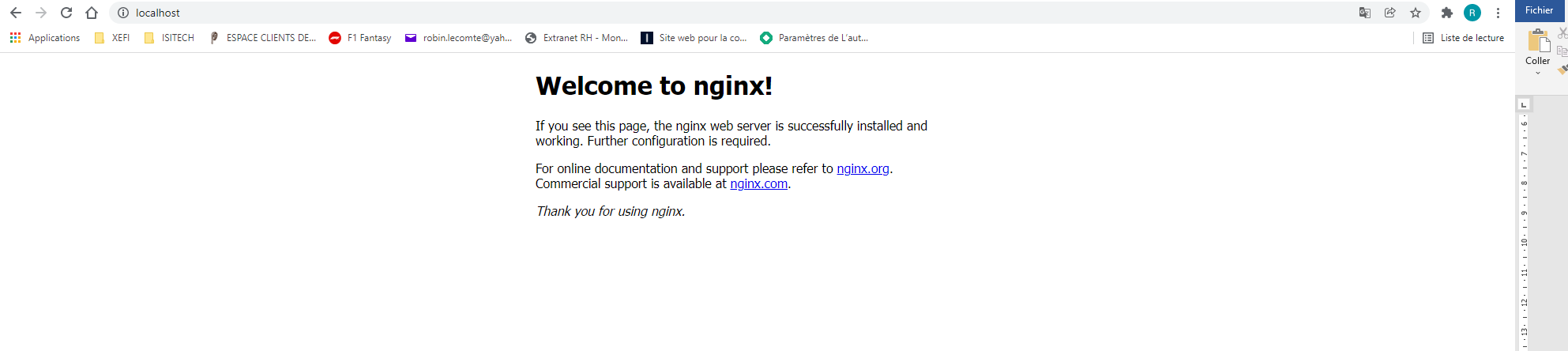


Pour finir, j’ai recréé mon container mais cette foi j’ai modifié la fin de la commande. J’ai collé le nginx :1.11 et mon container c’est créé correctement.

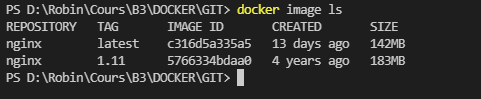
Docker container run --publish 80:80 -d --name WEB nginx :1.11





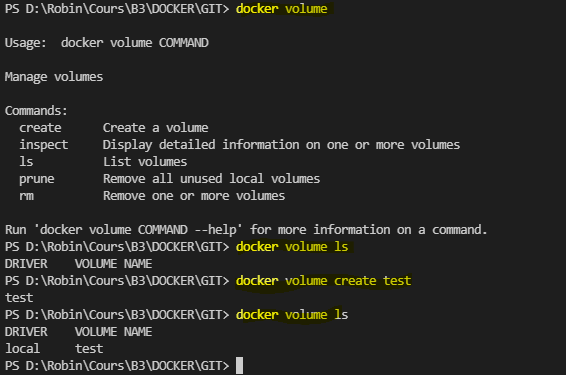
Par la suite, j’ai bien accès via l’adresse web localhost à la pas web suivante.

Par la suite j’ai réalisé la commande docker image ls afin de visualiser les images que j’ai

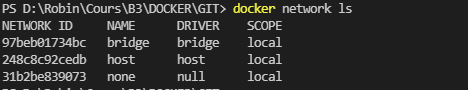


Par la suite, j’ai réalisé la création d’un volume que j’ai nommé test.

Au début on voit que j’ai aucun volume et par la suite j’en créé un.

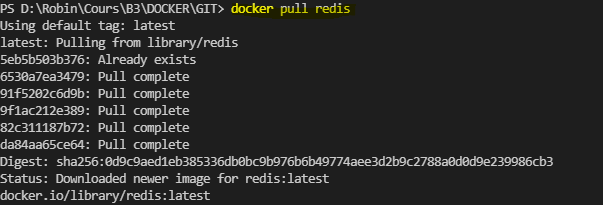


J’ai ensuite réalisé un Docker network afin de visualiser mes différentes cartes réseaux. La host correspond à ma physique sur ma machine, la bridge est une carte réseau virtuel dédié à mon docker, celle-ci me permet d’avoir accès à internet sur mon docker.

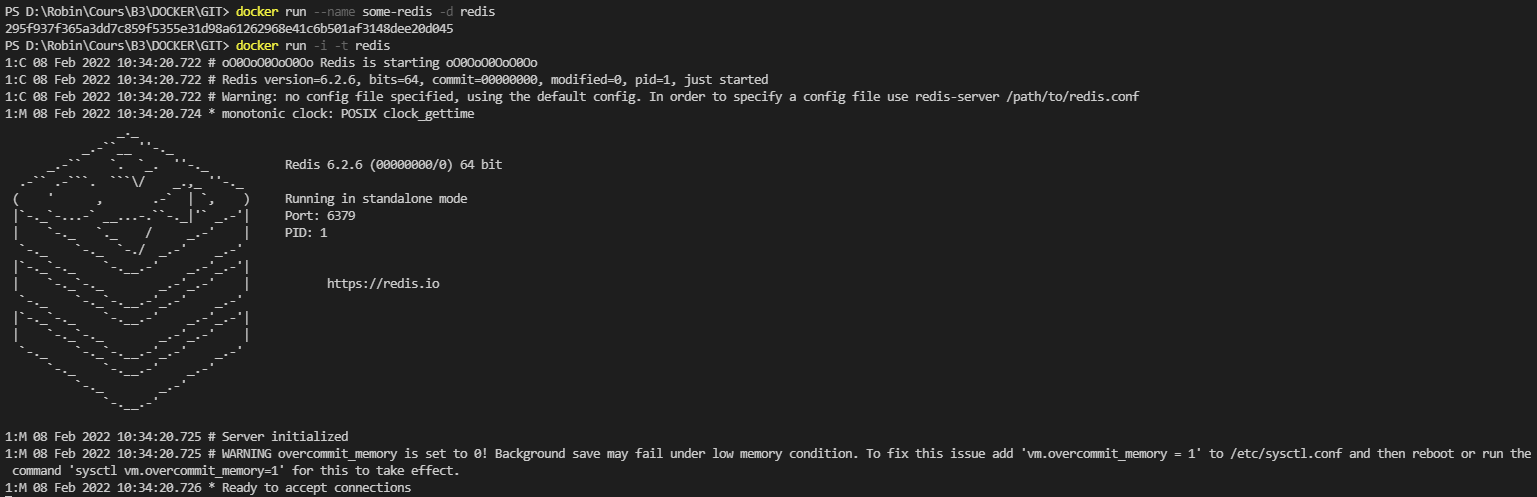


1. Importation et exportation d’une image stocké sur le docker hub.

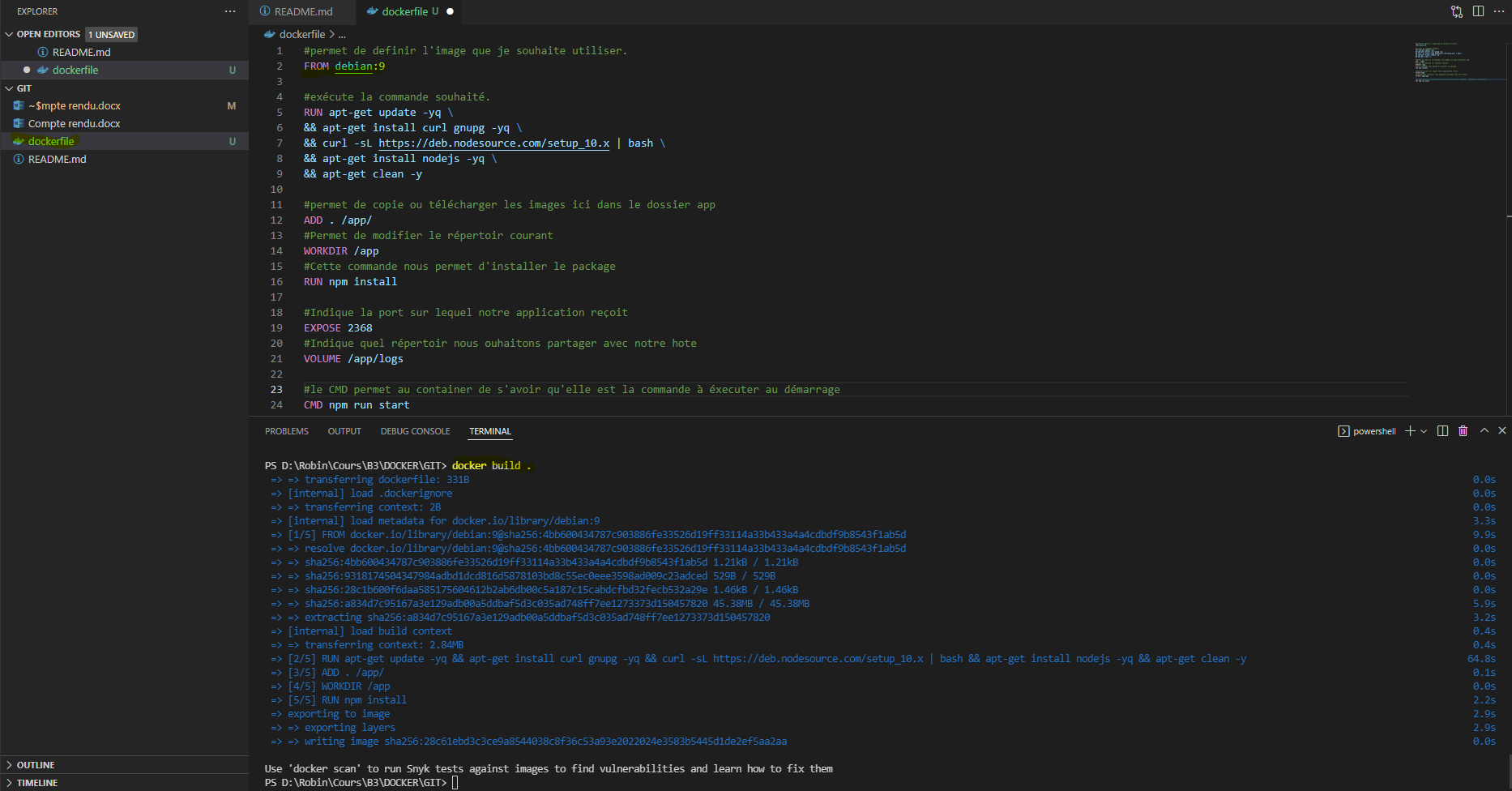
Ensuite, j’ai réalisé la commande Docker pull redis afin de télécharger l’image redis.



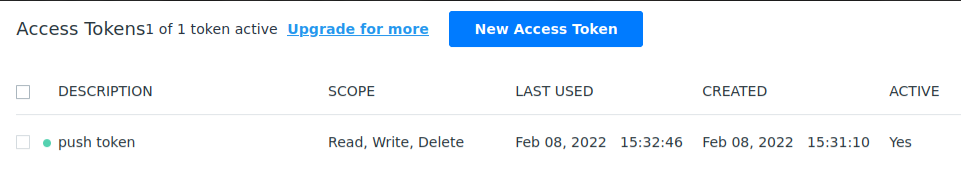
Ensuite, j’ai démarré mon instance redis puis je suis rentré dedans afin de pouvoir la manager.

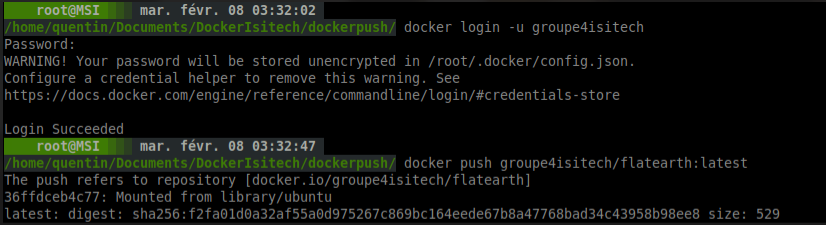


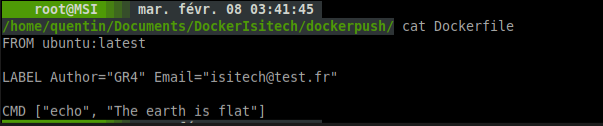
Par la suite j’ai créé un dockerfile Dans celui-ci j’ai défini Les différents argument que je souhaitais lui apporter. Ici je souhaite mettre en place l’image d’une debien9. On retrouvera tous les arguments qui ont été commenté sur la capture d’écran. Une foi sauvegardé, j’ai effectué la commande docker build . dans le terminal.



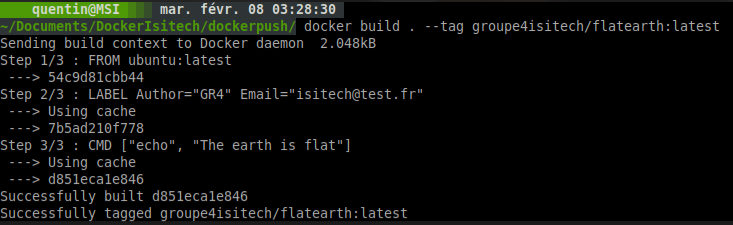
Création sur le docker hub de notre groupe ici groupe4isitech. Nous avons également généré notre token afin de pouvoir nous pouvoir nous authentifier en ligne de commande. Ici on



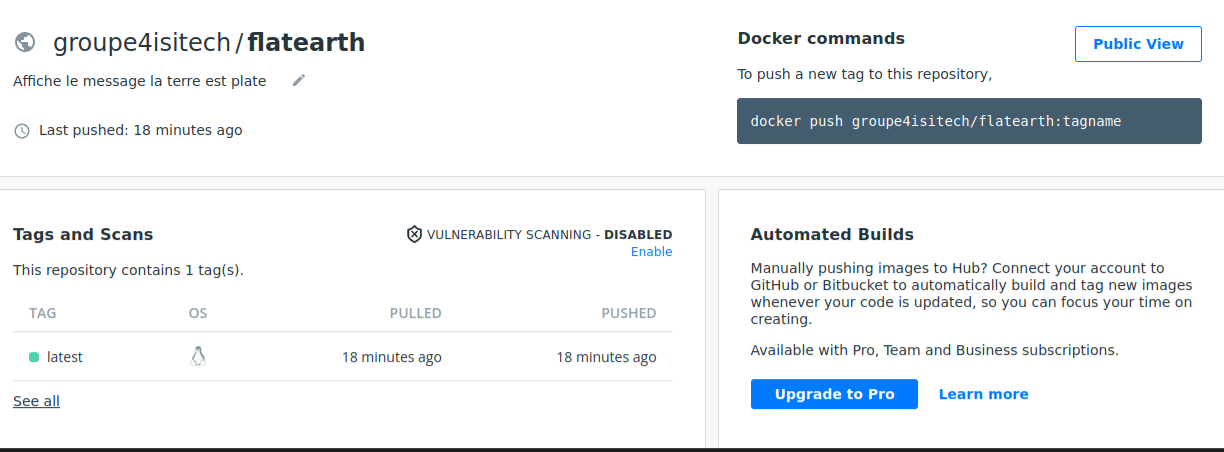
La commande docker login nous a permis de nous identifier sur notre compte. Docker push permet de faire remonter sur le docker hub les images que nous avons créées.

Ci-dessous voici le contenue du fichier Dockerfile.

Construction de l’image puis nommage de celle-ci afin de le push sur le docker hub.

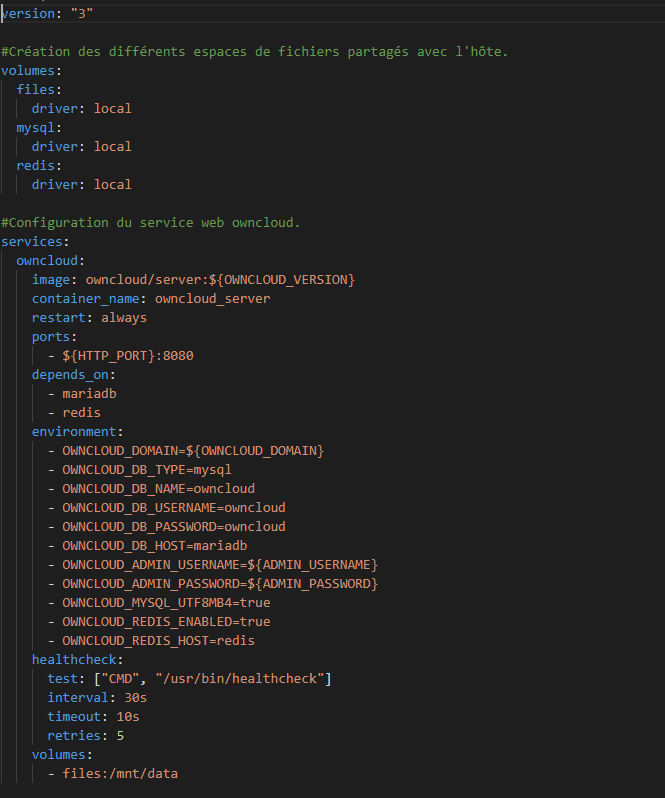


Sur la capture suivante, on voit bien remonter notre image sur le docker-hub.



1. Création d’un Docker-Compose

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



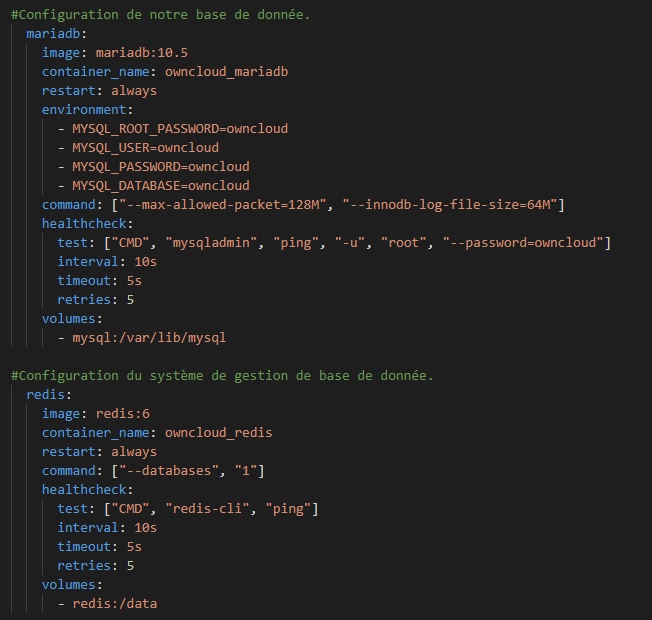


Image : récupère l’image du container que l’on souhaite utiliser. (Owncloud, mariadb, redis)

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.

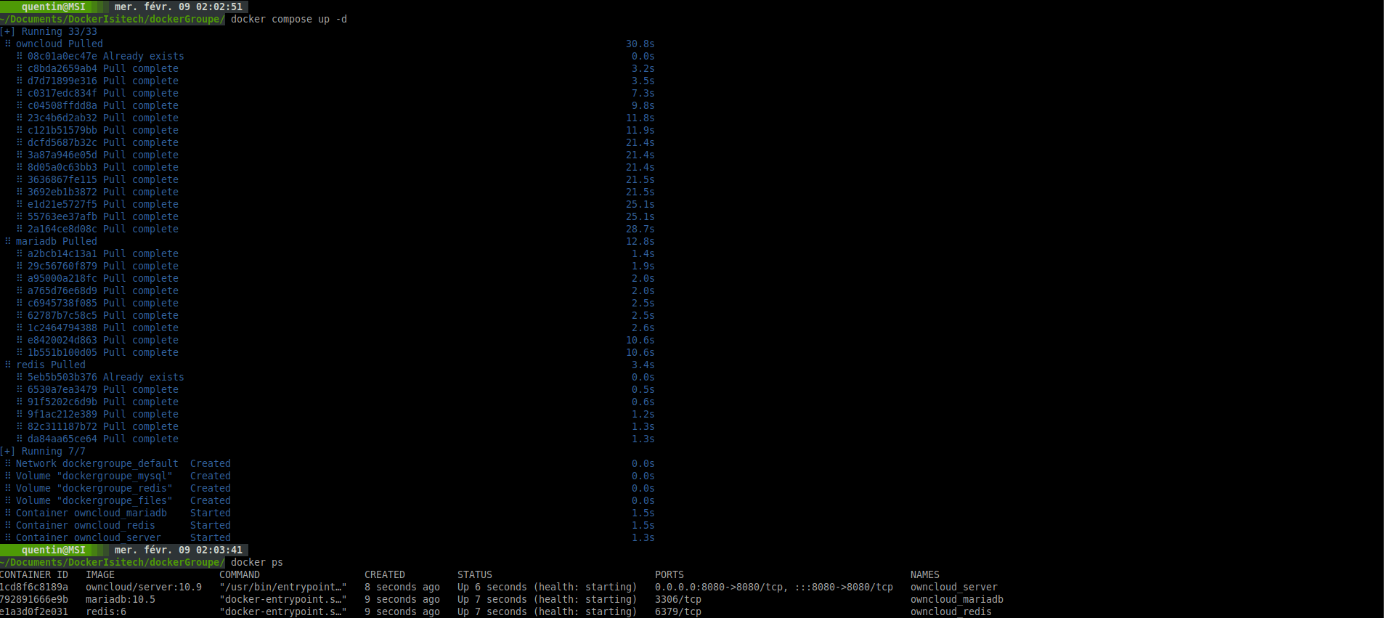
Environment : définition des variables d’environnement.

Healthcheck : Check de l’état du container.

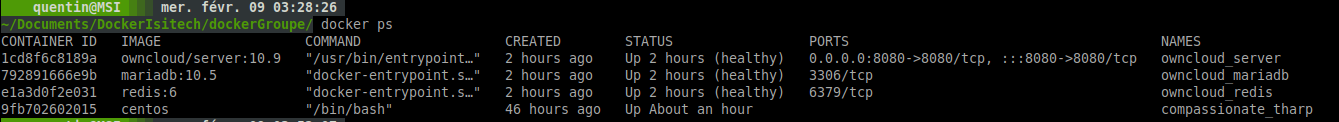
Volumes : Définition de l’emplacement du volume de fichier partagé.

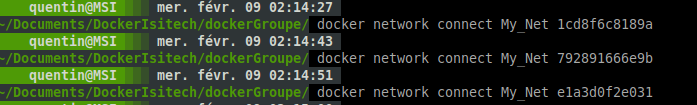
Command : Permet de rajouter des arguments lors de la création du container

Là nous retrouvons la commande docker compose up -d.

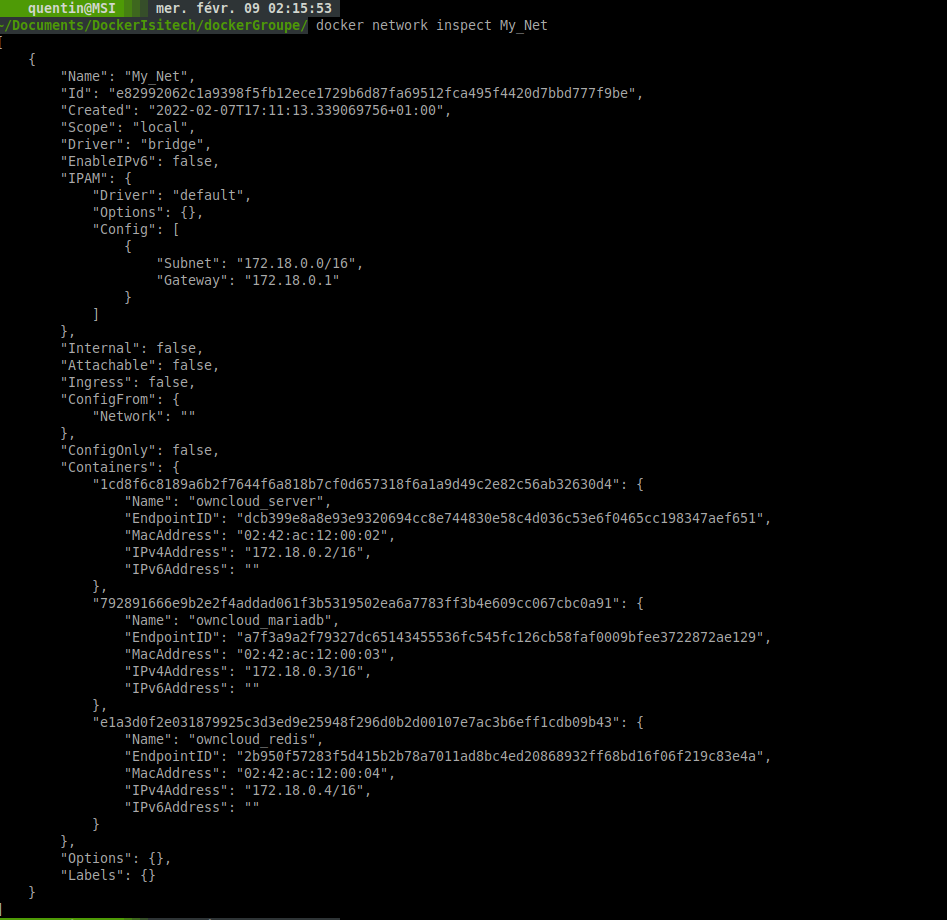
Cette commande un foi lancé, nous exécute notre fichier .yml et donc, nous créé nos différents container avec les paramètres précédemment établit.

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

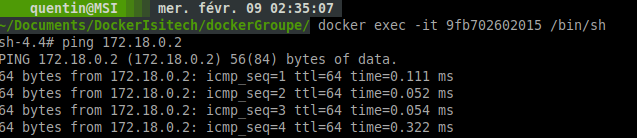


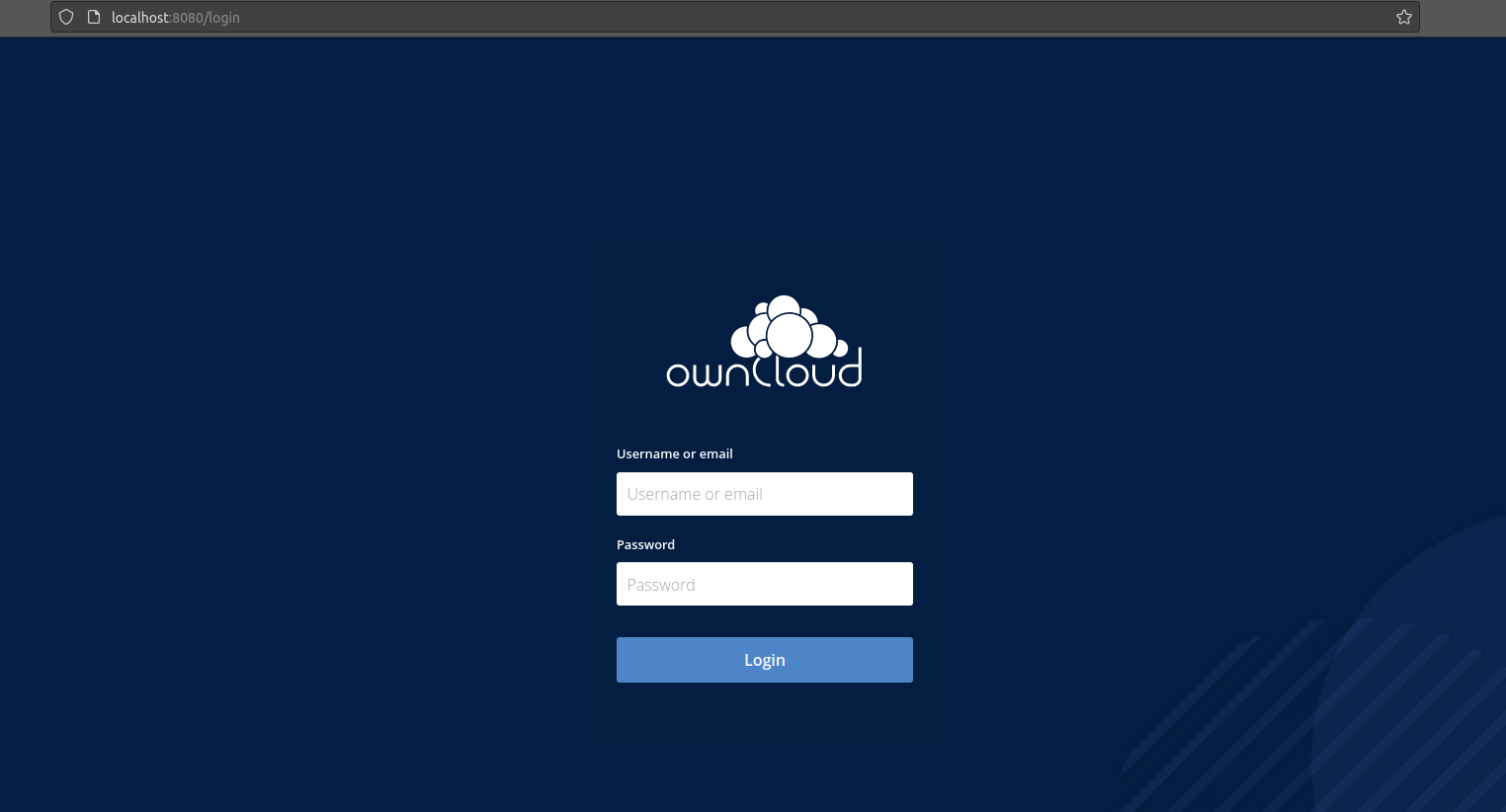
Nous avons connecté nos containers à un réseau virtuel commun précédemment créé via la commande : docker network create My\_Net

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.

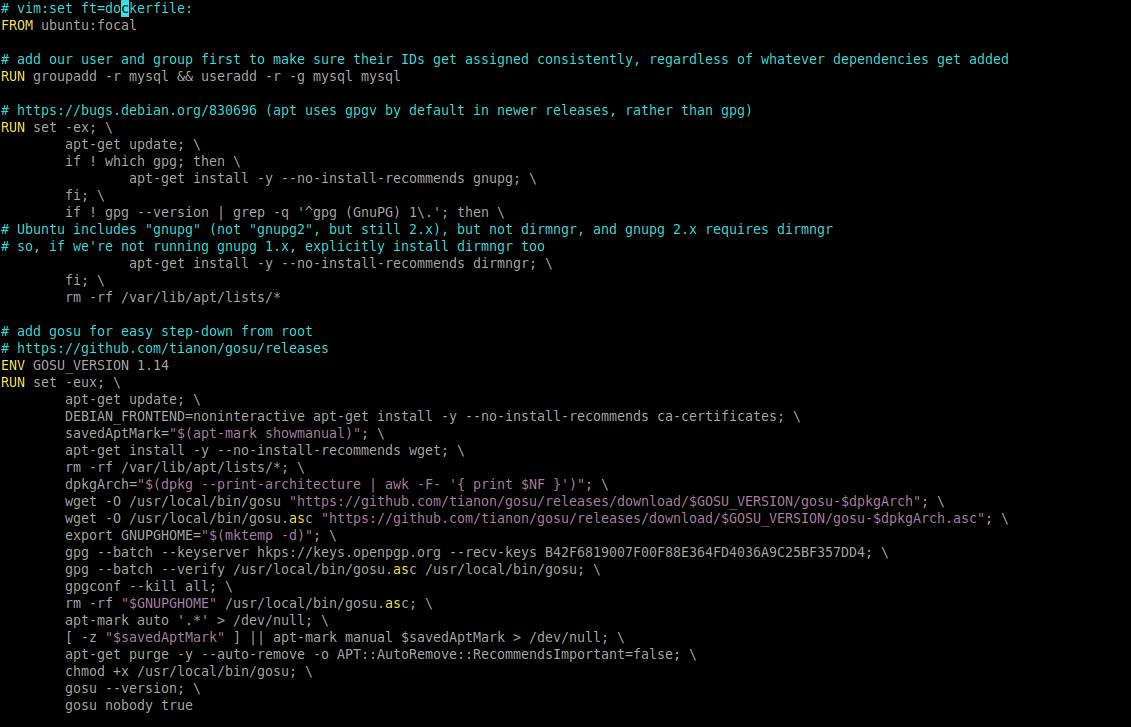


1. Projet groupe 4 owncloud

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 :

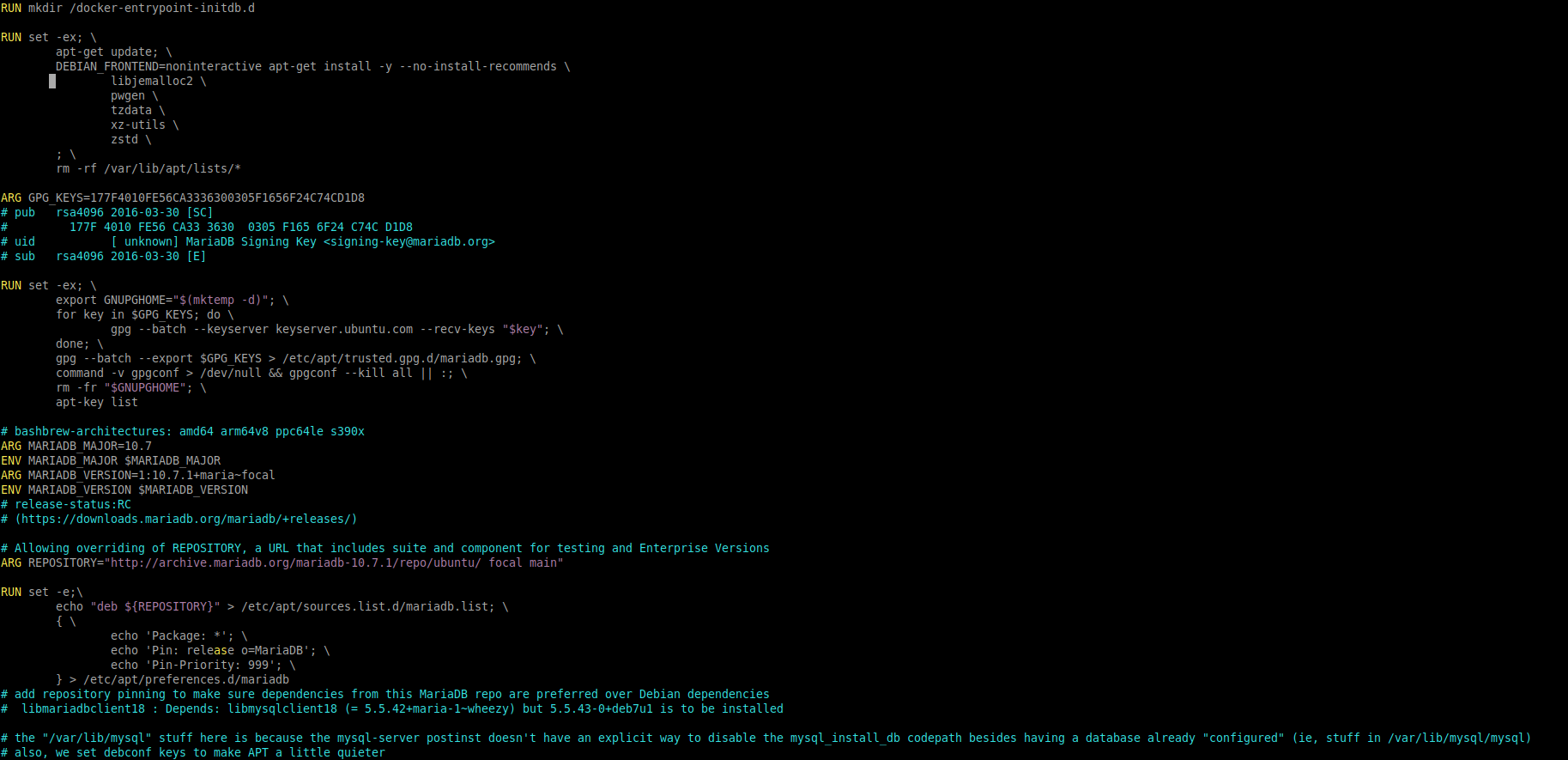


FROM: Ici on part de l’image d’un Ubuntu.

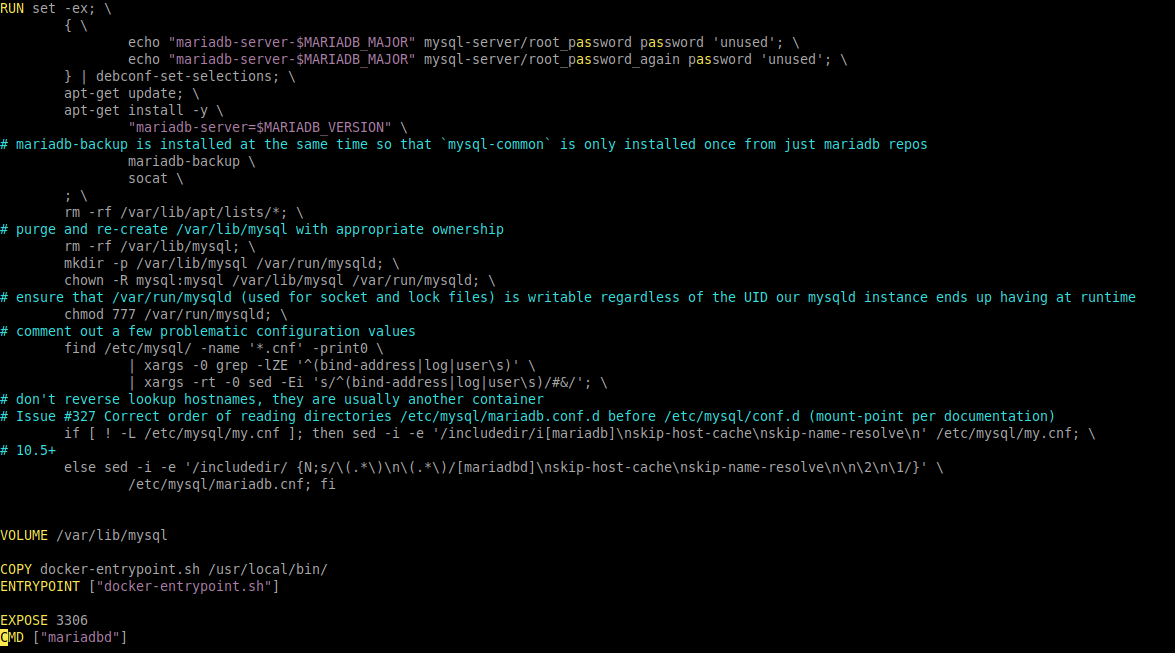
RUN : On créer le groupe mysql et l’utilisateur mysql qui viendra dans ce groupe.

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.

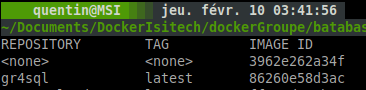
VOLUME : Chemin du volume.

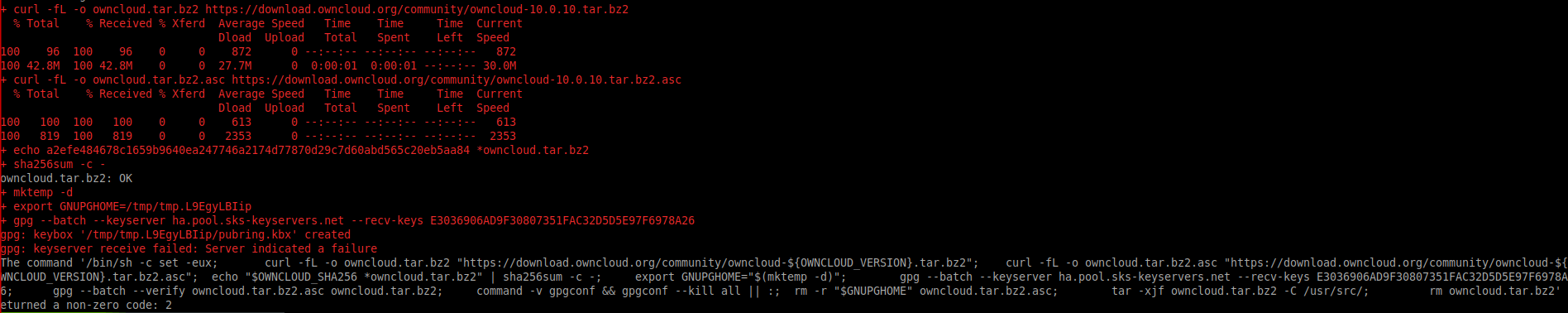
COPY : copie du script à exécuter. (Ce scripte définit le comportement du container une foi exécuté)

ENTRYPOINT : Lancement du script.

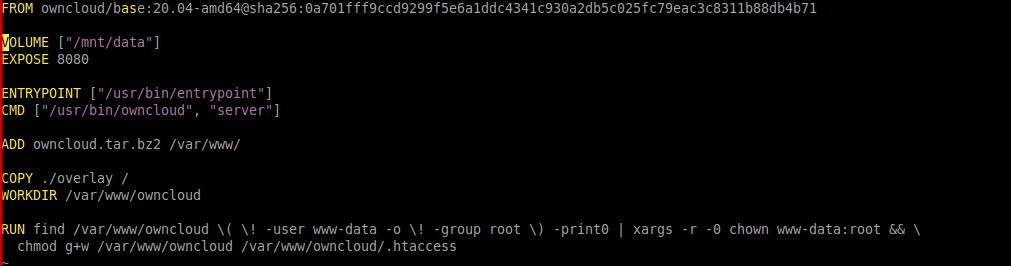
EXPOSE : ouverture du port commun d’une base de données SQL (3306)

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 disponible sur le GitHub mais nous nous sommes efforcés de comprendre leur fonctionnement.