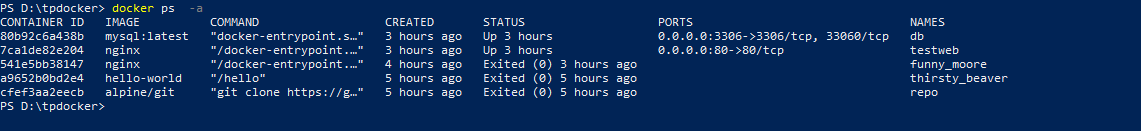
**Commande docker**

**Les différentes commandes utilisé le 7/02/2022** :

* Docker ps – a : permet de voir les images et les différents détails (id, status, ports), le -a nous permet de voir les images cachées

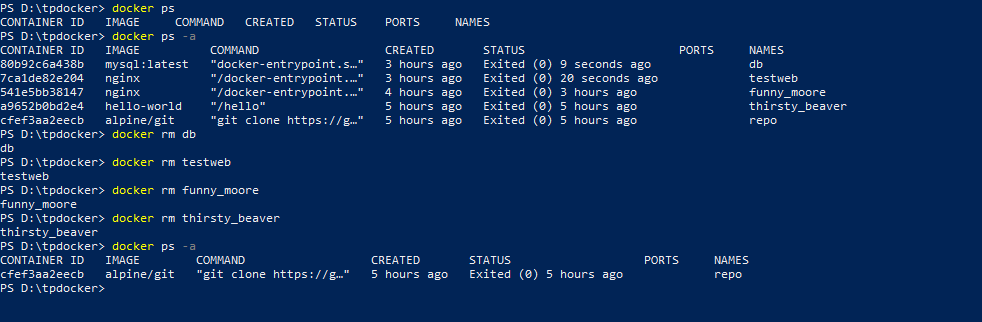
( docker images peut faire la même choses sans détails avancés, mais nous affiche la taille de l’image)



* Les différentes commandes ci-dessous vont servir à

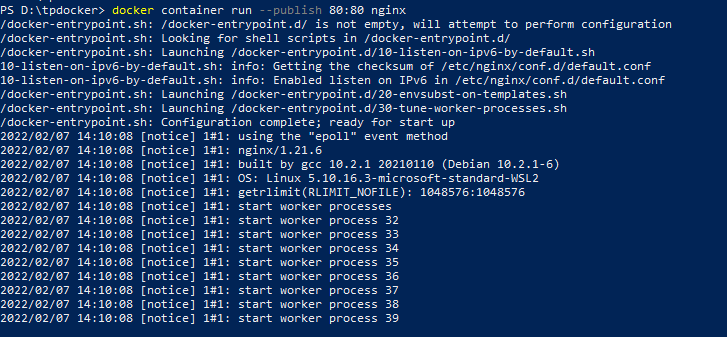
Effacé les images : rm docker ‘id’ ou ‘name’

Les lister : docker ps / docker ps -a



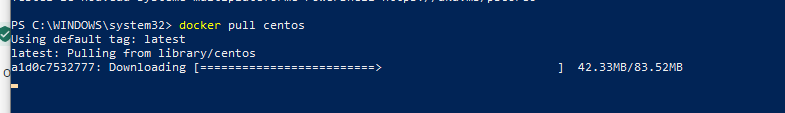
* Avec la commandes « container run –publish 80:80 « name »

On ouvre donc notre container avec le port 80 avec le « 80 :80 »

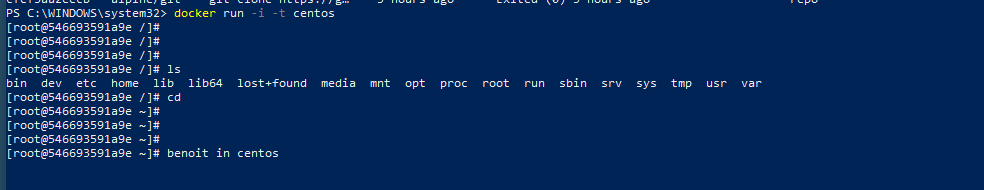


* Malheureusement le commande n’était pas complète, mais notre container s’est quand même lancé, mais cette dernière a bloqué notre cmd il aurait fallu rajouter « --detach » a note commande.
* Docker pull « name os » exemple : centos Ubuntu, glpi etc..

La commande télécharge donc notre images centos





Nous démarrons CentOS puis nous lançons CentOS 

**Les différentes commandes utilisé le 8/02/2022** :

Dockerfiles :

Le Dockerfiles sont des fichiers qui permettent de construire un docker pour nos besoins étape par étape, exemple de configuration :

FROM centos:lastest

LABEL author ="benoitsp" Email="benoit.soupou974@gmail.com"

ENV container docker

RUN yum -y update && yum clean all

RUN yum -y install httpd; yum clean all; systemctl enable httpd.service

RUN (cd /lib/systemd/system/sysinit.target.wants/; for i in \*; do [ $i == \

systemd-tmpfiles-setup.service ] || rm -f $i; done); \

rm -f /lib/systemd/system/multi-user.target.wants/\*;\

rm -f /etc/systemd/system/\*.wants/\*;\

rm -f /lib/systemd/system/local-fs.target.wants/\*; \

rm -f /lib/systemd/system/sockets.target.wants/\*udev\*; \

rm -f /lib/systemd/system/sockets.target.wants/\*initctl\*; \

rm -f /lib/systemd/system/basic.target.wants/\*;\

rm -f /lib/systemd/system/anaconda.target.wants/\*;

VOLUME [ "/sys/fs/cgroup" ]

EXPOSE 80

CMD ["/usr/sbin/init"]

(Pour le moment ne fonctionne pas)

Si les lignes de commande fonctionne il faut effectué un « docker build . »

Docker compose :

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services inter-connectés.

Un exemple de configuration de docker compose pour wordpress sous base de données mysql :

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: somewordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

volumes:

db\_data:

**(Projet) du 9/02/2022**

Mettre en ligne notre notre images docker sur le hub :

Afin de pouvoir mettre en ligne notre images il va donc falloir exécuter la commande suivante :

Docker tag ocr-docker-build:latest YOUR\_USERNAME/ocr-docker-build:latest

En l’adaptent à nos besoins, cette dernière qui va nous crée un lien entre image crée et celle que nous voulons envoyer sur le hub.

Une image contenant texte, moniteur, intérieur, télévision

Description générée automatiquementCi-dessous voici le contenue du fichier Docker file.

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

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

Création d’un Docker-Compose

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

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

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.

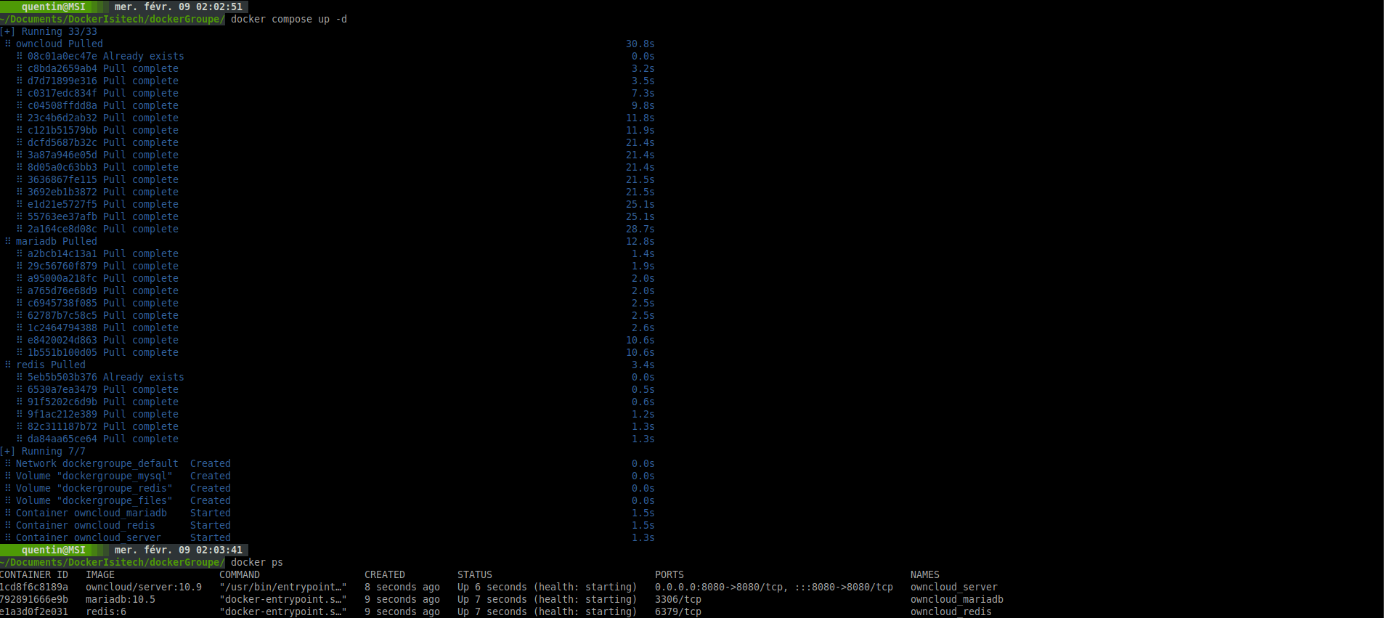
Environment : définition des variables d’environnement.

Healthcheck : Check de l’état du container.

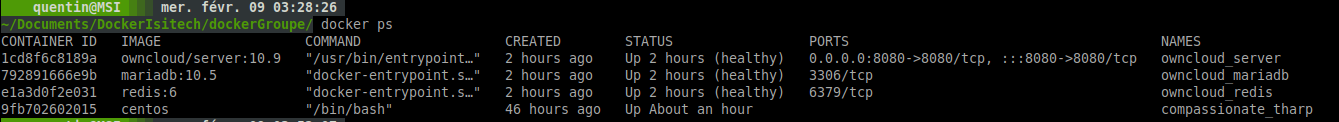
Volumes : Définition de l’emplacement du volume de fichier partagé.

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.



Une image contenant texte

Description générée automatiquementNous 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.

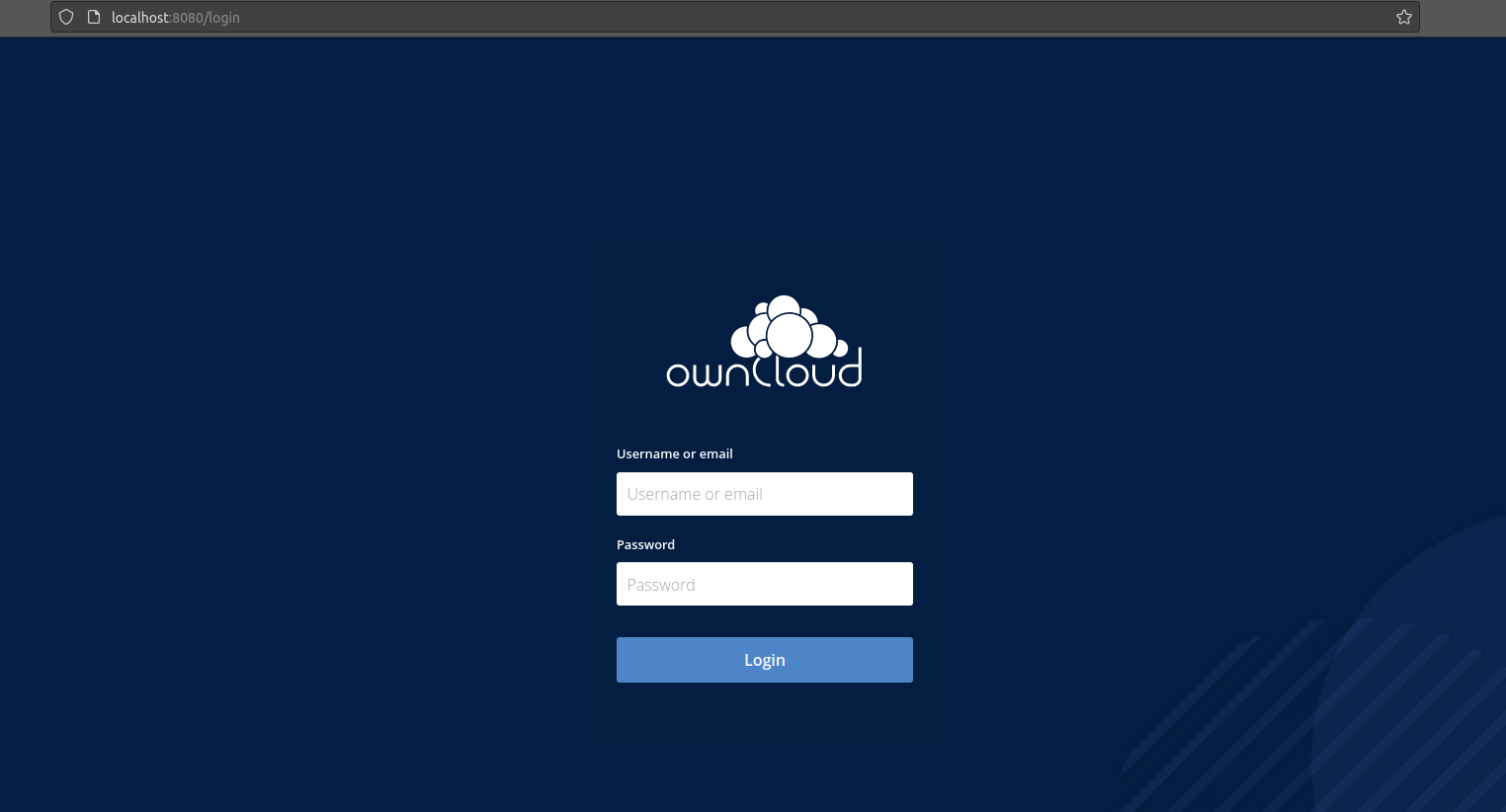
Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

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 :

Une image contenant texte

Description générée automatiquement

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.

Une image contenant texte

Description générée automatiquementRUN : Création du répertoire docker-entrypoint-initdb.d

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

Une image contenant texte

Description générée automatiquementRUN : 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.

Une image contenant texte

Description générée automatiquementNous avons créé notre image correspondante.

Une image contenant texte

Description générée automatiquementEnsuite 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.

Une image contenant texte

Description générée automatiquement

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.