DOCKER

[**Les Containers** 2](#_Toc21693395)

[**Getting a shell inside container, no need for SSH** 6](#_Toc21693396)

[**Docker for network , concepts for private and public comms in containers** 9](#_Toc21693397)

[Docker networks :CLI Management 9](#_Toc21693398)

[**Docker networks DNS and how containers find each others** 12](#_Toc21693399)

[**Docker Images** 15](#_Toc21693400)

[**Cree rune image** 20](#_Toc21693401)

[**Créer un image avec nodeJS** 23](#_Toc21693402)

[**Persistant data** 23](#_Toc21693403)

[**Solution 1 : Data volumes ou volume de données** 24](#_Toc21693404)

[**Solution 2 : Data bind mounting** 26](#_Toc21693405)

[**Docker Compose** 28](#_Toc21693406)

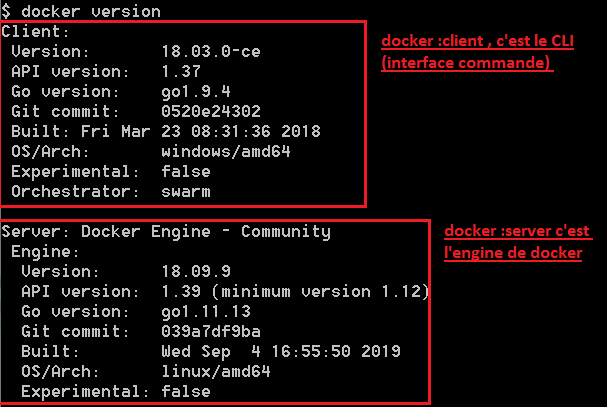
**Les Containers**

Containers ne sont pas des mini-VM

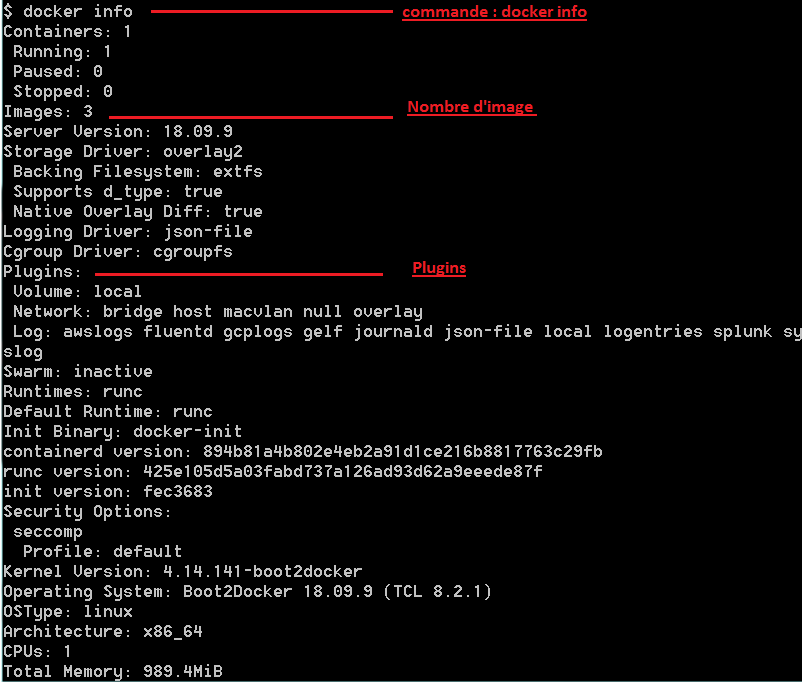
* C’est seulement des processus
* Limité leur accès aux ressources
* Ils sont fermés quand les procès sont arrêtés

Docker commandes :

**Docker version**



**Docker info**



Docker images

Pour voir toutes les images installer

Docker container - -help

Avoir l’aide sur les containers

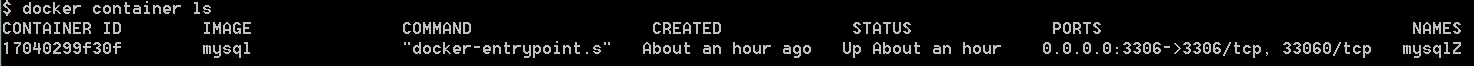
Docker container run - - publish 80:80 - -detach nginx

Récupère un conteneur de

Cette commande fait plusieurs taches :

* Cherche l’image en local si elle ne le trouve pas elle la télécharge depuis le repositoy Docker Hub
* Cree un nouveau container avec un id unique à partir de cette image ,
* Prépare le container pour le démarrer
* Fournir un ip virtuel au applis qui sont embarquées dans le container
* Ouvre le port 80 sur l’id de la machine host,c’est possible de changer le port de cible **9000**:80
* Fait un routage de trafic sur l’ip de container avec le port 80
* Démarrer le container avec les commandes spécifiées dans le fichier dockerfile qui se trouve dans l’image
* --detach : pour lancer le container en arrière plan en dore de la console

Docker container ls

Afficher la liste des containers qui sont démarrés

Docker container ls -a

Afficher toute la liste des containers

Docker container logs {nameContainer}

Récupérer le log de l’application installer dans le container

Docker container top {idContainer}

Afficher les commandes qui sont exécutés dans le container qui dans un état active

docker container ls ou docker ps dans les anciennes versions ps = process

Afficher les containers démarrés

Docker container rm id de container

C’est pour supprimer le container qui réside dans la mémoire, c’est possible de saisir que les 3 ou 4 chiffre de l’id container

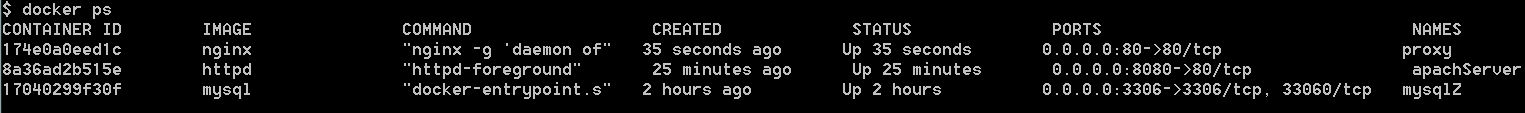
Exp : docker container rm 36f 553 d6c f00 , suprimer 4 containers

**Creation des contain Mysql , Apach , Ngnix**

docker container run --detach --publish 3306:3306 --name mysqlZ -e MYSQL\_RANDOM\_ROOT\_PASSWORD=yes mysql

docker container run - detach --name apachServer -p 8080:80 httpd

docker container run --detach --name proxy --publish 80:80 nginx

 docker container ls == docker ps dans les enciennes versions

Tester la connexion avec apach server

$ **curl 192.168.99.100:8080**

<html><body><h1>It works!</h1></body></html>

Diffèrent options pour l’affichage des containers

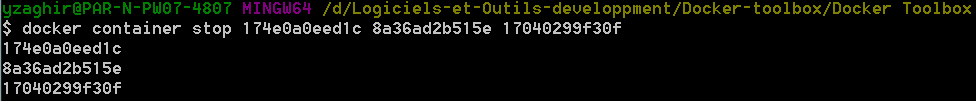
docker container ls -q quick

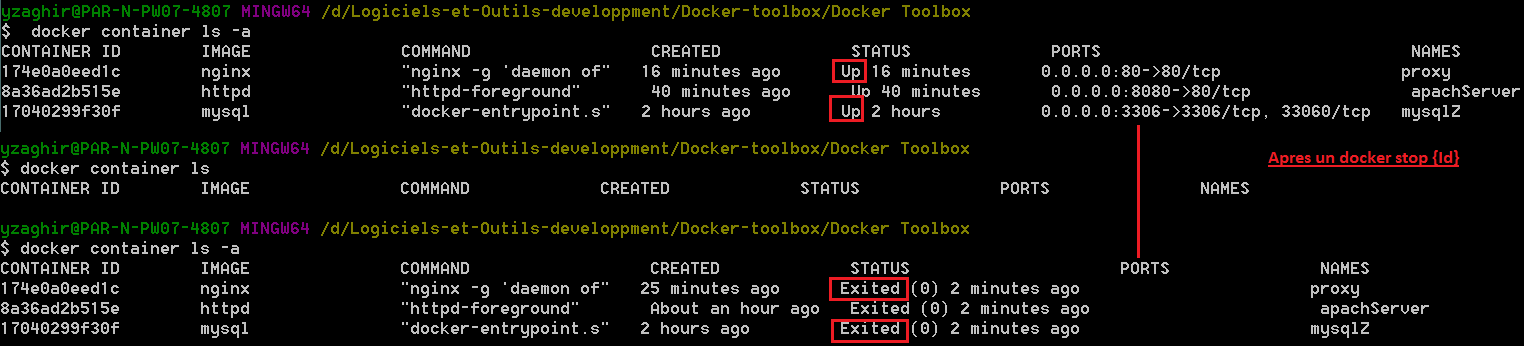
docker container ls afficher les container active

docker container ls -a all afficher tous les container



docker container strop {idContainers}

Pour arrêter les containers démarrés ici en arrêtes les 3 containers

Vérifier l’état des containers, avec un docker ps ou docker container ls et un docker container ls –a

Il faut savoir que les containers vie toujours dans le mémoire ,docker container ls –a montre bien que y a nos containers .Pour virer les containers il faut faire un

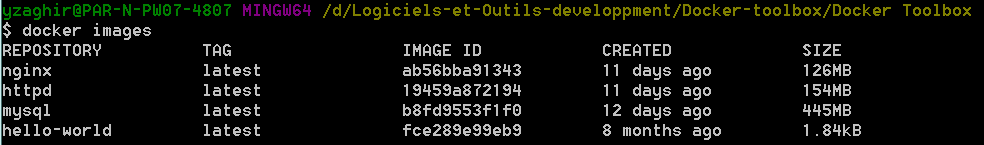
Docker container rm {idContainer}

On peut forcer l’arrêt d’un container même s’il est démarré avec docker container rm -f {id}

Les containers sont des instances créer à partir des images déjà télécharger, la même approche on la trouve dans les langages objets , on peut créer plusieurs instance d’un objet avec des paramètres à partir d’une classe , et c’est exactement ce que qu on fait avec docker , on crée plusieurs container à partir de la même image

Docker images

Afficher les images



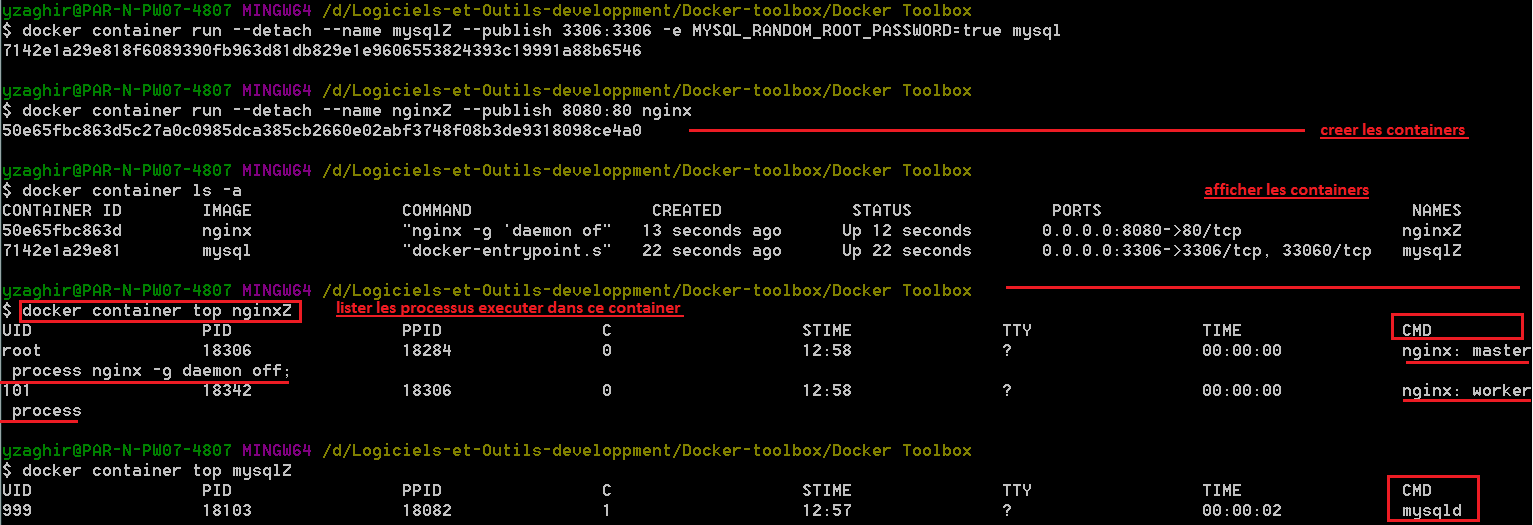
**Docker propose des outils de monitoring pour les containers**

Docker container top list des processus pour un seul container

Docker container inspect détails de la configuration pour un seul container

Docker container stats l’état de performance pour tous les containers

demo

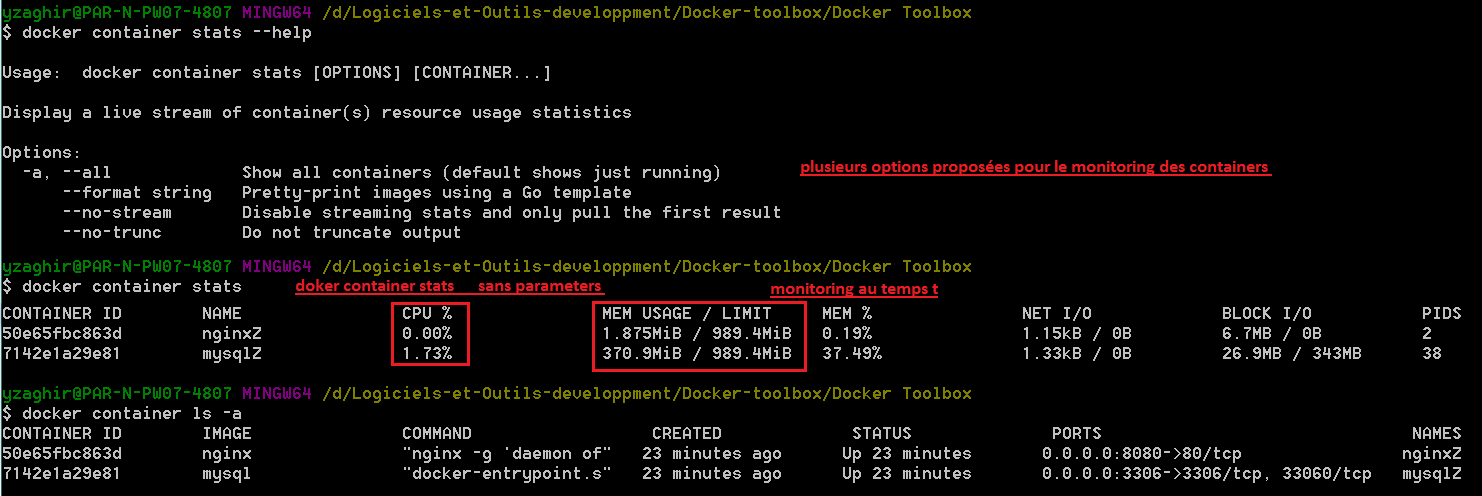


Pour afficher les metadata , la configuration, et comment le container est lancé , on utilise

Docker container inspect {id} ou name de container

Exp : docker container inspect mysqlZ ou docker container inspect 7142e1a29e81 , on récupère toute la configuration de container sous format Json

Pour voir l’etat des containers ,on utilise

Docker container stats

**Getting a shell inside container, no need for SSH**

C’est possible d’accéder au shell qui se trouve dans le container sans passé par un SSH

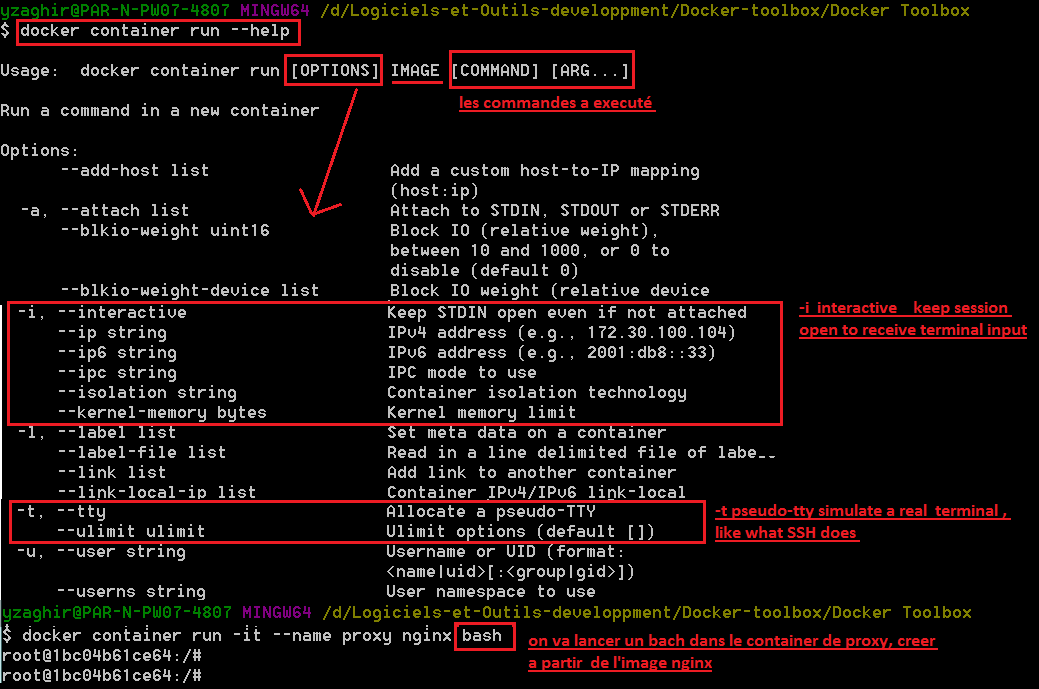
docker container run -it start new container Interactively

docker container exec -it run additional command in the existing container

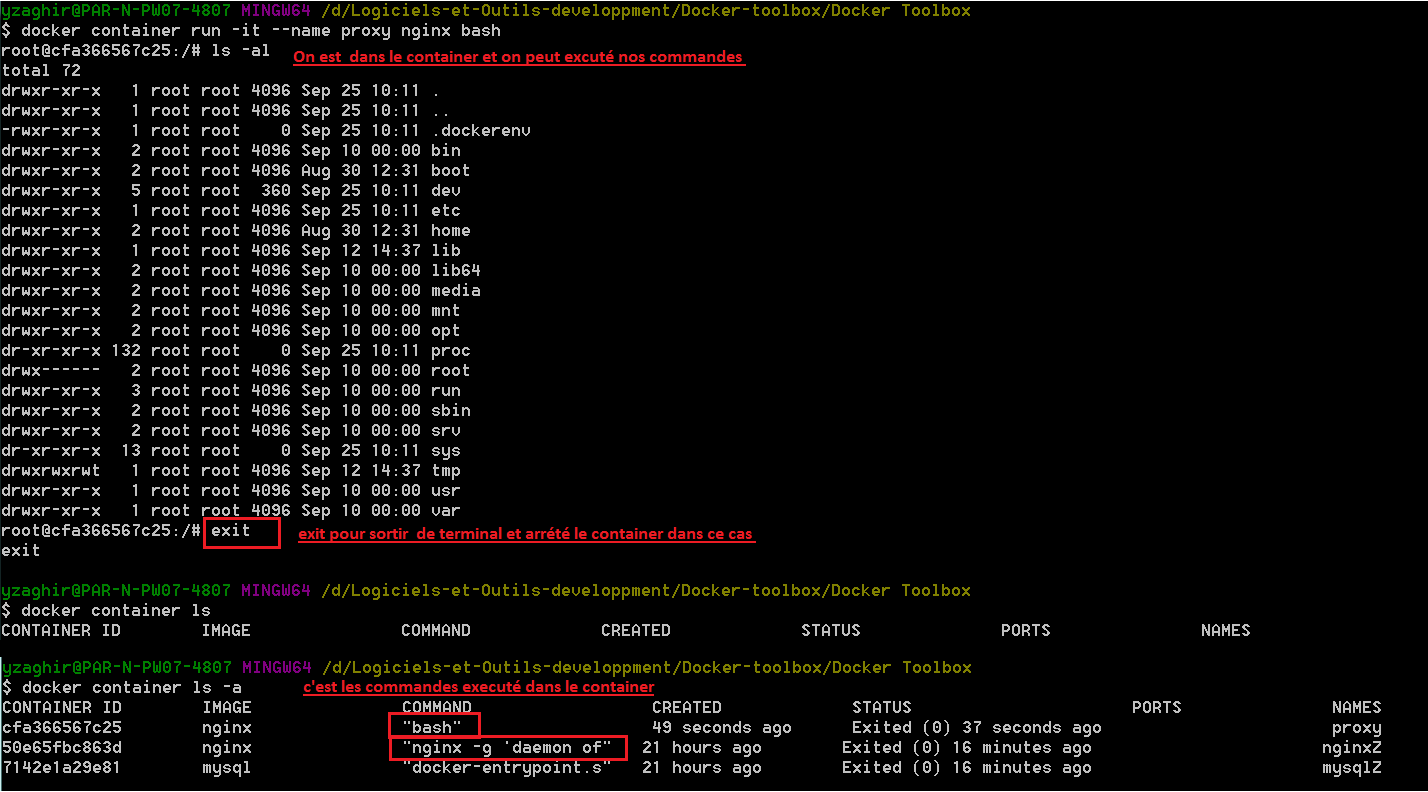
Voir la différence entre Linux distribution et un container

On peut accéder à l’intérieur d’un container sans passé par l’outil SSH

docker container run --help

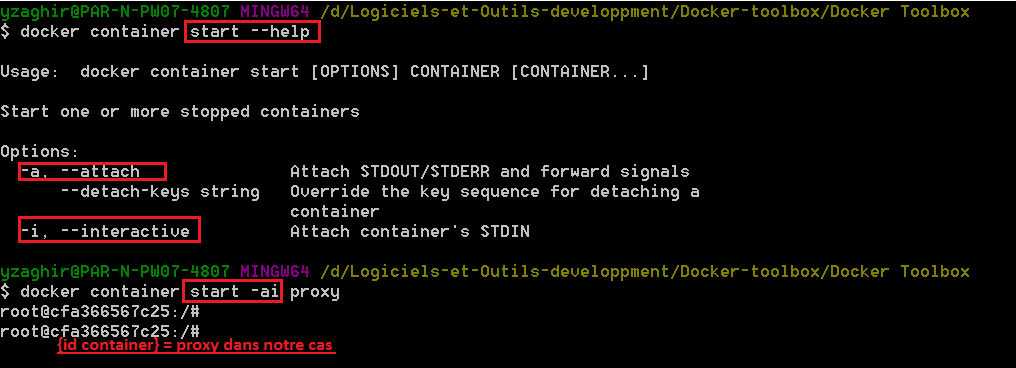


Bash shell : if run with it , it will give you a terminal inside the running container

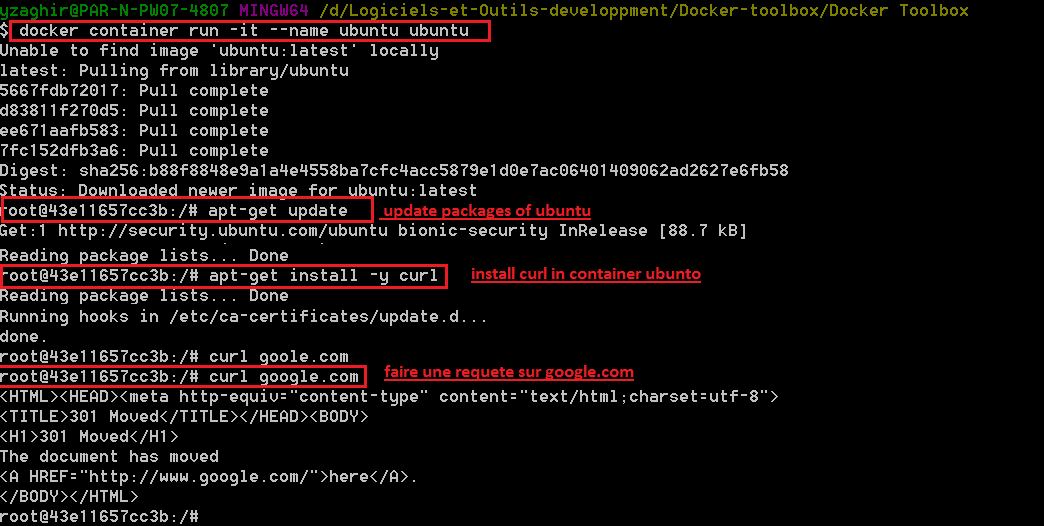


Pour démarrer le container on a plusieurs options , et puisque notre container se lance avec un bash on va utiliser le paramètre --ai pour le démarrer

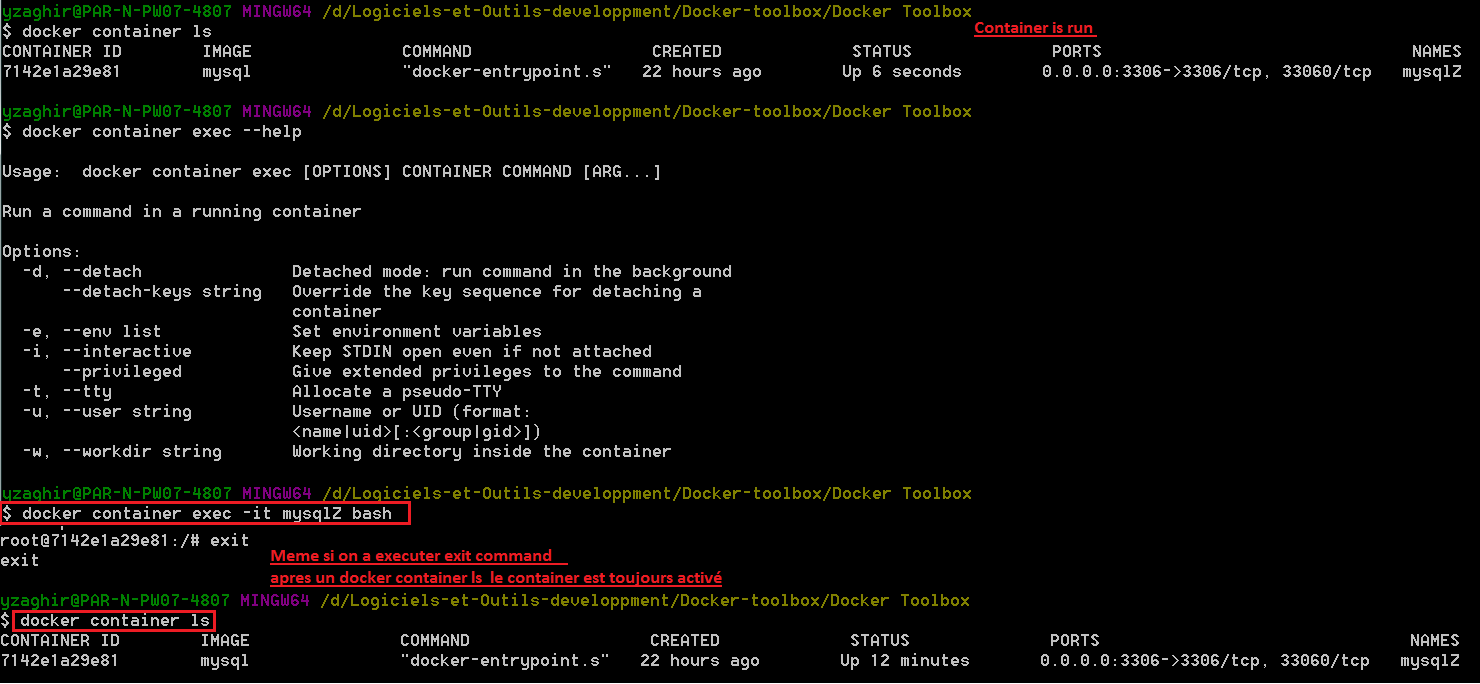
Docker container start - -ai {id ou name container }



Exemple



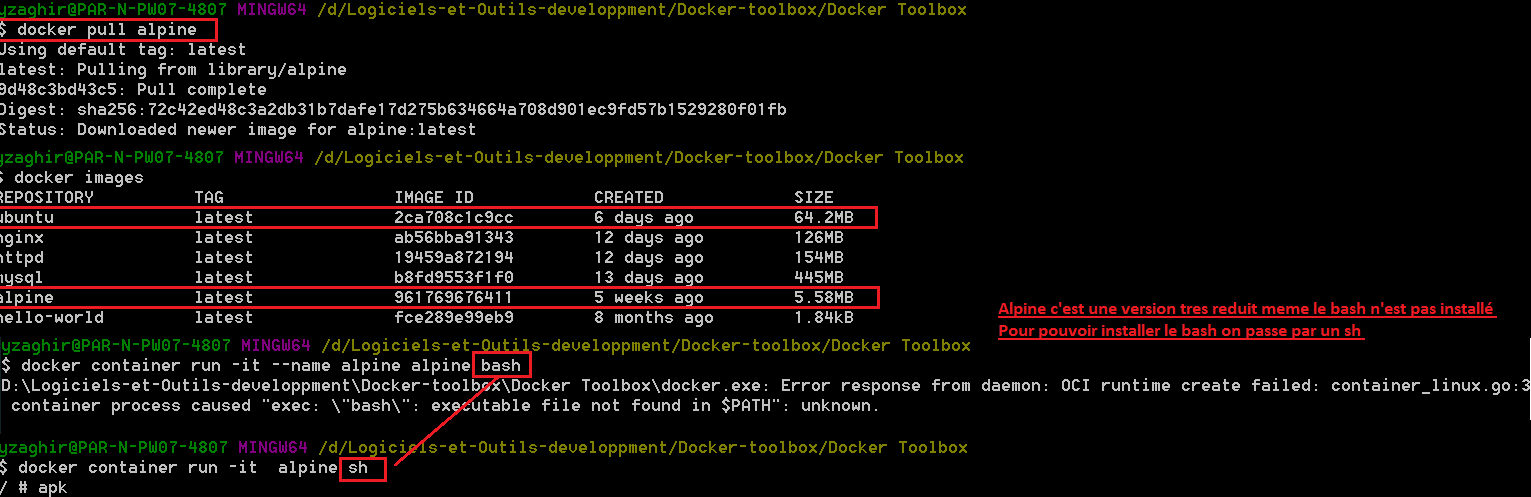
Docker container exec execute additionnelles taches dans un container démarré déjà

Docker container exec - -ai {id ou name container }

On peut retirer une image avec la commande

Docker pull {nom image}

Sur docker hub il y a des images de petites tailles que ubuntu , exemple Alpine



**Docker for network , concepts for private and public comms in containers**

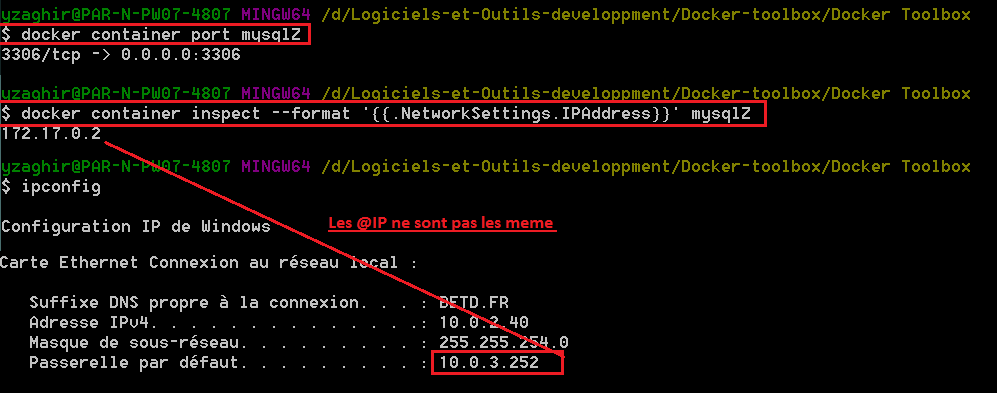
Docker propose des fonctions pour gérer la partie réseau

Récupère l’adresse ip du container, on format le résultat si non on récupère le gros fichier Json avec tout le paramétrages

docker container inspect --format '{{.NetworkSettings.IPAddress}}' nomContainer

Récupérer le port de container pas de host

Docker container port nomContainer



Dans une seule machine physique on peut lancer plusieurs containers , docker et le Os system utilisent le nat des port pour chaque container

Docker networks :CLI Management

Show networks docker network ls

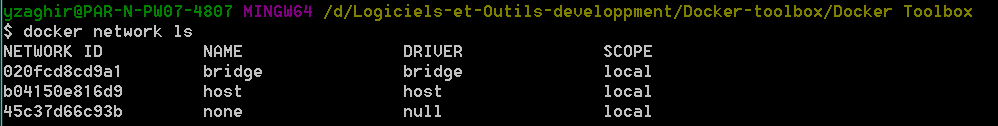
Inspect a network docker network inspect

Create a network docker network create –driver

Attach a network to container docker network connect

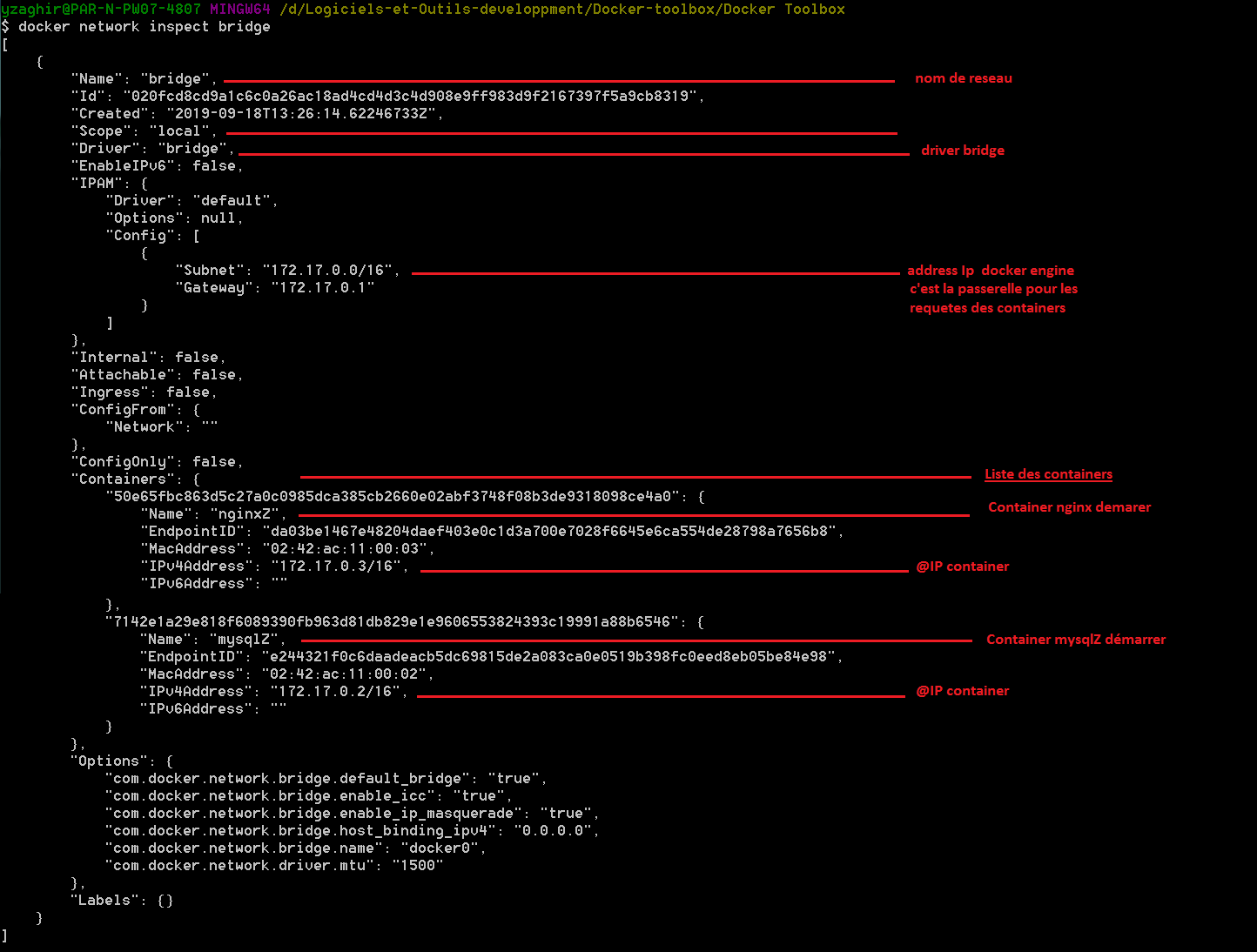
Detach a network from container docker network disconnect

Docker network ls

Afficher all networks 

**Le réseau bridge** est par défaut le réseau virtuel de docker, lequel il est nat derrière l’adresse ip de la machine physique

Au moment de la création d’un nouveaux container si on ne spécifie pas le réseau au container , se dernier sera connecter au réseau bridge

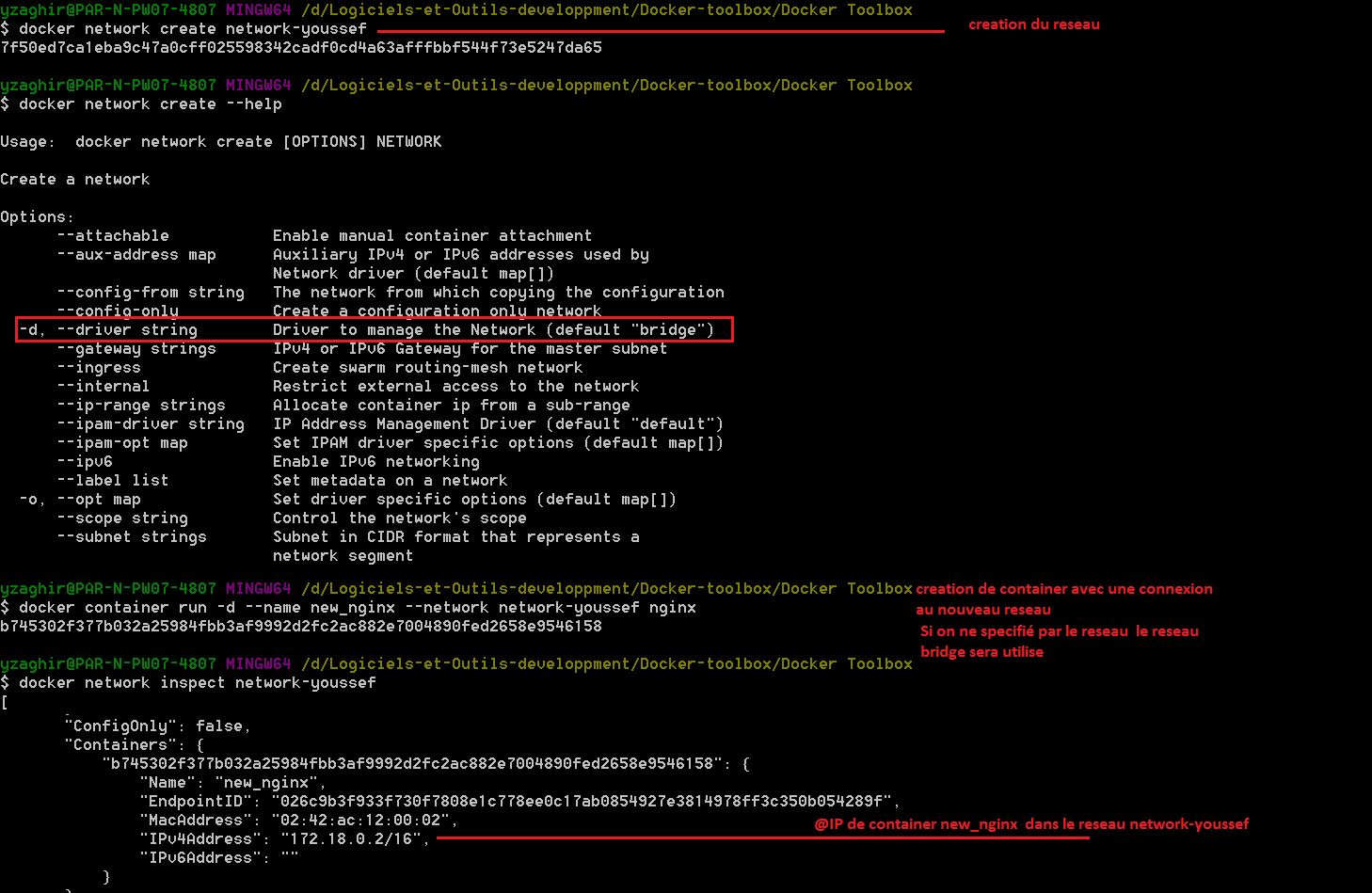


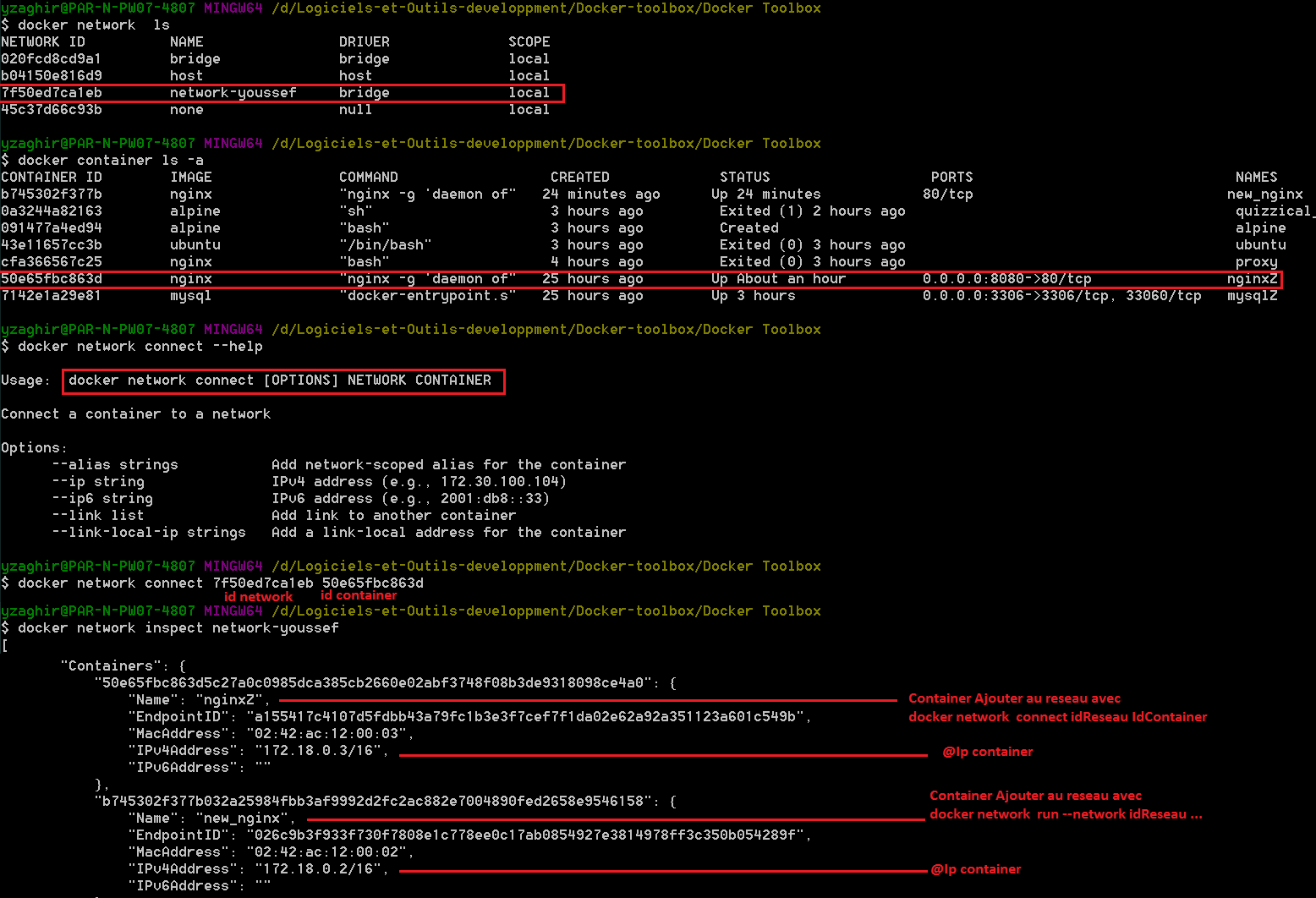
**Le réseau host** , it gain performance by skiggins virtual networks but sacrifices security of container model

**Le réseau none** , remove eth0 and only leave you with localhost interface in container

docker network create network-youssef

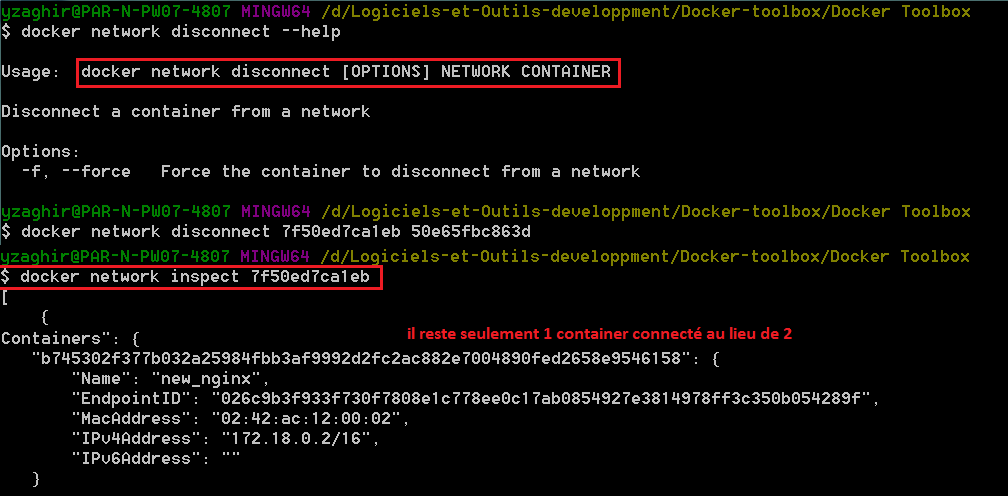
création d’un nouveau réseau, si on ne spécifie pas le réseau le driver bridge sera utilisé



Connecter un container à un réseau docker network connect [OPTIONS] IdNETWORK IdCONTAINER

Déconnecter un container d’un réseau :

docker network disconnect [OPTIONS] idNETWORK idCONTAINER



**Docker networks DNS and how containers find each others**

Dans cette partie on va voir

* Understand how DNS is the key to easy inter-container comms
* See how it work by default with custom networks
* Learn how to use --link to enable DNS on default bridge network

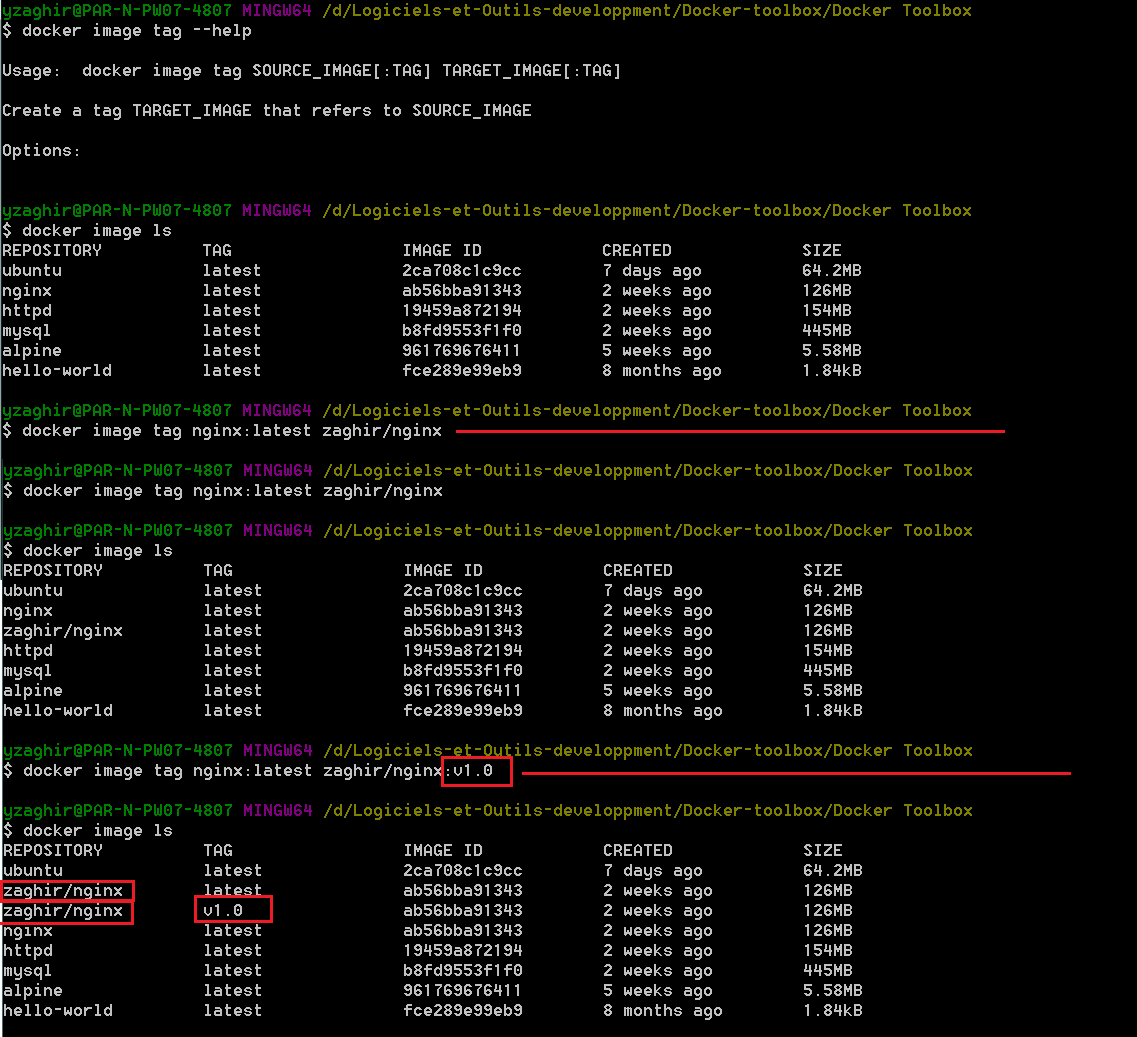
Static IP’s and using IP’ for talking to containers is an anti-patern .Do your best to avoid it.

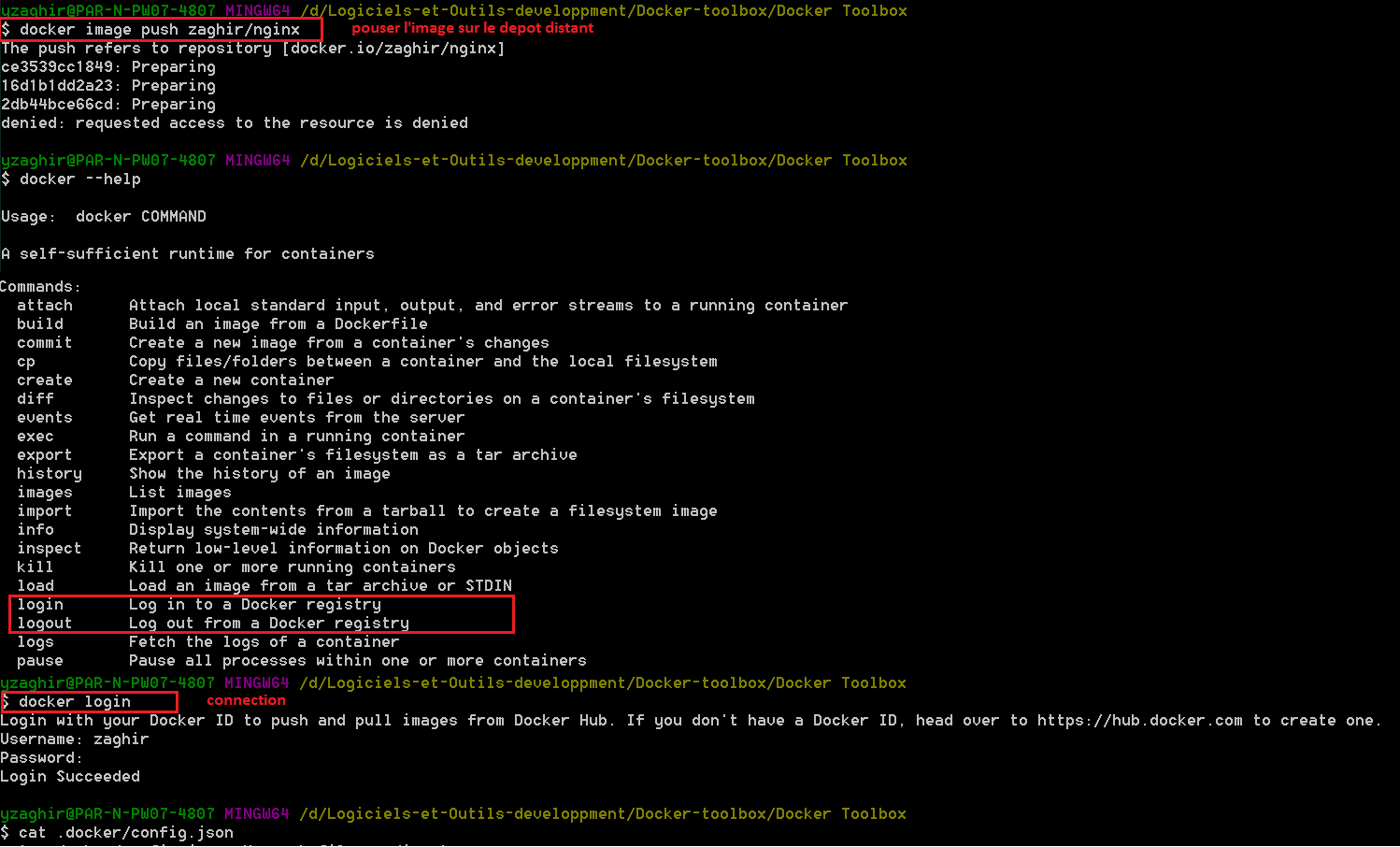
Docker DNS : Docker deamon has a built-in DNS server that containers use by default

DNS default names :Docker defaults the hostname te the container’s name, but you can also set aliases

La résolution des Docker DNS permet de résoudre le problème de mapping en l IP et le nom,

Si les containers sont dans le même réseau docker grace au dns les containers communique facilement





docker image push zaghir/nginx

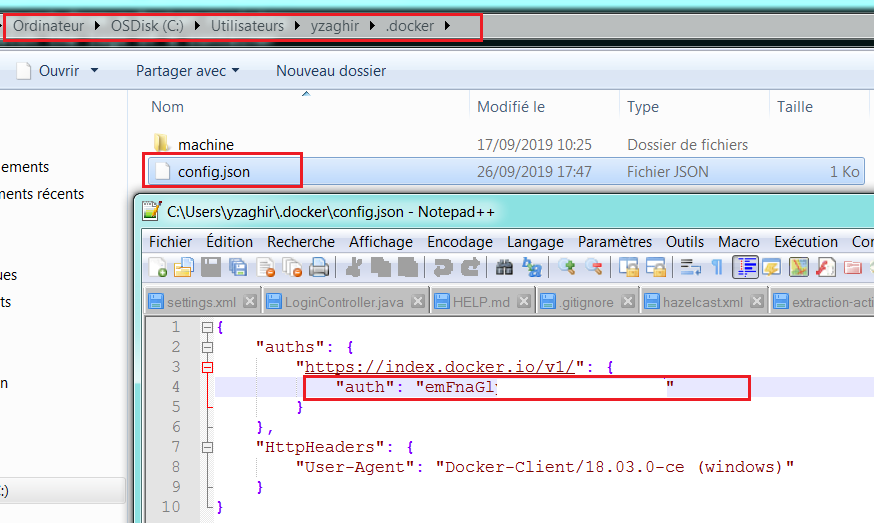
Pousser l’image sur le dépôt distant

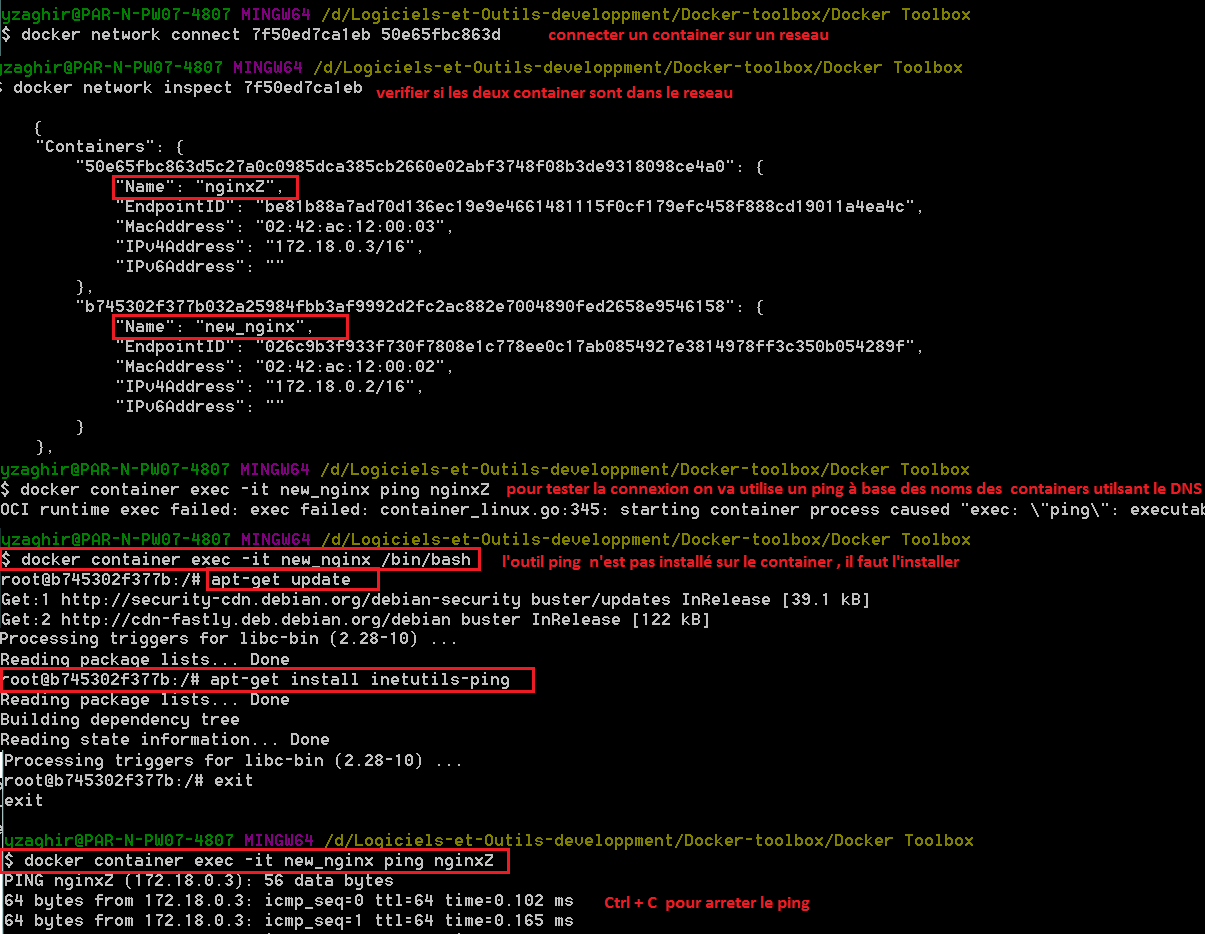
docker login

Connection au dépôt distant

Docker logout

Deconnect du dépôt distant, il faut se déconnecter si on finit notre travail parce que la clé d’authentification est dans le fichier config.json dans le dossier docker





**Docker Images**

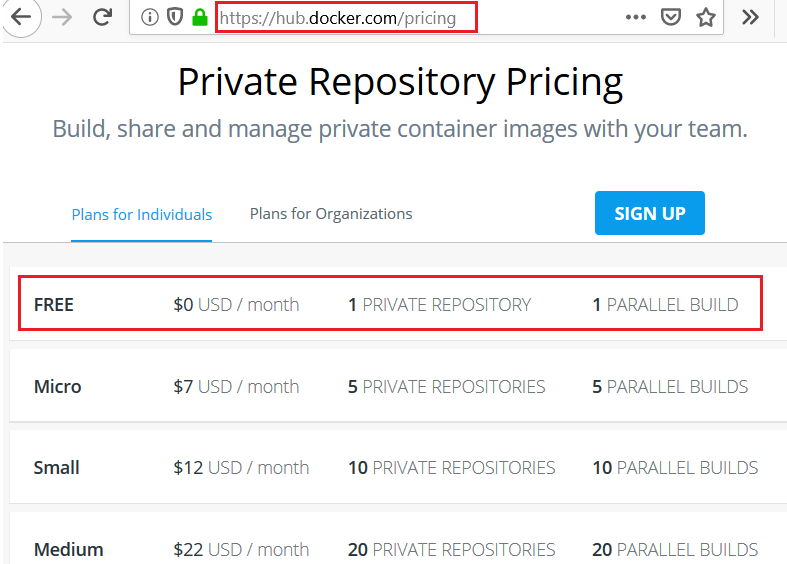
Dans cette section on va voir :

* All About images , the building blocks of containers
* What’s in an image(and what isn’t)
* Using docker hub repository
* Managing our local image cache
* Building our own images

What’s in an image (And what isn’t)

* App binaries and dependencies
* Metadata about the image data and how to run the image
* Official definition:” An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for une within a container runtime”
* Not a complete OS. No kernel, kernel module(e.g. drivers)
* Small as one file(your app binary) like golang static binary
* Big as Ubuntu distribution with apt , and Apach n PHP, and more installed

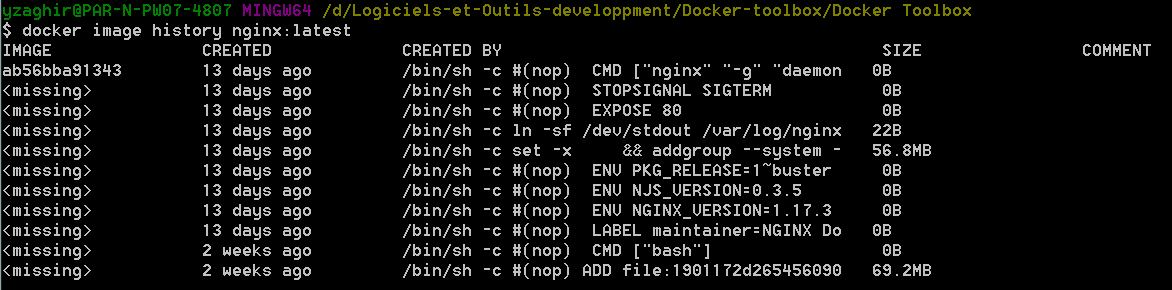
Docker utile un gestionnaire des packages pour chercher ses images , il y a deux type de dépôt les dépôt local la ou les images sont stockés après leur téléchargement et un dépôt distant : <https://hub.docker.com> , sur le dépôt distant on peut trouver des images officielles et non officielles. Nos images créer en local c’est possible de les publié dans docker hub et on a la possibilité de les rendre public consultable par tout le mon ou privé



**Image layers**

docker image history nom-Image:version²²

exemple : docker image history nginx:latest

Cette commande affiche le historique des différents layers avec les changements faits sur l’image depuis la création d’image , chaque modification sur l’image crée un nouveau layer(une couche)

Quand on fait un changement dans l’image on peut modifier le système de fichier du coup la taille de changement plus grande ou seulement modifier les métadonnées dans une taille de changement qui égale à 0

Au moment de la création de l’image ou modification une nouvelle couche (Layer) et créer avec un nouveau code SHA .A avec ce mécanisme on enregistre une seule fois l’image docker ,dans ce sens on a pas besoin des couches de même type si en upload ou download

Docker utilise un gestionnaire de paquets , si une image fait appel à d’autre images et ces images sont déjà dans le dépôt local , docker récupère seulement l’image qui n’existe pas dans son dépôt

**e019479f5894**

**d019479f5894**

**5**

**ID : d019479f5894**

**Parent : "c719479f5894"**

**Name : "copie A"**

**ID : e019479f5894**

**Parent : "c719479f5894"**

**Name : "copie B"**

**4**

**ID : c019479f5894**

**Parent : "b719479f5894"**

**Name : "port"**

**3**

**c019479f5894**

**ID : b019479f5894**

**Parent : "a719479f5894"**

**Name : "Apach"**

**2**

**b019479f5894**

**ID : a719479f5894**

**Parent : ""**

**Name : "Custom Img"**

**a719479f5894**

**1**

**Layer**

**Image**

L’image copie A et copie B utilisent les mêmes images (custom img , Apach , port) docker enregistre une seule copie qui va être utiliser après par les autre ce qui permet d’ optimiser les ressources

Docker image inspect image-name

Docker image inspect nginx

This command returns Json metadata about the image

**About Image tags :**

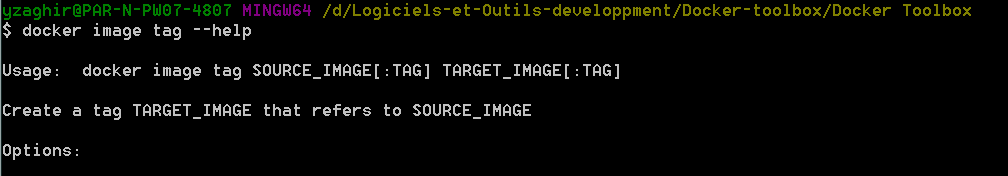
Docker image tag

This command assign one or more tags to an image

The default tag is latest is it not specified

Official repositories: they live at ‘root of namespace ’ or the registry , so they don’t need account name in front of repo name

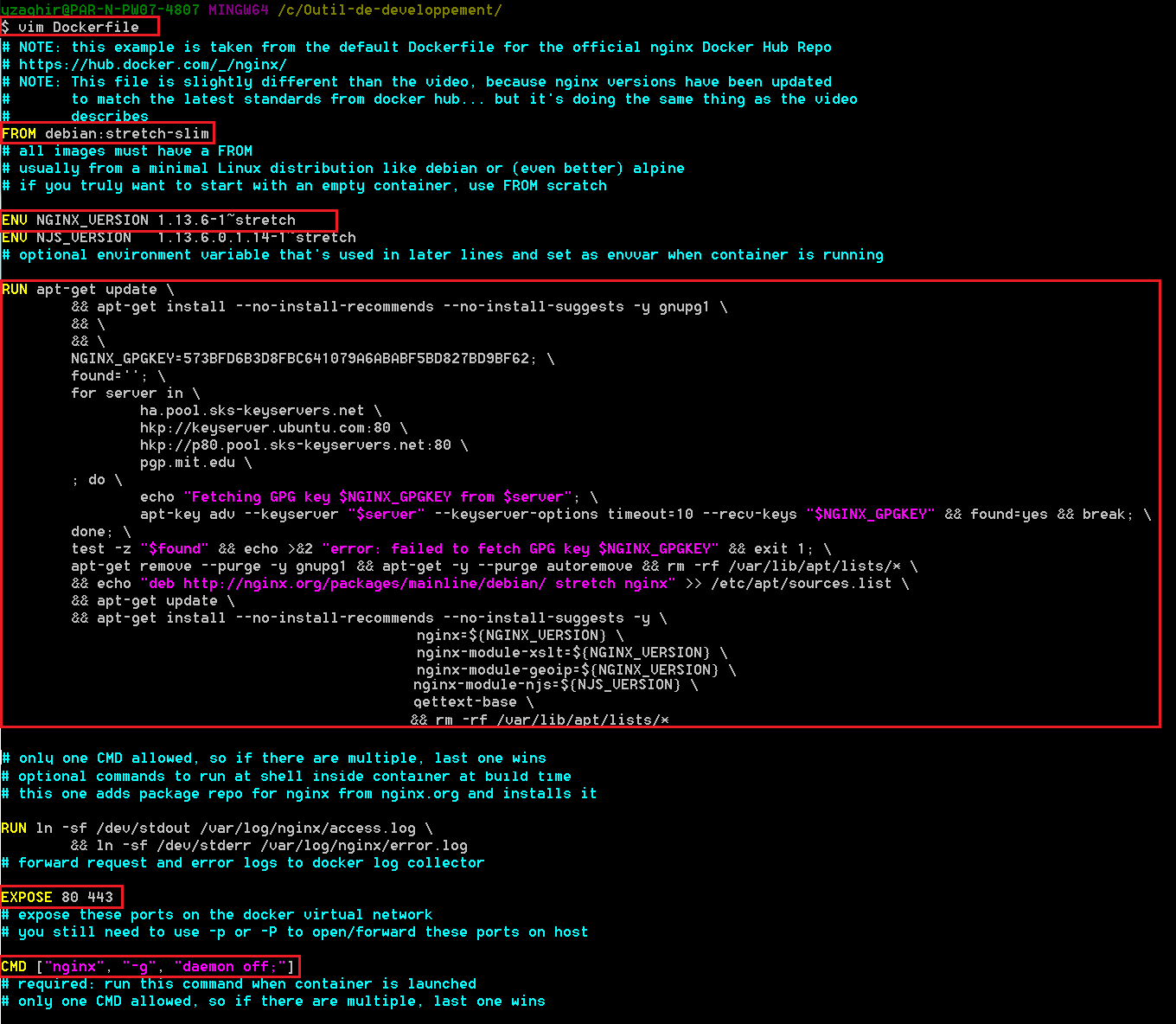
Ex : mysql/mysql-server



Dockerfile c’est la recette pour créer l’image

Docker build –f some-dockerfile

-f = file pluis le chemin de fichier

Cette commande permet de builder une image

**FROM :**

The from command is in every Dockerfile. It's required to be there.

It's normally a minimal distribution.

In this case it's Debian.

But a lot of times nowadays it's going to be Alpine.

And really, the reason you would use these is to save yourself time and pain. Because these minimal distributions

are actually much smaller than the CDs you would use to install a virtual machine from them.

For example, the Ubuntu one doesn't even have cURL in it, whereas obviously, if you installed a full Ubuntu installation on a VM it would have cURL and a lot of other commands already installed.

Because all of these distributions are official images, it means that they're going to be always up to date with the latest security patches and you can depend and trust on them.

And one of the main benefits for using them in containers is to use their package distribution systems to install whatever software you need in your package.

In this case, we're actually using the Jessie release, which is what's after the colon.

And, as you remember from our Docker Hub conversation, there's lots of different options you can specify here for the Debian repository.

**ENV:**

ENV is for environment variables.

It's a way to set environment variables, which are actually very important in containers because they're , the main way we set keys and values for container building and for running containers.

In this case, it's actually setting the version of Nginx it would like us to install and this environment, variable will be set so that any subsequent lines will be able to use it.

**RUN :**

RUN is for executing a command

Like install a software with package repository, or you need to do some unzipping , or some file edits inside the container itself

Run commands can also run shell scripts that you've copied in earlier in the file or any commands that you can access from inside the container at that point in time in the file.Since we're coming from Debian, this run command has access to all the commands and binaries that would have been installed with that release.

And in this one for Nginx, this list of commands is more or less adding a key to the repository where

you can get the package to install the latest Nginx from the official Nginx repository.

There's a few things here that are really key in making good Dockerfiles and we'll actually talk

about this more in a later section where we talk about best practices.

But two things to note.

The reason that we're adding all these commands with the double ampersand here, so that they're chained one after the other, is because, if you remember, each stanza is its own layer.

What this does is ensures that all of these commands are fit into one single layer.

It saves us a little time.

It saves us space. And it's so common that you'll probably see it in every Dockerfile on Docker Hub.

**LOG :**

This one is all about pointing our log files to the stdout and to the stderr.

We'll see later, the proper way to do logging inside a container is to not log to a log file. And there's

no syslogd or any other syslog service inside a container. Docker actually handles all of our logging for us. All we have to do inside the container is make sure that everything we want to be captured in the logs is spit to stdout and stderr and docker will handle the rest.

There's actually logging drivers that we can use in the Docker Engine itself to control all the logs for all the containers on our host.

And so that's really what you want to do with them.

It adds more complexity to your app if your app is actually doing the logging itself.

And then, if you have to deal with files in every container, now you've got a problem of how do you get those files out, and searchable, and accessible. Here, we're taking the default Nginx logs and we're actuallylinking them to the stdout.So that Docker can capture them.We also have an expose command.

**EXPOSE :**

By default, no TCP or UDP ports are open inside a container. It doesn't expose anything from the container to a virtual network unless we list it here.

And, of course, because this is a Web and proxy server it's going to expose 80 and 443.

Now, this expose command does not mean these ports are going to be opened automatically on our host.That's what the dash P command is whenever we use docker run.

**CMD**

Lastly, we have the CMD here. The CMD is a required parameter that is the final command that will be run every time you launch a new container from the image, or every time you restart a stopped container.

There is some really excellent documentation on all of these stanzas, plus a whole lot more, that

we're going to go into later, on the Docker documentation website at docs.docker.com,

which you can find in the resources of this lecture.

**Cree rune image**

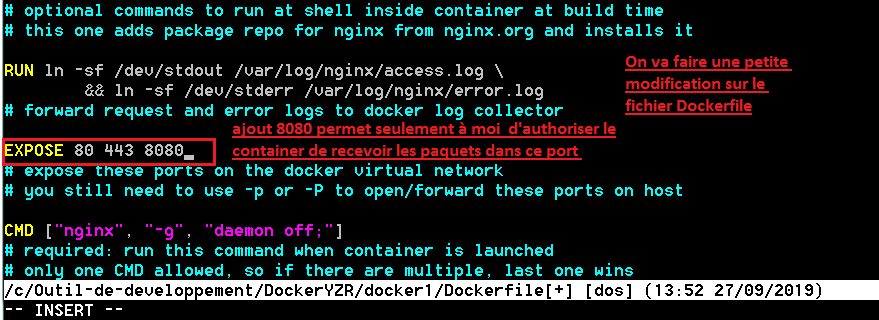
docker image build -t customnginxz .

-t = tag , il faut respecter le format de nommage de docker , il faut un nom tout en minuscule



Petite modification sur le Dockerfile de notre image créé

vim Dockerfile





Le contenu de fichier Dockerfile utilisé

# NOTE: this example is taken from the default Dockerfile for the official nginx Docker Hub Repo

# https://hub.docker.com/\_/nginx/

# NOTE: This file is slightly different than the video, because nginx versions have been updated

#       to match the latest standards from docker hub... but it's doing the same thing as the video

#       describes

FROM debian:stretch-slim

# all images must have a FROM

# usually from a minimal Linux distribution like debian or (even better) alpine

# if you truly want to start with an empty container, use FROM scratch

ENV NGINX\_VERSION 1.13.6-1~stretch

ENV NJS\_VERSION   1.13.6.0.1.14-1~stretch

# optional environment variable that's used in later lines and set as envvar when container is running

RUN apt-get update \

    && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 \

    && \

    NGINX\_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \

    found=''; \

    for server in \

        ha.pool.sks-keyservers.net \

        hkp://keyserver.ubuntu.com:80 \

        hkp://p80.pool.sks-keyservers.net:80 \

        pgp.mit.edu \

    ; do \

        echo "Fetching GPG key $NGINX\_GPGKEY from $server"; \

        apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys "$NGINX\_GPGKEY" && found=yes && break; \

    done; \

    test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX\_GPGKEY" && exit 1; \

    apt-get remove --purge -y gnupg1 && apt-get -y --purge autoremove && rm -rf /var/lib/apt/lists/\* \

    && echo "deb http://nginx.org/packages/mainline/debian/ stretch nginx" >> /etc/apt/sources.list \

    && apt-get update \

    && apt-get install --no-install-recommends --no-install-suggests -y \

                        nginx=${NGINX\_VERSION} \

                        nginx-module-xslt=${NGINX\_VERSION} \

                        nginx-module-geoip=${NGINX\_VERSION} \

                        nginx-module-image-filter=${NGINX\_VERSION} \

                        nginx-module-njs=${NJS\_VERSION} \

                        gettext-base \

    && rm -rf /var/lib/apt/lists/\*

# optional commands to run at shell inside container at build time

# this one adds package repo for nginx from nginx.org and installs it

RUN ln -sf /dev/stdout /var/log/nginx/access.log \

    && ln -sf /dev/stderr /var/log/nginx/error.log

# forward request and error logs to docker log collector

EXPOSE 80 443 8080

# expose these ports on the docker virtual network

# you still need to use -p or -P to open/forward these ports on host

CMD ["nginx", "-g", "daemon off;"]

# required: run this command when container is launched

# only one CMD allowed, so if there are multiple, last one wins

Création d’une image avec un server Nginx et une page index modifié

Fichier Dockerfile

# this same shows how we can extend/change an existing official image from Docker Hub

FROM nginx:latest

# highly recommend you always pin versions for anything beyond dev/learn

WORKDIR /usr/share/nginx/html

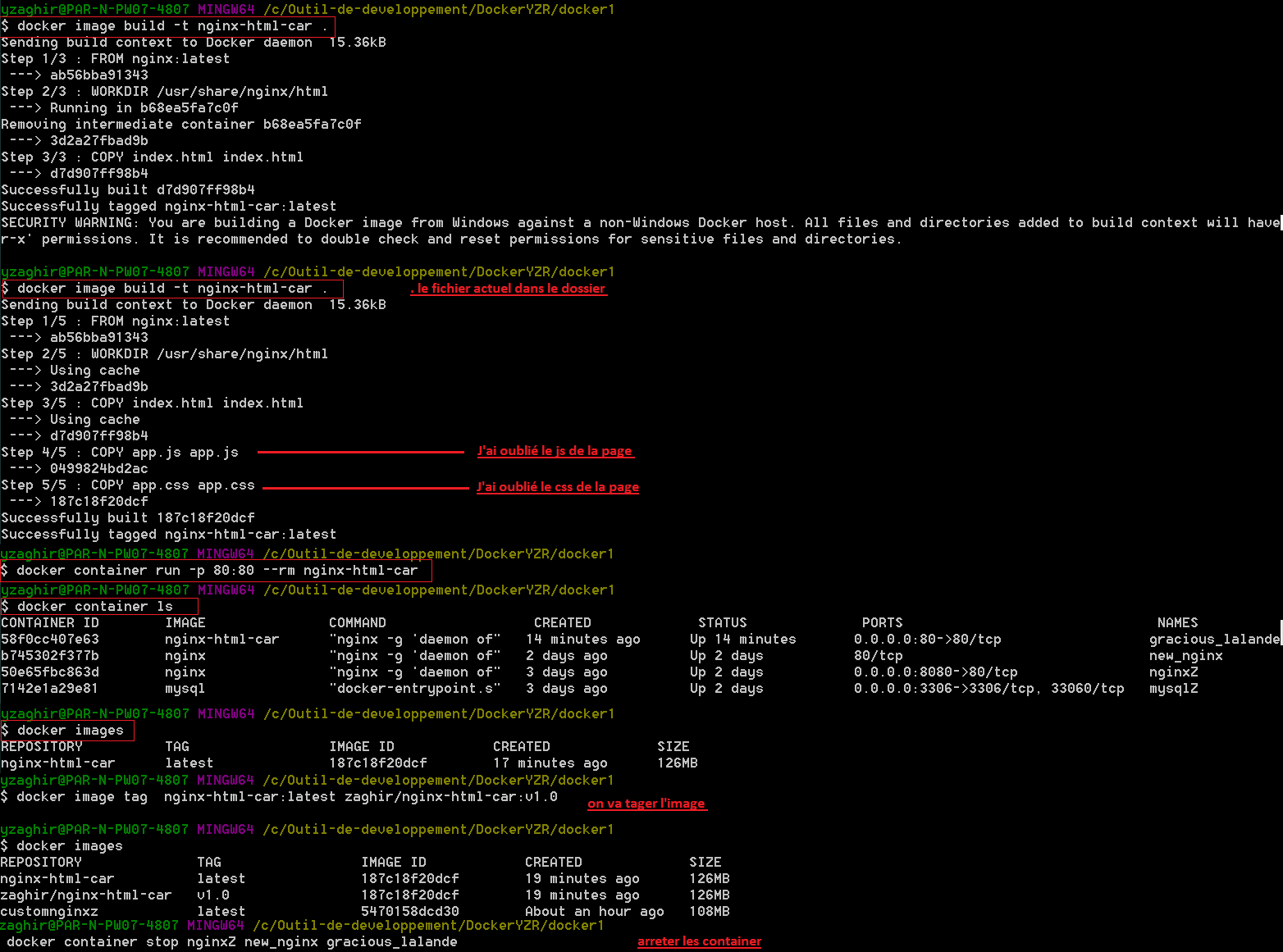
# change working directory to root of nginx webhost

# using WORKDIR is preferred to using 'RUN cd /some/path'

COPY index.html index.html

COPY app.js app.js

COPY app.css app.css

# I don't have to specify EXPOSE or CMD because they're in my FROM

**Créer un image avec nodeJS**

On va

* Prendre une application NodeJS et la dockoriser
* Prendre un dockerfile , le créé le testé , le puché, (le rm ), l’exécuté
* Utilisation de nodeJS basé sur l’image Alpine
* Excepter le résultat du site sur <http://localhhost>
* Pag et push dans votre compte (free) docker hub
* Supprimer votre image dans le cache locale, et exécuter encore une fois à partir hub

**Persistant data**

Vous venez de créer vos premiers conteneurs Docker et peut-être ne vous êtes pas encore demandé ce qu’il advient des données stockées à l’intérieur. Alors par défaut, les données contenues dans le conteneur disparaissent tout simplement avec lui lors de sa destruction. Ce peut être le comportement voulu dans la plupart des cas, mais il est parfois nécessaire de rendre les données persistantes. Par exemple, une base de données utilisateur ne devrait pas disparaitre lorsque l’on supprime le conteneur contenant le moteur de la base de données !

Docker offre plusieurs manières de gérer les données persistantes. Nous verrons dans un premier temps la notion de volume de données puis les conteneurs de données dans un deuxième article.

**Solution 1 : Data volumes ou volume de données**

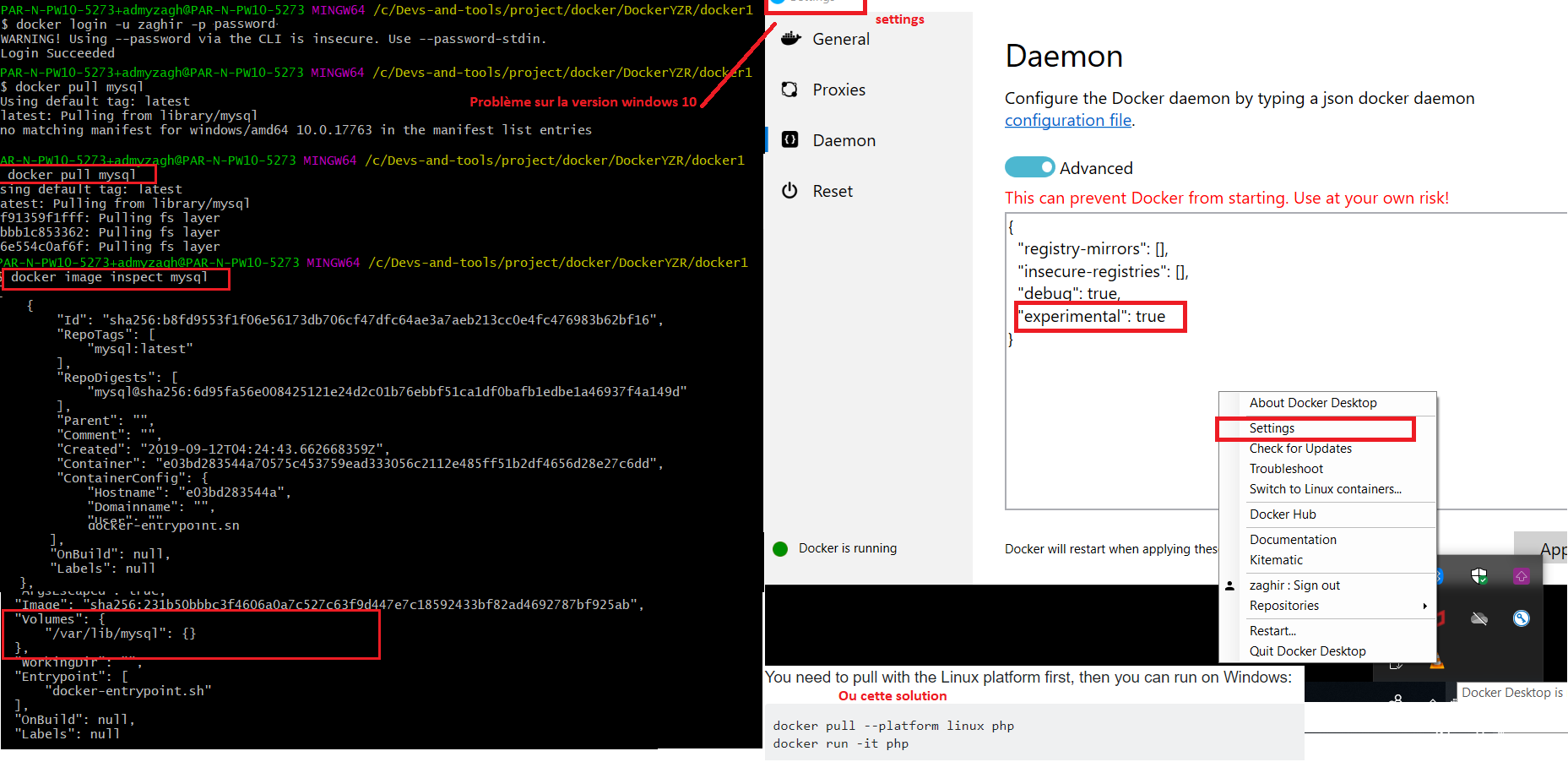
Docker offre le concept de volume de données. L’idée est de stocker les données dans un répertoire spécifique du serveur hôte Docker plutôt que dans le système de fichiers par défaut du conteneur. Ce stockage externe, appelé volume de données, permet de rendre les données indépendantes de la vie du conteneur. Le volume permet également de partager ces données entre plusieurs conteneurs.

Le volume de données dans Docker propose ces fonctionnalités :

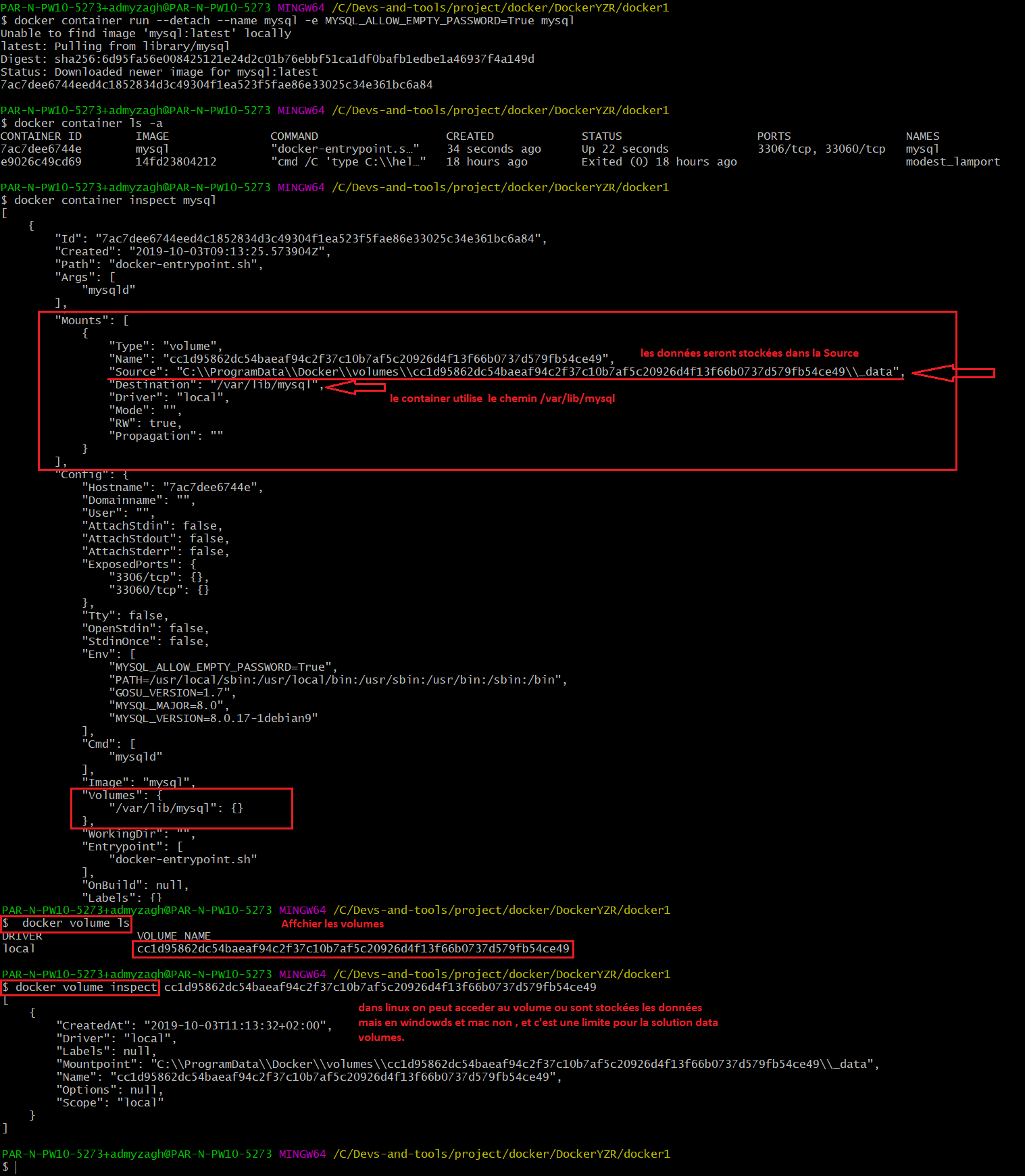
* Le volume est initialisé à la création du conteneur. Si l’image de base contient déjà des données à l’initialisation du volume, alors elles seront copiées dans le volume
* Un volume de données peut être partagé ou réutilisé par plusieurs conteneurs
* Les changements dans le volume de données sont appliqués immédiatement
* Le volume de données est persistant même si le conteneur lui-même est supprimé
* NB : A noter que Docker ne supprimera jamais automatiquement un volume même si plus aucun conteneur n’y fait référence. Ce sera à vous de faire le ménage !

Demo

On va récupérer une image de mysql , les données saisi dans la base doivent être persister même si on a plus de container mysql , car les données sont plus importantes que le container



On va pas voir Dockerfile parce qu’il n’est pas maintenant dans les metadata

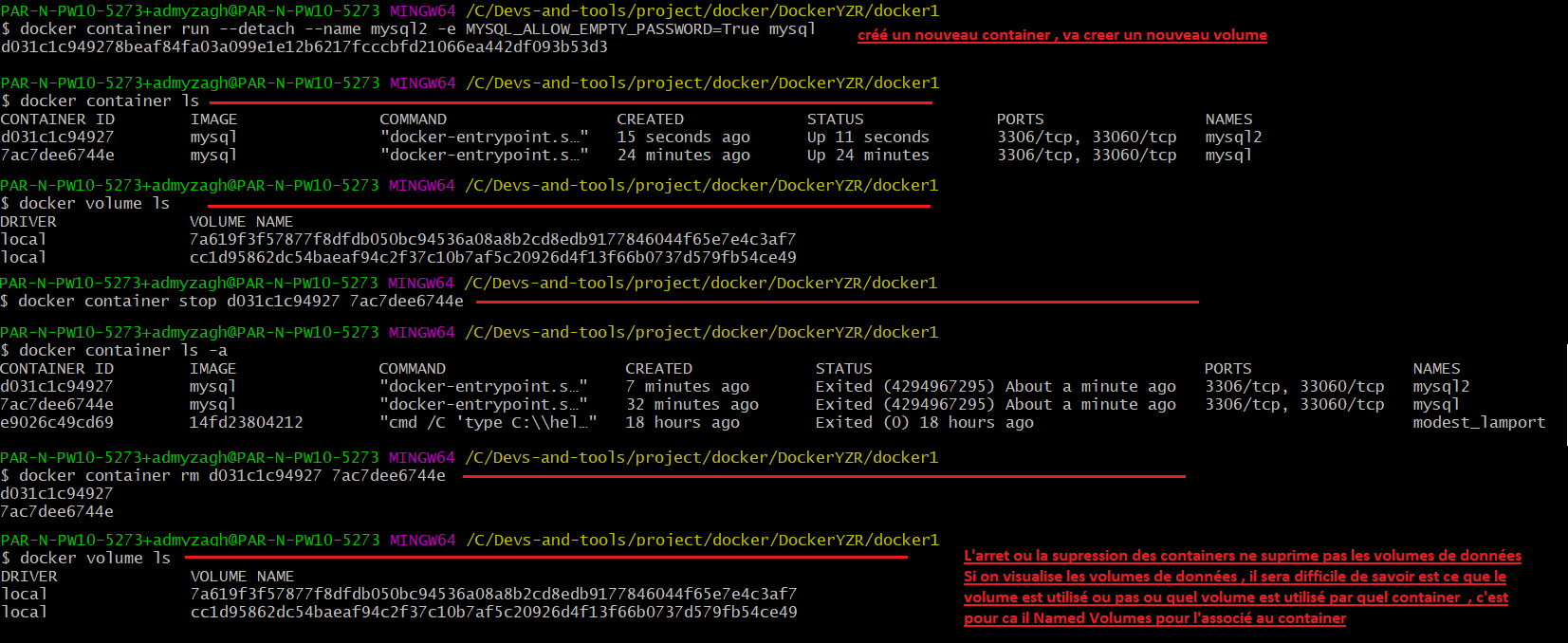


docker volume ls

afficher les volumes

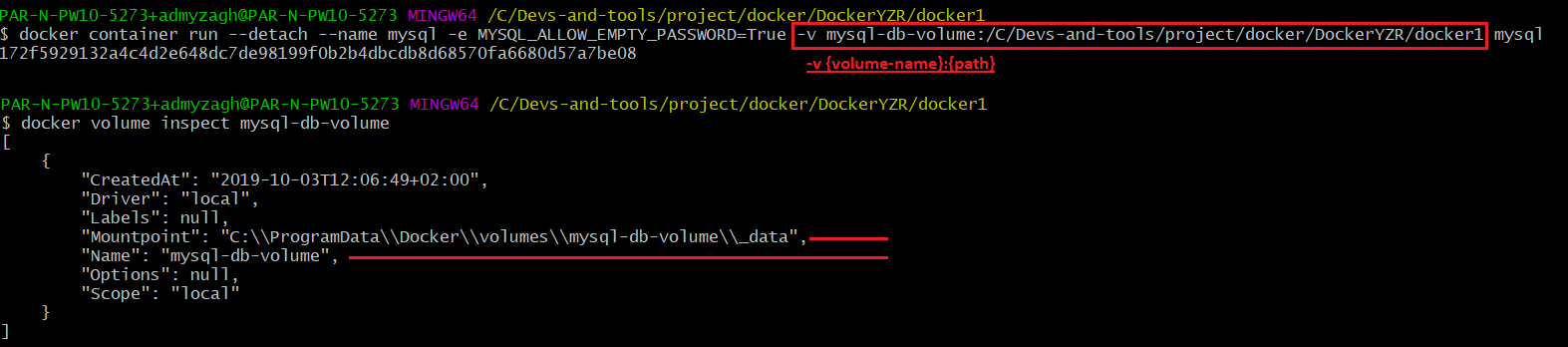
docker volume inspect shaIDVolume

inspecter la configuration du volume

Remarque :

Definier un volume pour le container

Docker -v ,{volume-name} :{path}

Meme si on ajoute un nouveau container avec le volume mysql-db-volume , le nouveau volume ne sera pas créé

**Solution 2 : Data bind mounting**

Cette solution permet de câbler directement les dossiers ou les fichiers dans la machine host avec les données utilisées dans les containers

C’est très utile pour les développeurs

Démo

On va utiliser une image de nginx

Dockerfile est dans le dossier de travail

# this same shows how we can extend/change an existing official image from Docker Hub

FROM nginx:latest

# highly recommend you always pin versions for anything beyond dev/learn

WORKDIR /usr/share/nginx/html

# change working directory to root of nginx webhost

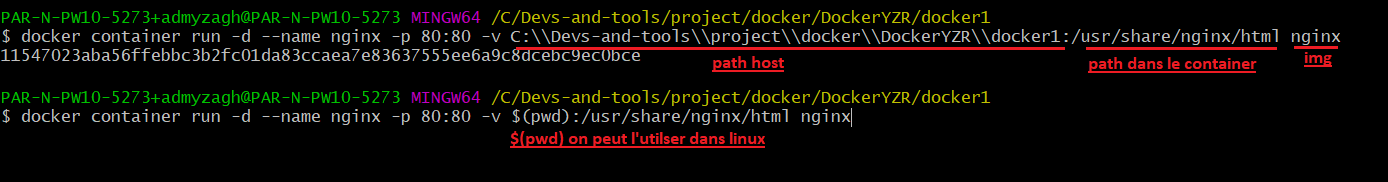
# using WORKDIR is preferred to using 'RUN cd /some/path'

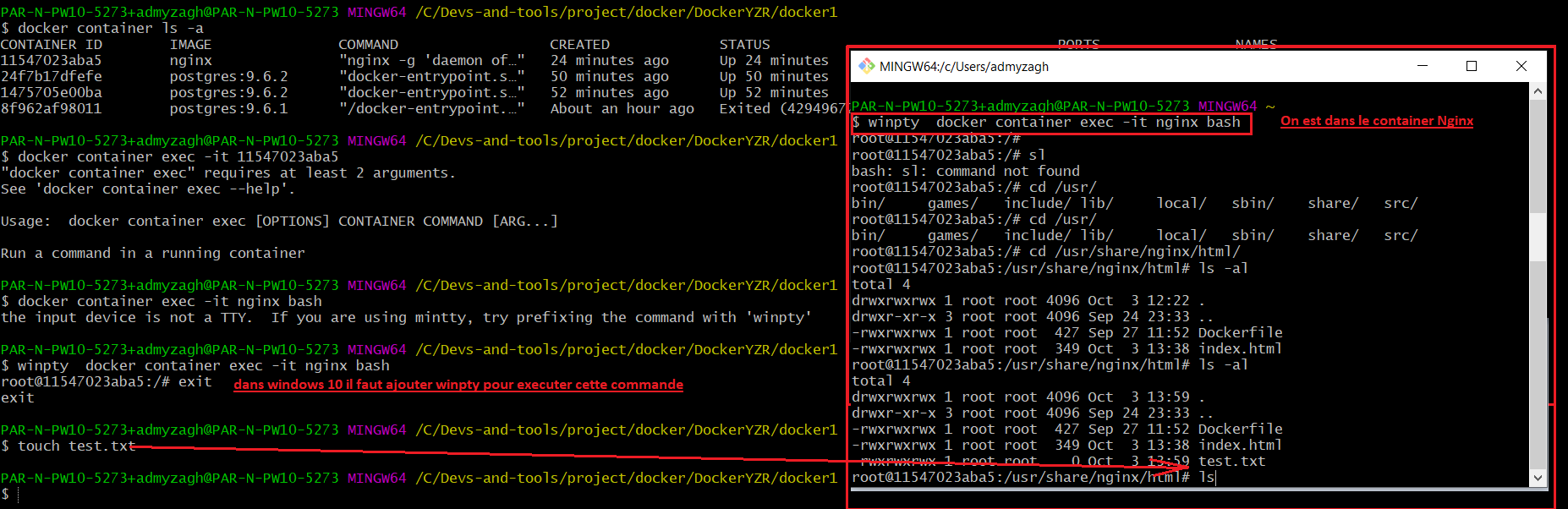
COPY index.html index.html

# I don't have to specify EXPOSE or CMD because they're in my FROM

docker container run -d --name {container-name] -p 80:80 -v{path in host}:{path in container} {img-name}

ca permet de synchroniser les données utilsées par le container et la machine local (Host)

docker container run -d --name nginx -p 80:80 -v C:\\Devs-and-tools\\project\\docker\\DockerYZR\\docker1:/usr/share/nginx/html nginx



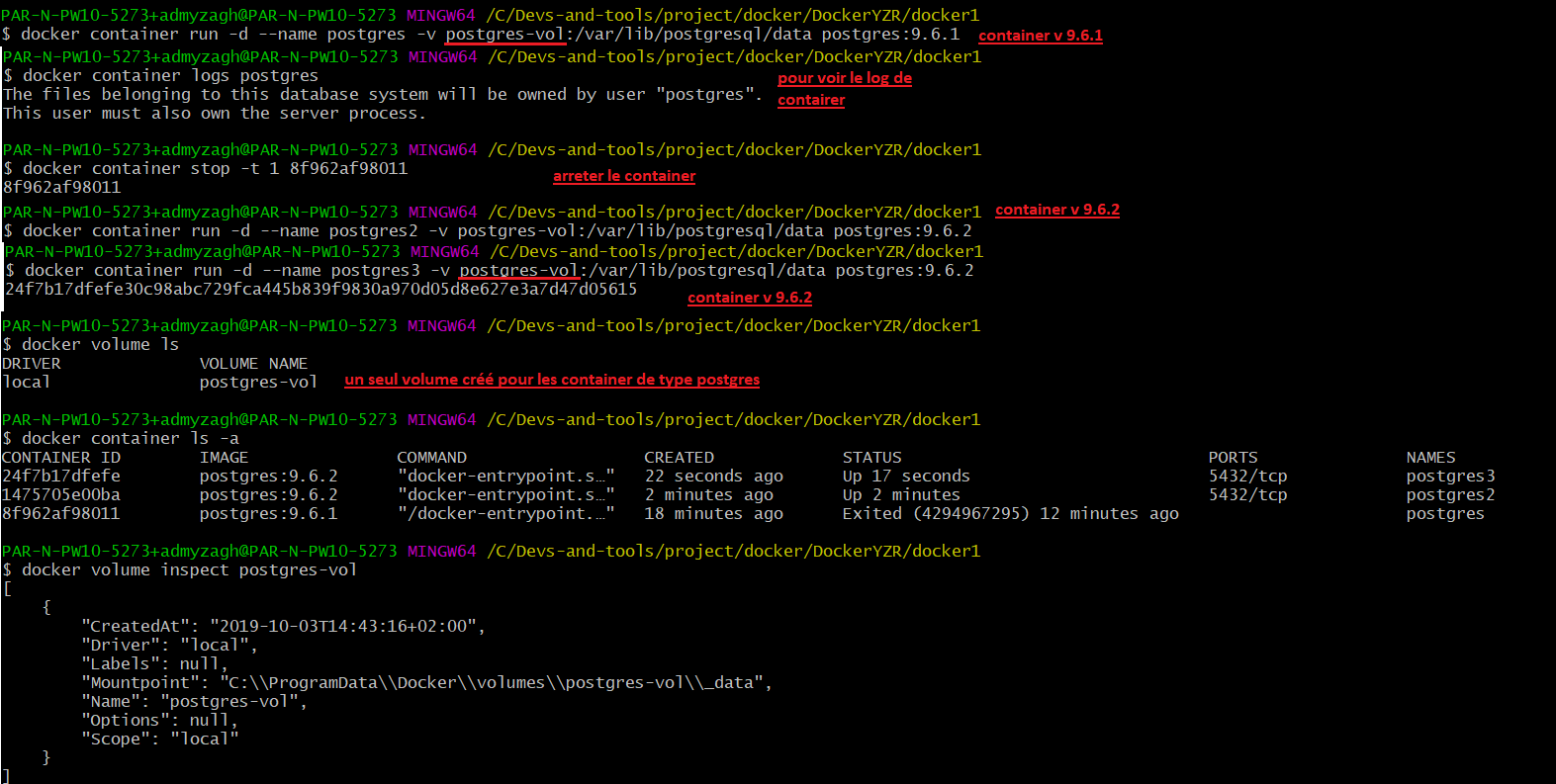
faire un test avec http://localhost

Démo

On va utiliser deux version de Postgres  la version 9.6.1 et 9.6.2

Le but est d’utiliser les données des bases créer par la version 9.6.1 dans la version 9.6.2 , on peut monter de version sans touché au données

Pour concrétiser ca on va utiliser data volumes pour échanger les données

$ docker container run -d --name postgres -v postgres-vol:/var/lib/postgresql/data postgres:9.6.1

**Docker Compose**

Why : configure relationship between containers

Why : save our docker container run setting in easy-to-read file

Why : create one-liner developper environment startups

Comprised of 2 separate but related thing

* 1 YAML-formatted file that describes our solution option for :
* Containers
* Networks
* Volumes
* 2 A CLI tool docker-compose used for local dev/test automation with those YAML files
* Il y a plusieurs versions(1 ,2,2.2,3,3.1) de docker-compose.yml
* Depuis la version 1.13 docker-compose.yml est utilisé directement en prod avec Swarm
* docker-compose.yml et le nom de fichier par défaut mais en peut utiliser d’autre noms des fichiers avec docker-compose -f

Les commandes les plus utilisées :

docker-compose up #setup volumes/networks and start all containers

docker-compose up #stop all containers and remove containers/volumes/networks

Ces deux commandes permettent de partir sur un environnement à un autre pour les développeurs. Pour les développeurs ca sera mieux d’utilisé vagran ou d’autre outils pour optimiser les environnements en local qui seront utiliser en développement des applications

docker-compose permet de récupérer rapidement à partir des dépôts un environnement près configurer pour un nouveau développeur

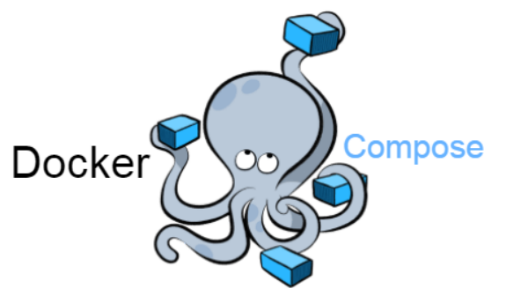
Si tous les projets ont un Dockerfile et docker-compose.yml , le nouveau développeur peut récupérer l’environnement avec les commandes :

git clone gitthub.com/some/software

docker-compose up

docker-compose up -d , lancer docker-compose en arrière-plan

docker-compose ps , afficher tous les docker-compose qui sont en exécutions

docker-compose down -v , arrêter les containers et les volumes aussi (-v)

**docker-compose up** démarre les services décrits dans mon docker-compose.yml et ne me rend pas la main.  
**docker-compose up -d** fait la même chose mais me rend la main une fois que les services sont démarrés.  
**docker-compose up –build** reconstruit les services avant de les lancer.

**docker-compose down** stoppe les services.

**docker-compose restart** redémarre l’ensemble des services.  
**docker-compose restart nginx** redémarre un des services (ici **nginx**).

**docker-compose exec rails bash** me fournit une console bash au sein du conteneur **rails**.  
**docker-compose exec rails bin/rails db:migrate** effectue un **rails db:migrate** au sein du conteneur rails.

**docker-compose logs** me retourne l’ensemble des logs des services depuis le dernier démarrage et me rend la main.  
**docker-compose logs -f** affiche les logs des services et continue à les « écouter » sans me rendre la main.  
**docker-compose logs -f rails** fait la même chose pour le conteneur **rails** uniquement.

Toutes les commandes passées dans l’invite de commande pour créer un container à partir d’une image avec un volume de stockage et network pour la communication entre les containers, on automatisent tout dans le docker-compose

Docker-compose permet de nous créer une image en exécution. En fait, il recherchera des images dans le cache et, s'il contient des options de construction, il construira cette image lorsque vous utiliserez la commande docker-compose up. Cela ne construira pas l'image à chaque fois, Il ne le construira que s'il ne trouve pas l’image. C’est possible d’utiliser docker-compose build ou docker-compose –build

Utilisation :

Créer un dockerfile pour créer l’image : ici nginx.Dockerfile

Créer un docker-composer.yml pour configurer les containers des nginx

Exécuter la commande docker-compose up

Faire un fichier de configuration pour le server nginx : nginx.conf

Docker-compose.yml

version: '2.1'

# only build nginx.config into image

# uses sample site from  http://startbootstrap.com/template-overviews/agency

services:

  proxy:

    build:

      context: .

      dockerfile: nginx.Dockerfile

    image: nginx-custom

    ports:

      - '80:80'

  web:

    image: httpd

    volumes:

      - C:\\Devs-and-tools\\project\\docker\\DockerYZR\\volume

Sur version: '2.1' j’ai utilisé la version 2.1 car la v2 cause des problèmes sur windows

services :

build :

web :

C’est les deux services qu’on va créer avec docker-compose

build  chercher une image nginx custom , s’il ne le trouve pas il va récupérer le fichier dockerfile nginx.Dockerfile dans le répertoire actuel context: . point c’est ou en execute la commande, et il va créer l’image nginx-custom

web créer une image avec un volume

ngnix.Dockerfile

FROM nginx:1.11

COPY nginx.conf /etc/nginx/conf.d/default.conf

nginx.conf

server {

    listen 80;

    location / {

        proxy\_pass         http://web;

        proxy\_redirect     off;

        proxy\_set\_header   Host $host;

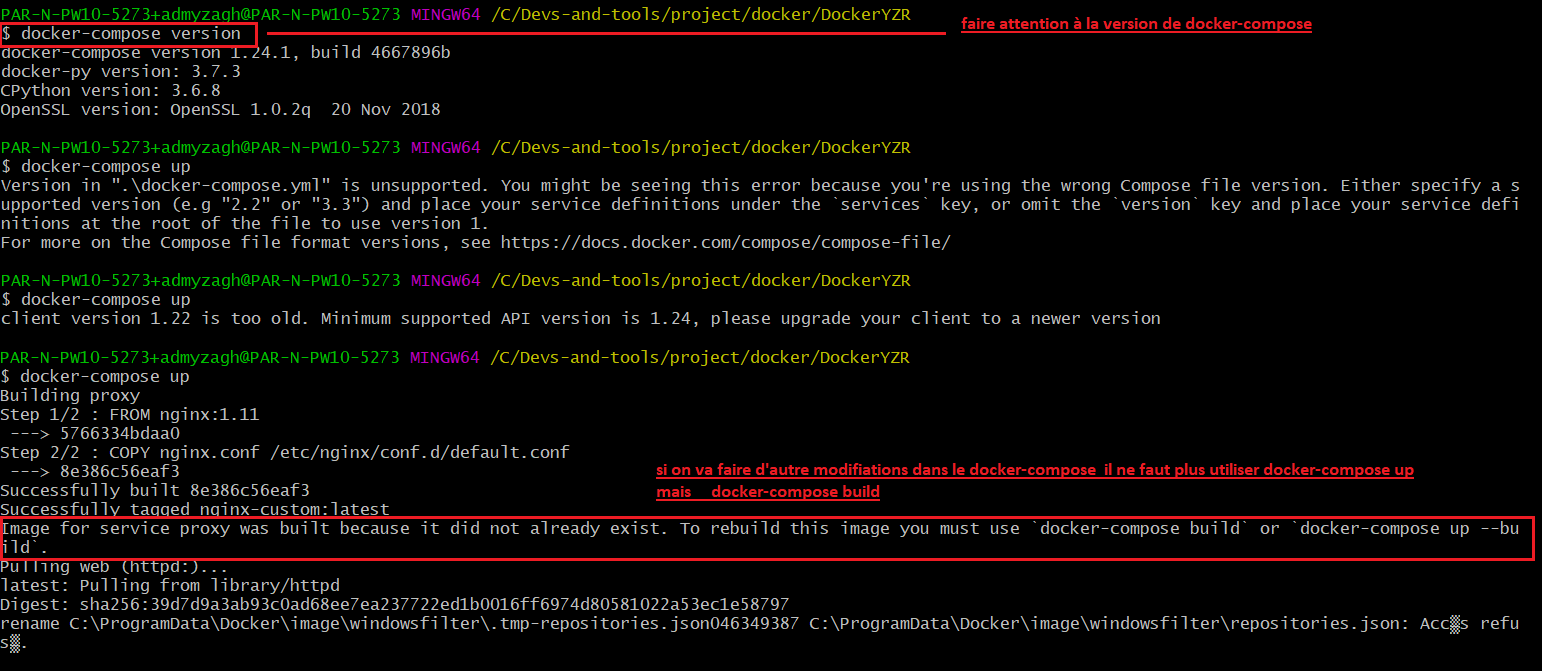
        proxy\_set\_header   X-Real-IP $remote\_addr;

        proxy\_set\_header   X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

        proxy\_set\_header   X-Forwarded-Host $server\_name;

    }

}



docker-compose version

