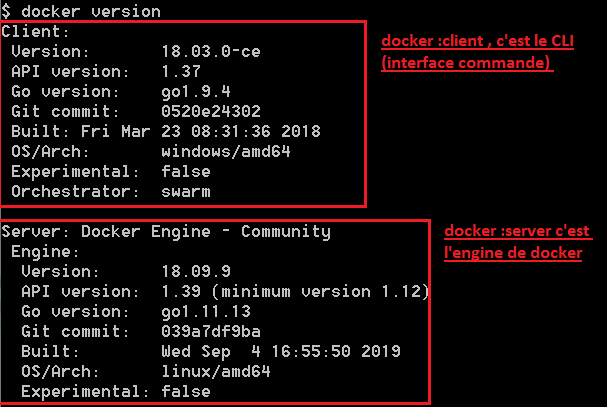
Containers ne sont pas des mini-VM

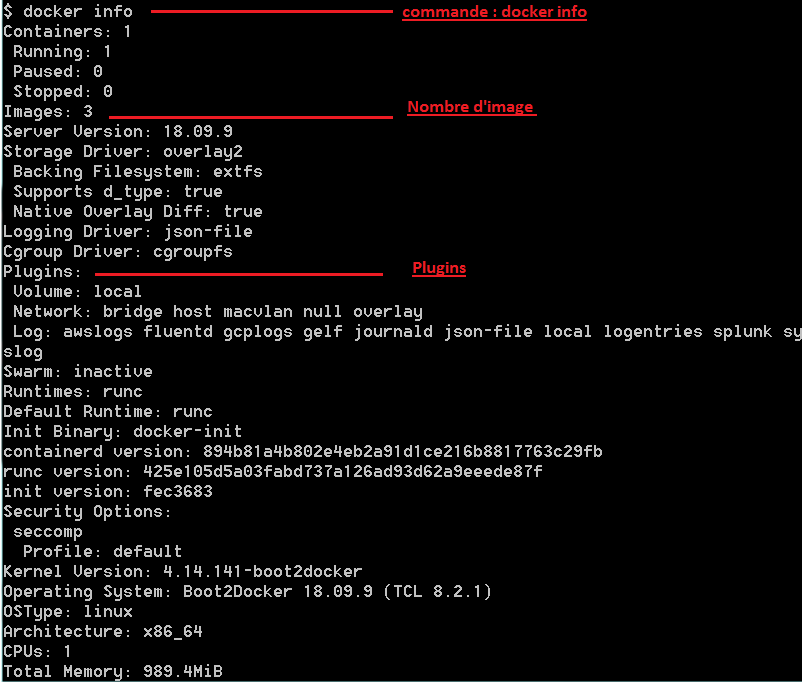
* C’est des seulement des processus
* Limité leur accès aux ressources
* Ils sont fermés quand les procès sont arrêtés

Docker commandes :

**Docker version**



**Docker info**



Docker images

Pour voir toutes les images installer

Docker container - -help

Avoir l’aide sur les containers

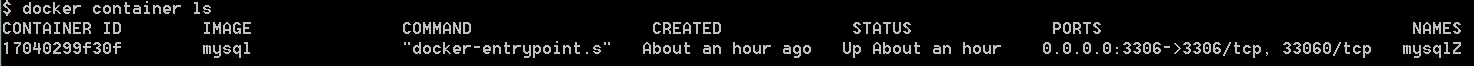
Docker container run - - publish 80:80 - -detach nginx

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

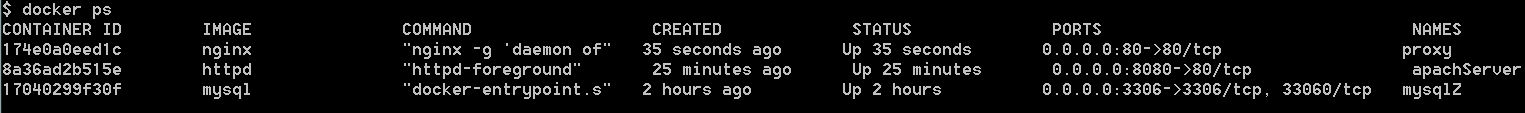
Exp : docker container rm 36f 553 d6c f00 , suprimer 4 containers

**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

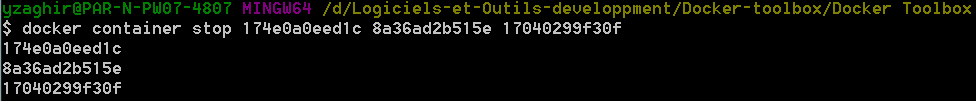
docker container ls -q quick

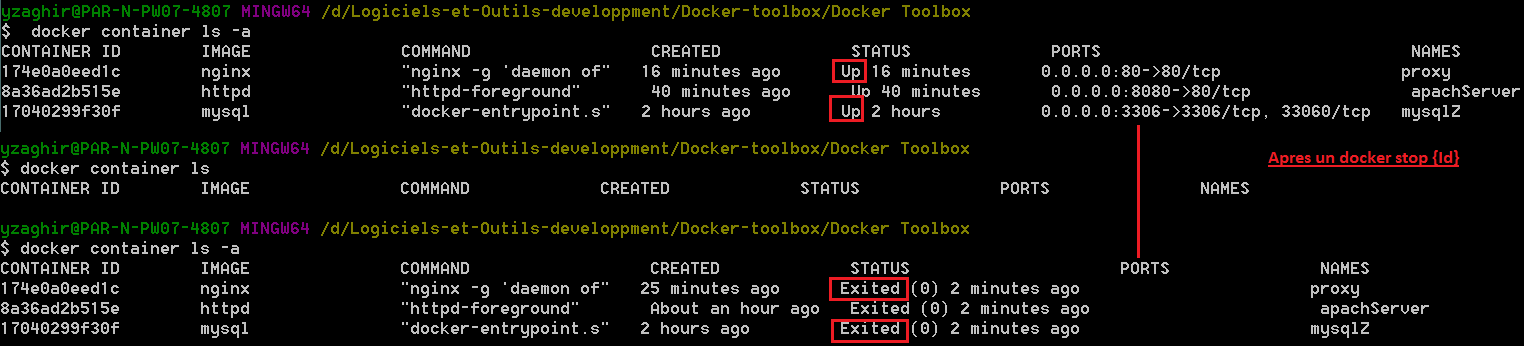
docker container ls afficher les container active

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

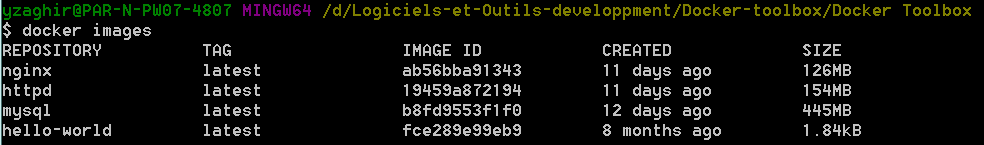
Docker container rm {idContainer}

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



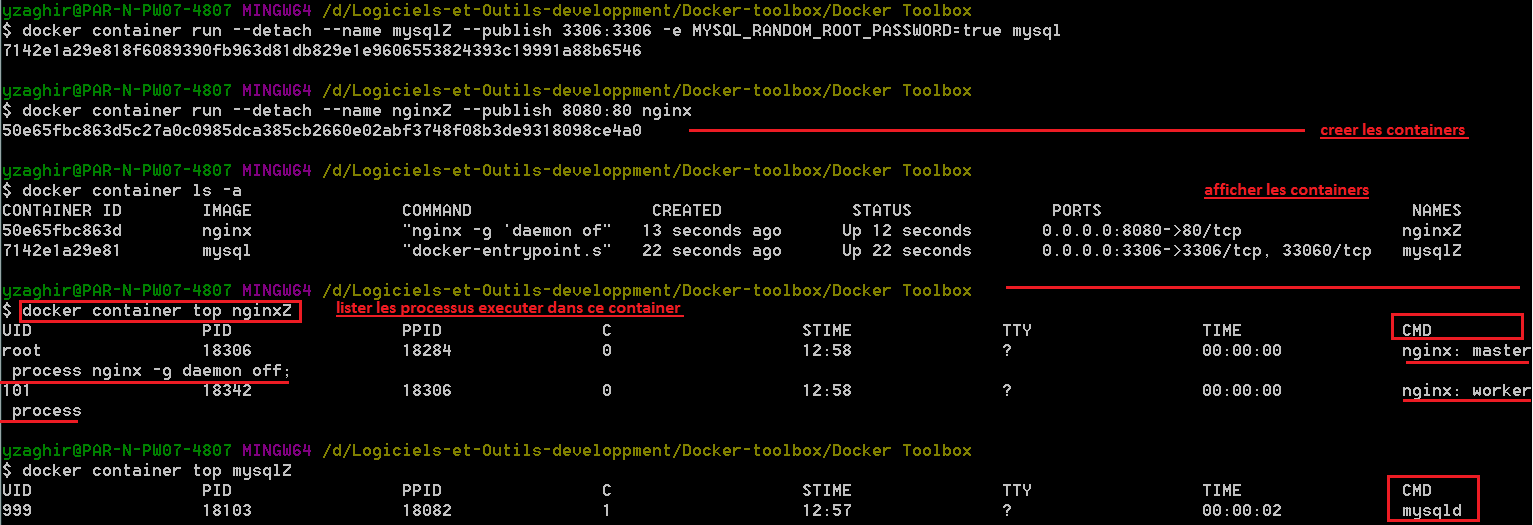
**Docker propose des outils de monitoring pour les containers**

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

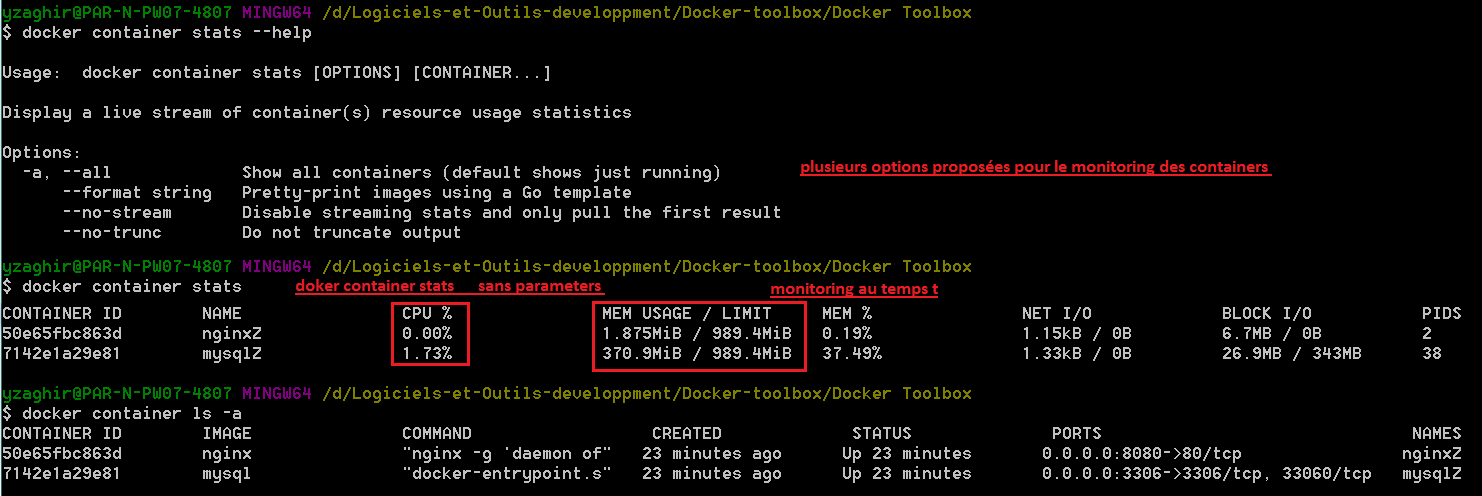


Pour afficher les metadata , la configuration, et comment le container est lancé , on utilise

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

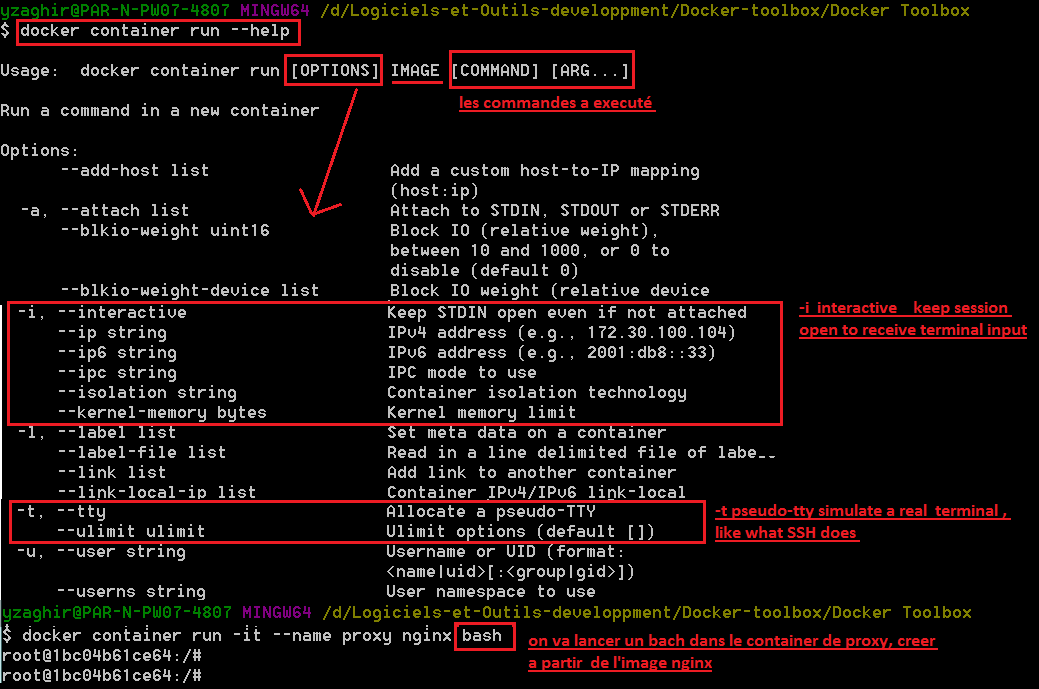
docker container run -it start new container Interactively

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

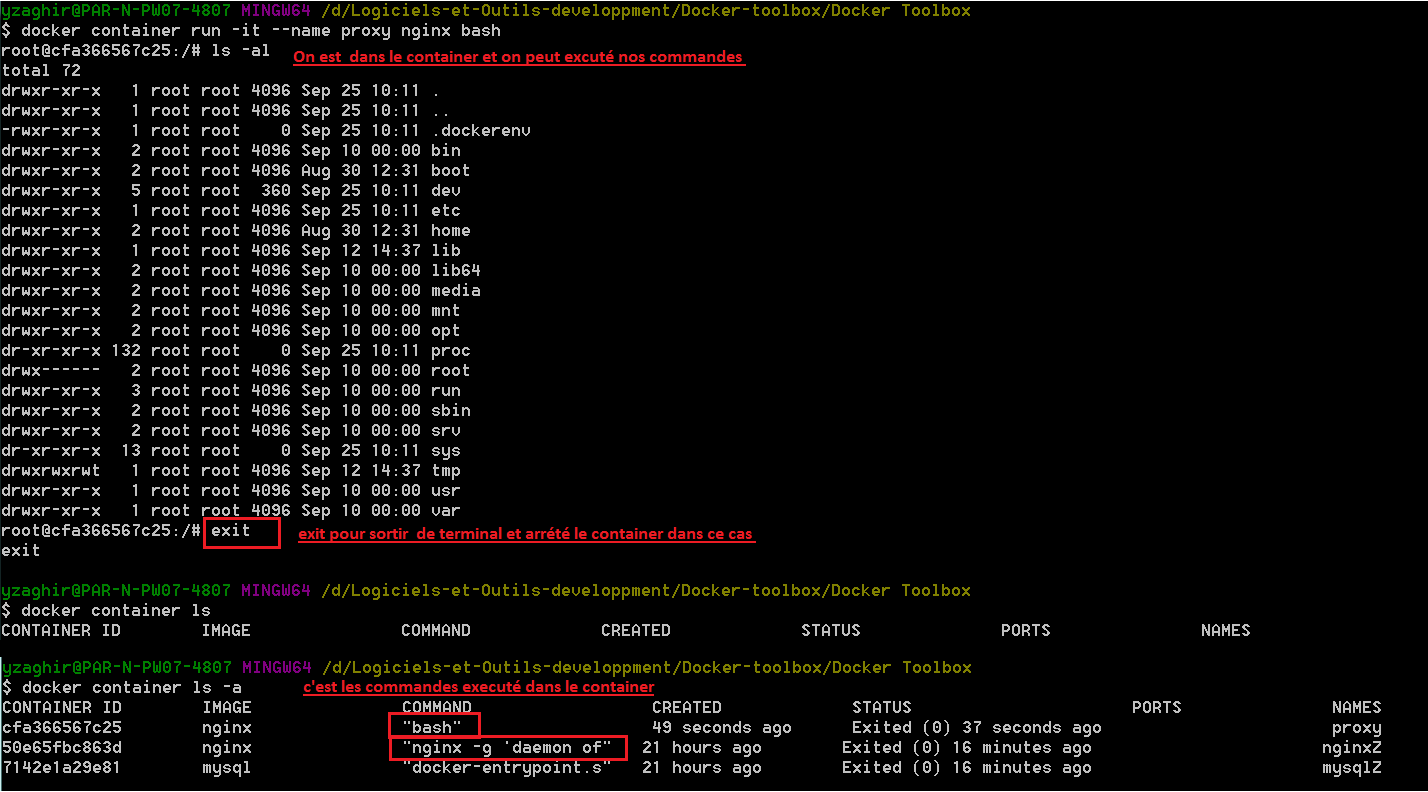
Voir la différence entre Linux distribution et un container

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

docker container run --help

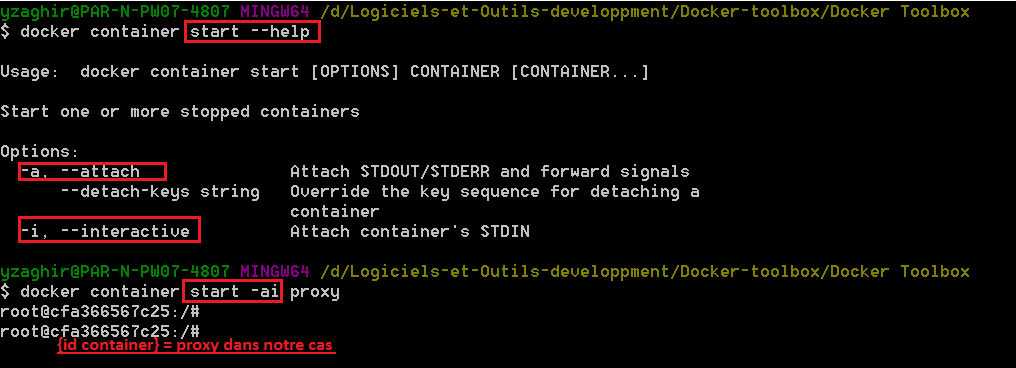


Bash shell : if run with it , it will give you a terminal inside the running container

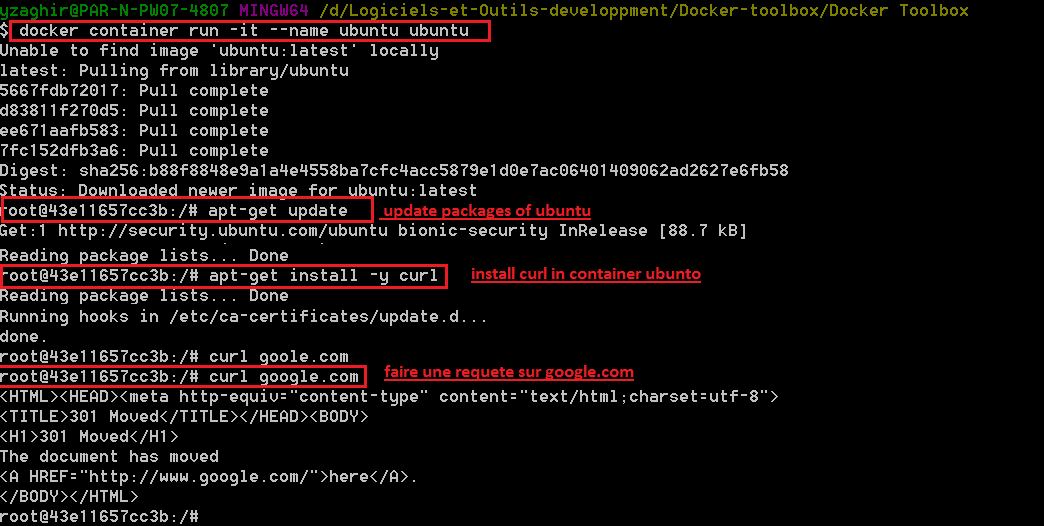


Pour démarrer le container on a plusieurs options , et puisque notre container se lance avec un bash on va utiliser le paramètre --ai pour le démarrer

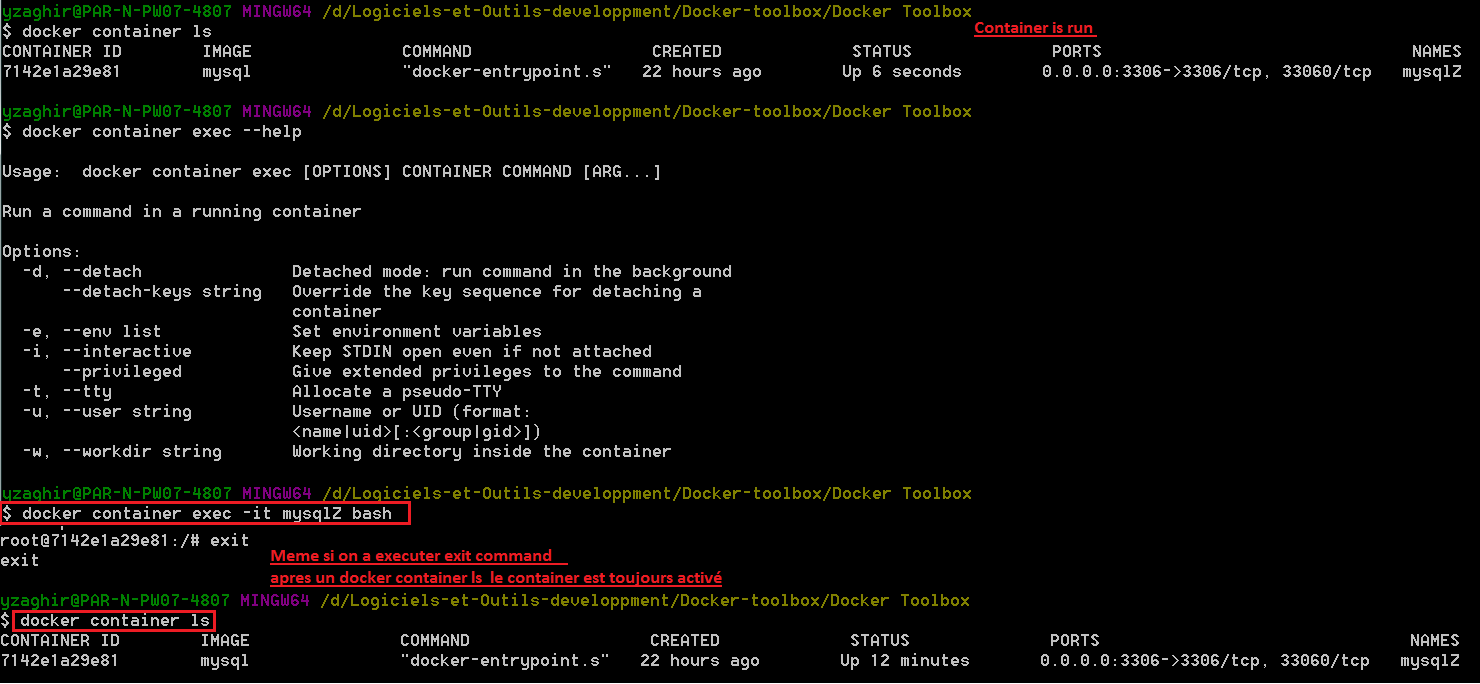
Docker container start - -ai {id ou name container }



Exemple



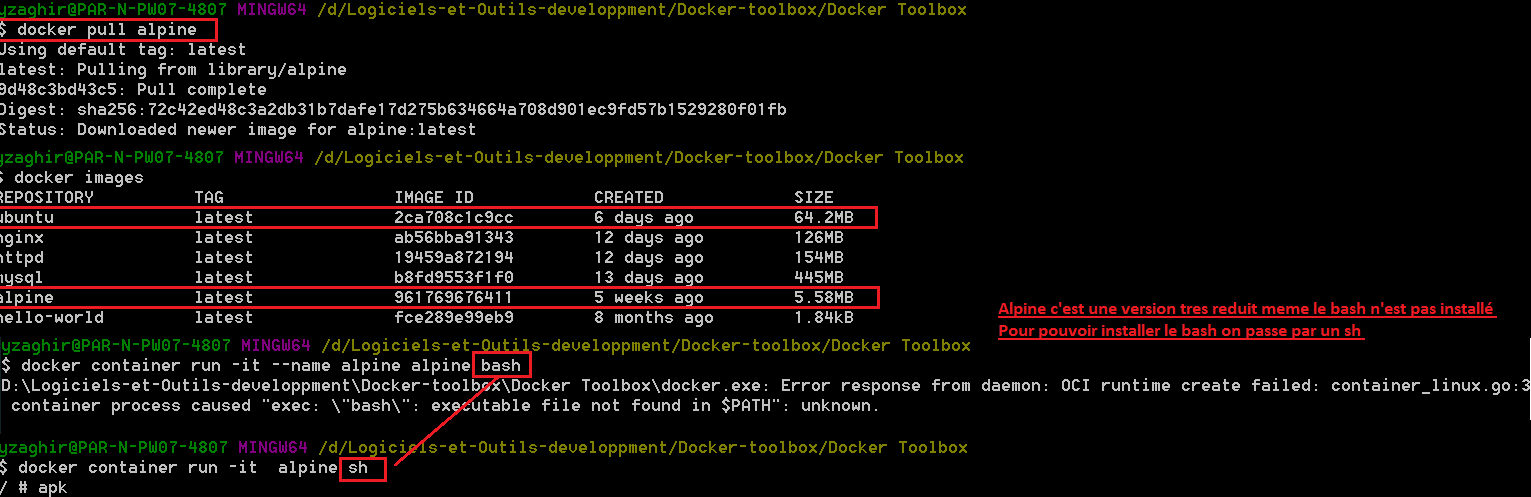
Docker container exec execute additionnelles taches dans un container démarré déjà

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**

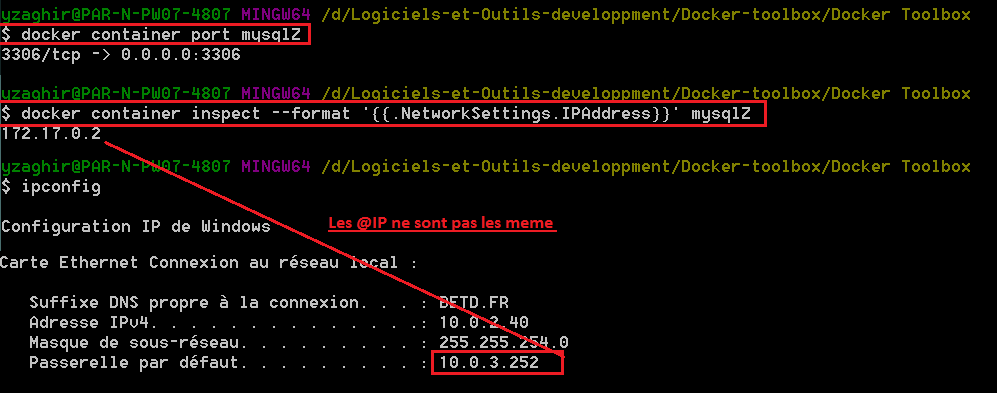
Docker propose des fonctions pour gérer la partie réseau

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

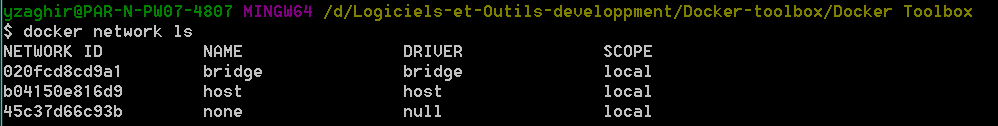
Inspect a network docker network inspect

Create a network docker network create –driver

Attach a network to container docker network connect

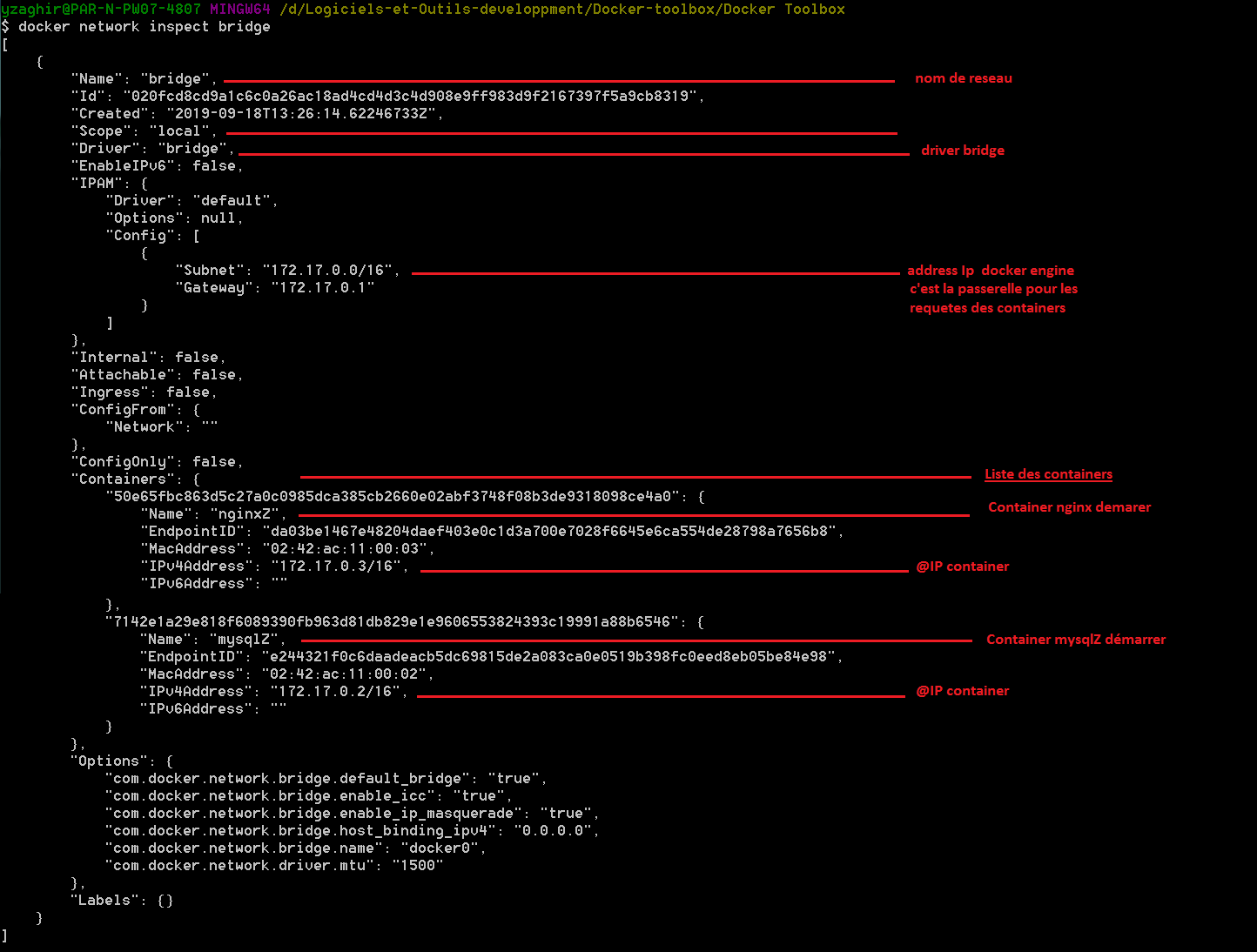
Detach a network from container docker network disconnect

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

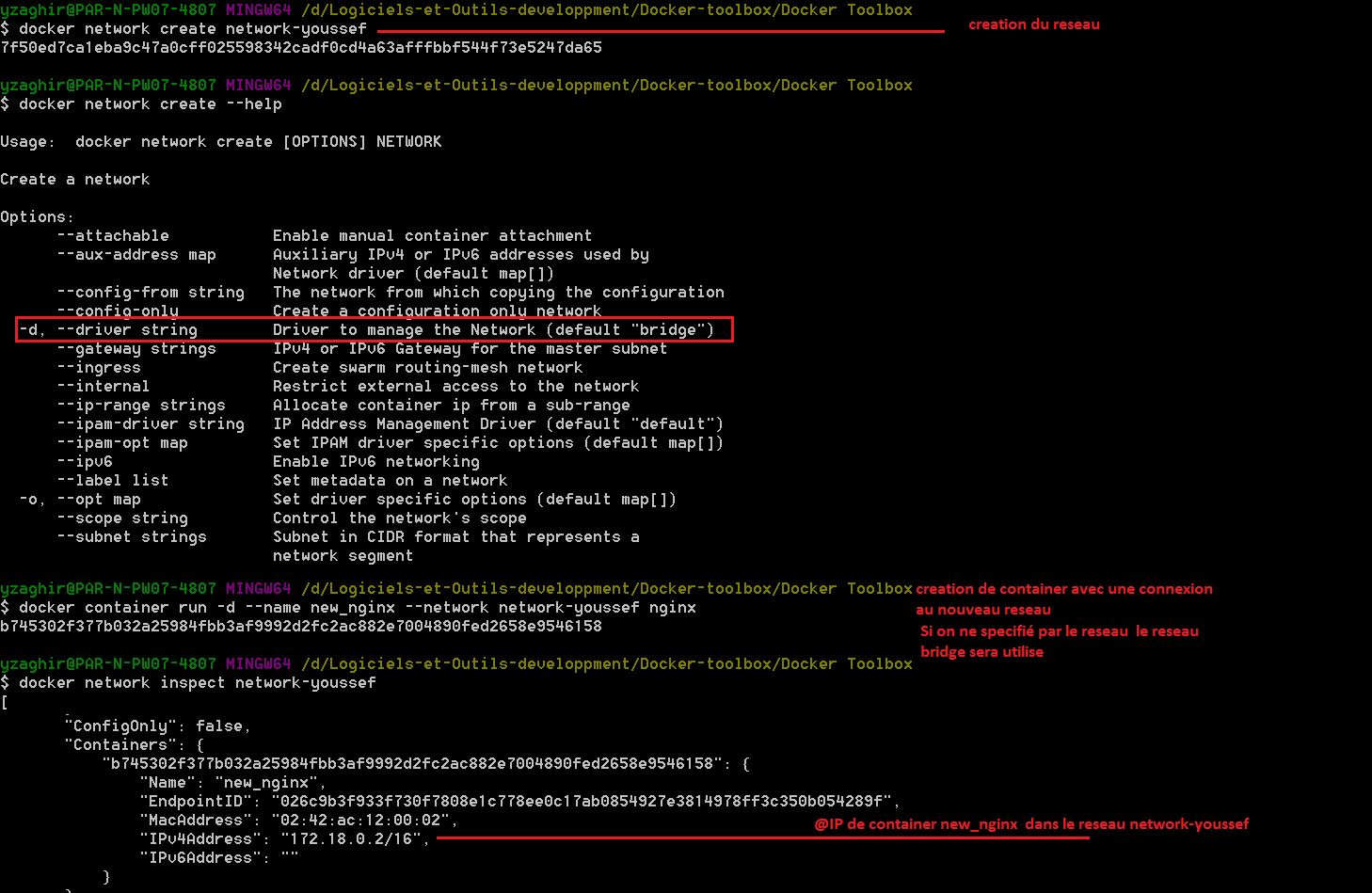


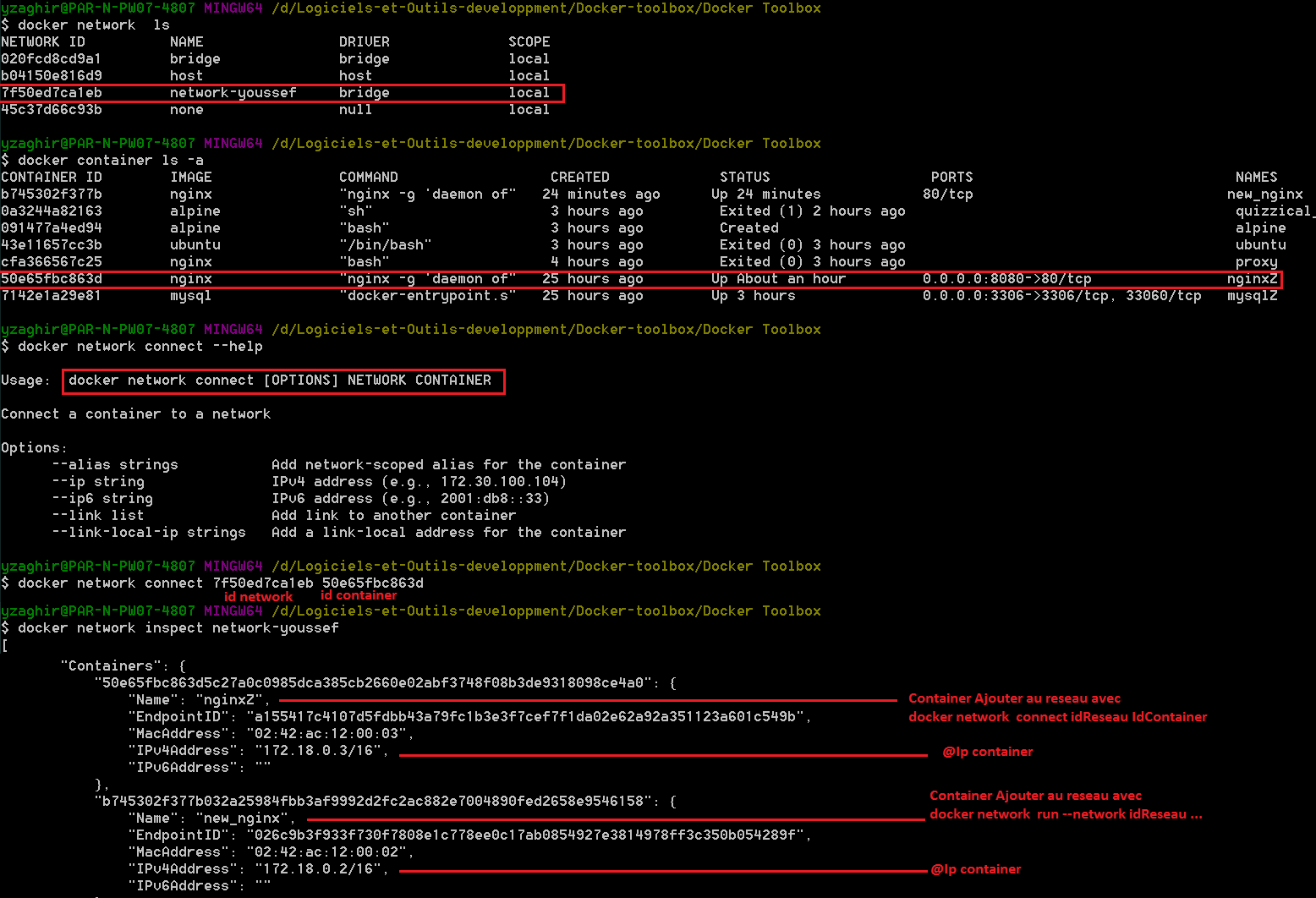
**Le réseau host** , it gain performance by skiggins virtual networks but sacrifices security of container model

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

docker network create network-youssef

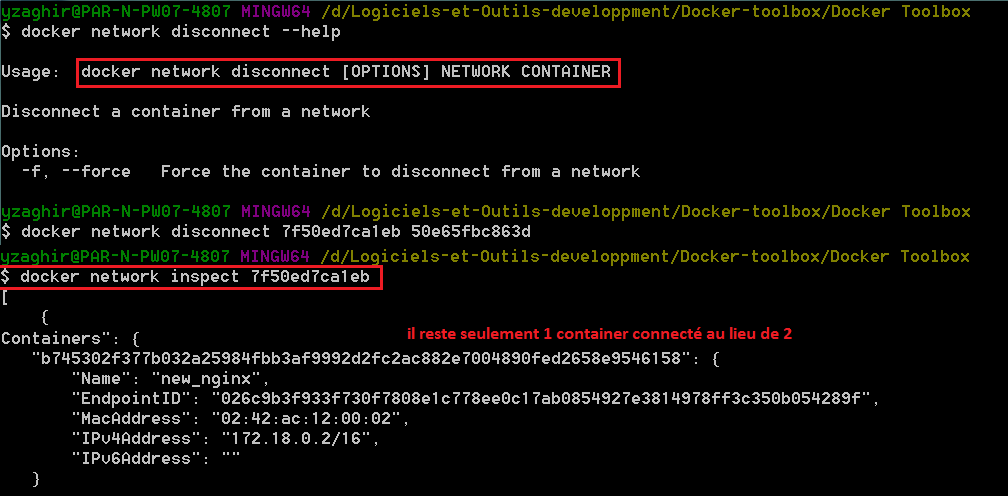
création d’un nouveau réseau, si on ne spécifie pas le réseau le driver bridge sera utilisé



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

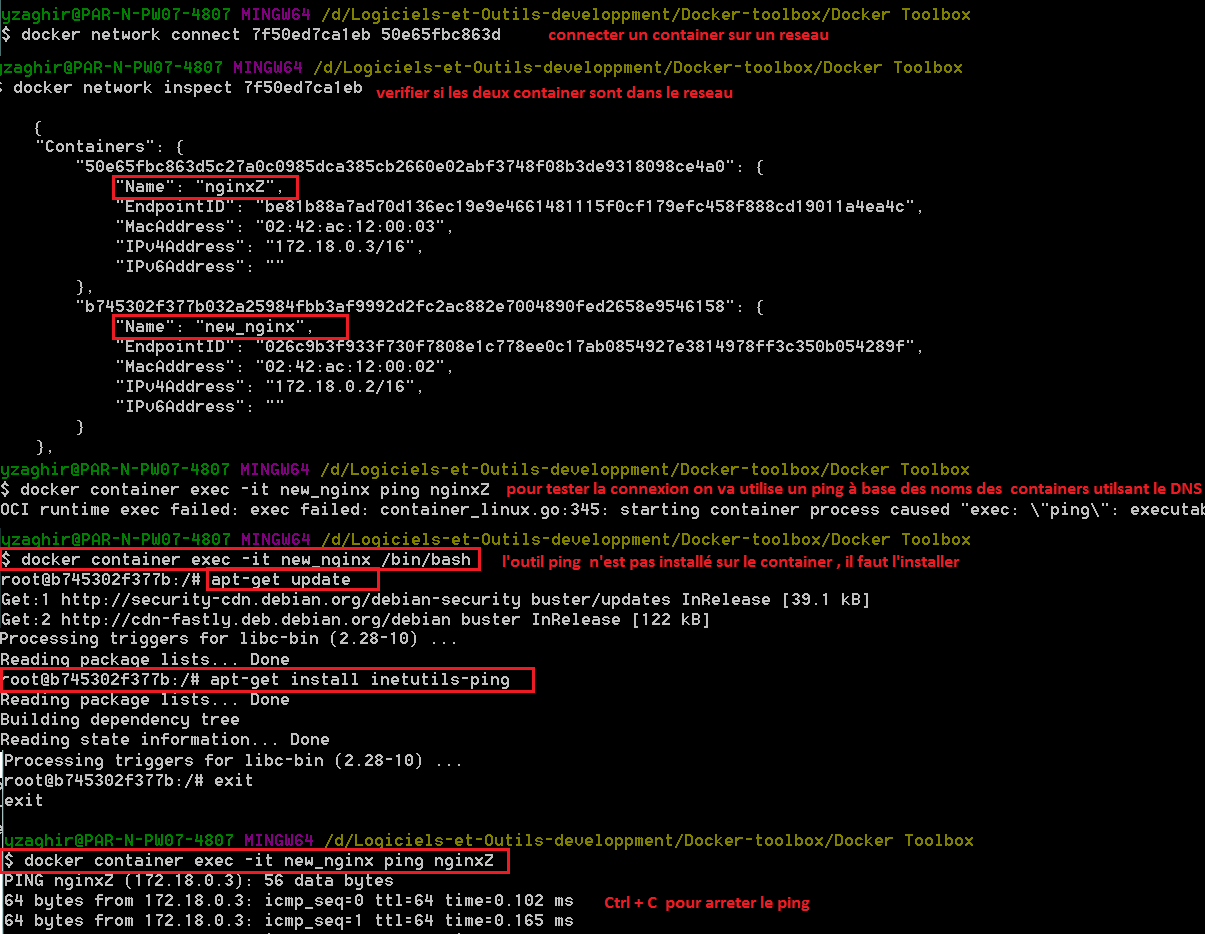
Static IP’s and using IP’ for talking to containers is an anti-patern .Do your best to avoid it.

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 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 image history nom-Image:version

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)

https://docs.docker.com/develop/develop-images/dockerfile\_best-practices/

A Docker image consists of read-only layers each of which represents a Dockerfile instruction. The layers are stacked and each one is a delta of the changes from the previous layer. Consider this Dockerfile:

**FROM ubuntu:18.04**

**COPY . /app**

**RUN make /app**

**CMD python /app/app.py**

Each instruction creates one layer:

FROM creates a layer from the ubuntu:18.04 Docker image.

COPY adds files from your Docker client’s current directory.

RUN builds your application with make.

CMD specifies what command to run within the container.