**DOCKER**

NOTES POUR MOI : wsl.exe –shutdown 🡪 Eteindre processus VMMEM qui bouffe la RAM

Docker est un programme avec un environnement isolé qui permet d’isoler des applications grâce à des containers.

Docker peut s’apparenter comme une Machine virtuelle, même si en réalité, cela ne l’est pas.

En effet, les VMs isole tout un OS disposant de ses propres ressources et donc son propre environnement. Du côté de Docker, on isole en fait la partie applicative, donc Docker utilise les ressources de la machine hôte, mais surtout, elles interagissent en temps réel avec les containers.

Les containers sont en fait une instance d’un OS, mais qui requière très peu de ressources comparées à une VM. On les démarre à partir d’image (image d’OS). Les images sont en quelques sortes des snapshots.

On va voir tout cela en application.

Je passe l’installation etc. On se retrouve directement dans PowerShell où l’on va pouvoir utiliser Docker directement comme une commande.

D’ailleurs, si on tape juste « Docker » dans le PowerShell, on obtient les informations des différentes commandes que l’on peut exécuter à la suite :

Une image contenant texte

Description générée automatiquement

Pour commencer à utiliser Docker, il nous faut donc des images.

On récupère nos images via la commande « Pull », par exemple, si on veut récupérer une image de Debian, voici la commande :



En voici donc le résultat :

Une image contenant texte

Description générée automatiquement

On peut voir que l’image s’est téléchargé, et d’ailleurs, il nous précise qu’il a utilisé les options par défaut et a donc pris la dernière version.

Si on veut vérifier qu’elle est bien présente, on exécute :

Une image contenant texte

Description générée automatiquement

On voit que l’image Debian est bien là.

Si on veut une version spécial, c’est tout à fait possible. Il faut juste aller consulter le site « hub.docker.com » pour voir quelles versions peuvent être installées.

Par exemple, on peut prendre celle-ci :

Une image contenant texte

Description générée automatiquement

Et donc, on a bien les 2 :

Une image contenant texte

Description générée automatiquement

Maintenant, moi je veux uniquement la stable, donc pour la supprimer, on utilise son ID :

Une image contenant texte

Description générée automatiquement

A présent, je veux démarrer mon Debian, donc via un container. On va donc utiliser la commande container, de la manière la plus simple possible pour le moment :

Docker run debian

Mais ça ne marche pas car c’est une image très simple, qui ne retourne rien si on ne lui précise rien au démarrage. Elle ne fait pas tourner de programme par défaut.

Il faut rentrer cette commande :

docker run --name debianTest1 -it debian /bin/bash

Il faut indiquer l’option “/bin/bash » pour lui dire d’exécuter Bash, car de base, il ne lance rien.

A partir de là, ma debian se lance en bash :



Exit pour la quitter. Maintenant, mon container existe :

Une image contenant texte

Description générée automatiquement

Si il apparait pas sans le -a, on le démarre avec la commande START :

Une image contenant texte

Description générée automatiquement

Mon container est maintenant actif, et si on veut rentrer dedans, donc dans Debian, on fait la commande exec :



Pour résumé :

Commande RUN : Crée un container via une image (soit précédemment téléchargé via pull, ou alors il la télécharge automatiquement si elle existe) et l’éxécute automatiquement

Commande START : Démarre un container « statique », donc non démarré.

Commande EXEC : Exécute le container ACTIF et « rentre dedans »

Commande STOP : Stoppe un container actif

--help : Après chaque commande, on peut faire ceci pour avoir les infos de commande.

AUTRES :

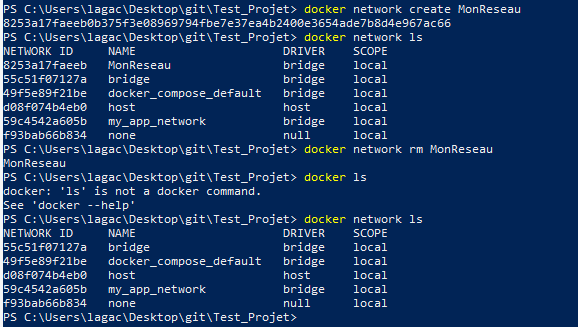
Docker network ls : Liste les réseaux

Docker network inspect bridge : Inspecte le reseau “Bridge” et retourne ses informations

**Créer un reseau manuellement :**

Docker network create NomDuNouveauReseau

A noter que pour **supprimer** un élément, c’est la même commande, donc rm, avec la même structure.



**Crée un conteneur en lui attribuant un un réseau :**

Docker container run -d –-name LeNomDuConteneur –network LeNomDeMonReseau LeNomDeMonImage



**PING D’UN CONTAINER A UN AUTRE :**

Une image contenant texte

Description générée automatiquement

**VISUAL STUDIO CODE, DOCKERFILE :**

Définition : Le Dockerfile est un fichier docker (sans extension d’ailleurs) qui permet de créer une image personnalisé avec différents paramètres. On va en fait récupérer une image, par exemple d’ubuntu, et pouvoir ajouter directement des commandes comme des apt update pour que l’on ait pas à le refaire quand on la monte dans un container.

Exécuter un code : docker build -t identifiant/node:10-alpine .

-t : pour nommer. Ici, le répertoire va être identifiant/node. Et le nom de l’image est 10-alpine

. à la fin pour exécuter le script qui se trouve à l’intérieur du dossier où l’on se trouve actuellement.

**EXEMPLE DOCKERFILE :**

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

Description générée automatiquement

Ici, on prépare une image d’alpine, qui va venir créer un répertoire DATA, puis se placer dedans.

On la build avec la commande montrer en-dessous. Avec une petite erreur dans un premier temps où j’ai oublié de mettre le répertoire 😉

On retrouve dans l’image crée comme ceci :

Une image contenant texte

Description générée automatiquement

Ensuite, on monte l’image. Donc on crée le container avec l’image précédemment crée. Ici je rajoute le fait de monter un volume dans le container.

Donc le volume « VolTest » va être crée et le container sera stocker sur ce volume, dans le répertoire /data. Voici le retour.



Donc on voit que le container s’est bien crée, et qu’il s’est bien placé dans le dossier data, comme on l’a renseigné dans le Dockerfile pour la création de l’image.

Concernant mon volume, il s’est bien crée :

Une image contenant texte

Description générée automatiquement

Maintenant, on va vérifier que le volume est bien config. On va créer un document texte dans le dossier data :

Une image contenant texte

Description générée automatiquement

Puis effacer le container « Contain1 » et en recréer un autre au même emplacement pour voir si le fichier texte existe toujours

Et c’est bien le cas !!

Une image contenant texte, capture d’écran, écran

Description générée automatiquement

**DOCKER-COMPOSE :**

Définition : Permet de déployer applications + autres viariables sous docker avec plusieurs conteneurs à partir d’image préparée. Ce fichier a l’extension **.yml**

Exemple pour Ubuntu :

Installation d’un serveur Ubuntu dans docker avec implémentation de paramètres :

Une image contenant texte

Description générée automatiquement

Au début du docker-compose, on renseigne la version à utiliser. Elle va être en corrélation avec la version de notre moteur Docker installé. Par défaut, si elle n’est pas renseignée, le fichier se met en 1.0.

Ici, on va récupérer l’image d’Ubuntu, la dernière, sur le hub docker. Donc l’image est déjà prête, donc la config système de l’image est sautée.

Dans le ficher docker-compose, on va donc lui renseigner de récupérer l’image « ubuntu », et de bosser à partir de celle-ci.

On peut donc commencer à rajouter des variables et des paramètres personnalisés pour la création du futur container.

Je mets comme nom au container « UbuntuTest01 » avec le mode de réseau « Host » qui peut me permettre de communiquer avec la machine hôte.

Je crée ensuite des volumes « movies » « config » et « tv » dans /path/to/..

Et j’expose le port 34200 en tcp (ici, il y a pas d’intérêt, c’est juste pour un exemple).

Je sauvegarde mon .yml et exécute la commande que l’on voit dans le terminal.

Le container se crée normalement et si on fait un « inspect », on peut vérifier ses paramètres :

Une image contenant texte

Description générée automatiquement

On retrouve ici nos volumes :

Une image contenant texte

Description générée automatiquement

Et le mode host avec le port 34200 en tcp sur ma machine host ET sur le container.

Une image contenant texte

Description générée automatiquement

En revanche, quand je lance ce docker compose, ma machine se config et s’éteint directement car il n’ a pas de processus qui s’exécute. Je ne l’ai pas renseigné. J’ai essayé de faire en sorte que cela s’affiche mais impossible… J’ai essayé à la fois sur le docker-compose.yml mais également à partir d’un dockerfile. Et donc mon docker-compose qui récupère l’image crée avec le dockerfile. Voyez par vous-même :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

J’ai modifé, re modifié et encore modifier en supprimant, en mettant mes commandes en commentaire car ça ne marchait pas etc.

Mais impossible de trouver comment faire.

J’arrive à faire des petites choses comme afficher un texte comme ceci :

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

Description générée automatiquement

Quand je build l’image, les commandes RUN passent bien également : Une image contenant texte

Description générée automatiquement

Mais si je veux lancer un ping, ça ne fonctionne pas non plus :

Une image contenant texte

Description générée automatiquement

**Container DOCKER-CRAFT :**

Affichage des containers sur un Minecraft. Possibilité d’en créer etc.

Une image contenant texte, capture d’écran, écran

Description générée automatiquement

Bien actif mais je peux pas me co car j’ai une version Minecraft cracké.. ☹

Dommage, je peux pas montrer visuellement mais j’ai vu avec mon groupe en partage d’écran ce que ça donne.