DOCKER

Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc. (wikipedia)

A) INSTALLATION

FORTEMENT DECONSEILLE D'UTILISER DOCKER SUR WINDOWS EN PRODUCTION

B) CONCEPTS

2 CONCEPTS: CONTAINERS et IMAGES

conteneur créé à partie d'une image
pour faire une relation avec le monde objet
une image est comme un classe et un container un objet

IMAGES conservées dans un registre pour servir de modele pour créer des conteneurs.
dans un registre une image est placée dans un depot
plusieurs versions de cette image dans le depot

DEPOT = projet gitlab et tag etiquette

IMAGE modele de conteneur
image figée et modele vivant

IMAGES => calques (like photoshop)

1) decor d'arriere plan
2) arbre
3) personnage

Modification successive des élément dans l'image (motif config , ajout lib , suppression…etc…)

C) DOCKERHUB

dockerhub

1) image officiel
2) nombre de téléchargement + note
3) faire tout soit même

D) EXECUTION

bash

docker run --rm bash echo salut

Unable to find image 'bash:latest' locally
docker cherche l'image localement
latest: Pulling from library/bash
tire l'imame depuis le docketrhub
f84cab65f19f: Pull complete

9f33ec3cbeb8: Pull complete

e664623ea448: Pull complete

Digest: sha256:ad5e40db3e26bed0d627e049f83e76988ea7d93133e03ea0649fbce51d161272

Status: Downloaded newer image for bash:latest
téléchargement dans le cache de notre serveur -> instanciaitoin de l'image et exécution
salut

REVENONS DUR COMMANDE

docker : commande qui donne des ordres à docker
run action à executer => créer et instance un conteneur
—rm supprime une fois l'action exécutée le conteneur (l'image reste sur l'ordinateur)
bash => nom de l'image
echo salut (commande à executer dans le conteneur

RELANçONS

docker run --rm bash echo salut

Maintenant :

docker images

REPOSITORY (depot de l'image)	TAG(version)	IMAGE ID(id unique)	CREATED(date creation	SIZE

Allons sur docker.hub.com
recherche bash
tag

on voit les versions

taper

docker run --rm bash:3.2 echo salut

Erreur…

il faut effectuer un

docker login

docker images

docker run —rm bash

rien

docker run —rm bash -ti —rm bash (lance -ti lance en interactif) entree-standard - sortie standard (bash prend prend en entree standard le clavier et en sortie le tty (écran)

uname -n
exit
uname -n

bash-4.4.#
uname -n (affiche le hostname)

2nd terminal

docker run —rm bash -ti —rm bash

autre terminal

docker ps

liste les conteneurs crées

CONTAINER ID (id unique)
IMAGE (nom de l'image)
commande (permet de faire fonctionner le serveur) +tard
CREATED creation de conteneur
STATUS
PORT +tard
NAME (on peut donner un nom ou docker invente)

exit sur autre term puis ps sur second

detail

docker images
docker history ID image

pas obliger de saisir l'intégralité de l'id

(completion possible)

PERSISTENCE DES DONNEES

docker run ti —rm bash
echo TOTO > fichier
cat fichier

ls -l

exit
docker run ti —rm bash
cat fichier

(—rm)

relancent sans —rm

cat fichier (pas mieux)

docker ps
pas de conteneur , à quoi sert -rm

docker —help
docker ps -a
le conteneur est bien présent

docker run CREE TOUJOUR UN NOUVEAU CONTENEUR

docker tabtab

docker start help

docker start -ai nomduconteneur

cat fichier

SI REDEMARRAGE les conteneur lancés sans —rm seront redémarrables

COMMENT persister ?

LES VOLUMES

pour persister les données il faut monter des volumes du système home sur docker , ces volumes ne sont pas supprimées lors de la destruction du conteneur

echo test > fic
docker run -ti —rm -v $(pwd)/fic:/texte bash

$(pwd)/fic emplacement sur la partie de gauche
texte point de montage sur docker

docker -ti —rm -v //:texte bash (pas conseillé)

CONTENEUR COMMUNICANTS

par défaut local au conteneur (pas de communication avec l(hâte)

docker run —rm nginx

a ouvert un port 80 par défaut local au conteneur

navigateur -> taper localhost => no trouvé (sauf si apache sur votre hôte)

comment accéder depuis l'exterieur

docker run —rm -p 80:80 ngnix (pas obliger d'avoir même port)

essayer sur vitre machine hôte http://localhost

Pour offrir un autre port à l'hôte (serveur qui execute docker)

docker run —rm -p 8080:80 nginx

essayer

RESEAU

Numéro 1
pilote réseau "none"
docker run -ti —rm —network none bash

(une seule interface)

Numéro 2
pilote de réseau bridge
dans terminal1,2 ,3
docker run -ti —-rm bash (bridge par défaut)

chaque conteneur a son adresse ip sur le même réseau
ils se voient entre eux.

ifconfig
ping ip

Isolation de differents bridges

cloner un bridge

docker network create —driver=bridge mon_reseau

on obtient un id de réseau (objet docker qu'on peut manipuler)

docker network —help
docker network ls
docker network rm <id ou nom>

Création de 2 conteneur sur notre bridge

docker run -ti —rm —network=mon_reseau —name srv1 bash
docker run -ti —rm —network=mon_reseau —name srv2 bash
docker gere un DNS interne.
ping srv1 srv2

Sinon on peut faire

docker run -ti —rm —name srv1 bash
docker run -ti —rm —name srv2 bash
plus de dns

maintenant

docker network connect mon_reseau srv1
docker network connect mon_reseau srv2

ping srv2

menage

si plus de conteneur rattaché on peut supprimer un réseau

docker network rm mon_reseau

docker network ls

3) pilote réseau host

permet de partager la même pile réseau que la machine hôte.

auparavant on mappait le port de l'hôte avec le port du conteneur
exemple vu 8080:80
maintenant avec host

ouvrir un port sur le conteneur l'ouvre sur l'hôte

docker run —rm —network=host ngnix

localhost (sans mappage)

même port 80

très peu utilisé

4) overlay (utilisé pour des conteneurs s'executant sur différents serveurs).
voir serveur Swarn

Volumes managés

docker volume create mes_datas
cree le volume qque part
docker volume ls

docker —rm -ti -v mes_datas bash:/src bash
echo coucou > fichier
exit

docker run —rm -ti -v mes_datas bash:/src bash
on retrouve le volume

on a juste le nom du volume au lieu du chemin courant

docker volume inspect mes_datas

pas beaucoup de différence par rapport au volume mappé

DOCKER COMPOSE

on va monter un environnement multi-conteneur chacun ayant son rôle

ce sont des **microservices** (chaque conteneur offre un service unique)

exemple avec wordpress: Serveur Web + Base de données

2 conteneurs doivent se voir

Créer docker-compose.yml

YAML (voir YAML)

```
version: "3"
services:
        wordpress:
                image: wordpress
                ports:
                        - 80:80
        db:
                image: mysql:5.7

volumes:
networks:
```

mysql docker hub aller à la section variables d'environnement lors de la construction du conteneur

4 variables à la construction du conteneur
MYSQL_ROOT_PASSWORD
        mdp administrateur du moteur mysql (permet de créer les données)
MYSQLDATABASE
        nom de la database à crée
MYSQL_USER
        nom utilisateur de la base de données
MYSQL_PASSWORD
        mot de passe utilisateur

```
version: "3"
services:
	wordpress:
		image: wordpress
		ports:
			- 80:80
		environment:
			- WORDPRESS_DB_HOST=db
			- WORDPRESS_DB_USER=joni
			- WORDPRESS_DB_PASSWORD=bigoud
			- WORDPRESS_DB_NAME=wordp
	db:
		image: mysql:5.7
		environment:
			- MYSQL_ROOT_PASSWORD=kerinou
			- MYSQL_DATABASE=wordp
			- MYSQL_USER=joni
			- MYSQL_PASSWORD=bigoud


volumes:
networks:
	galaxie:
```

Puis wordpress dans dockerhub voir les Variables d'environnement et ajouter à wordpress

docker-compose up

pour charger les conteneurs

docker-compose down pour arrêter et décharger les conteneurs

VOLUMES MAPPES

```
version: "3"
services:
  wordpress:
    image: wordpress
    ports:
      - 80:80
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=joni
      - WORDPRESS_DB_PASSWORD=bigoud
      - WORDPRESS_DB_NAME=wordp
    networks:
      - galaxie
    volumes:
      - ./data/wp:/var/www/html
  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=kerinou
      - MYSQL_DATABASE=wordp
```

```
    - MYSQL_USER=joni
    - MYSQL_PASSWORD=bigoud
  networks:
    - galaxie
  volumes:
    - ./data/db:/var/lib/mysql
#volumes:
networks:
  galaxie:
```

fichier wordpress et mysql conservés dans un répertoire local de host (data) dans lequel il y a les données mysql et les données wordpress

supprimons maintenant data et voyons la gestion par les volumes managés

VOLUMES MANAGES

on créée dans volumes

```
        wp:
        db:
```

ainsi

```
version: "3"
services:
  wordpress:
    image: wordpress
    ports:
      - 80:80
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=joni
      - WORDPRESS_DB_PASSWORD=bigoud
      - WORDPRESS_DB_NAME=wordp
    networks:
      - galaxie
    volumes:
      - wp:/var/www/html
  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=kerinou
      - MYSQL_DATABASE=wordp
      - MYSQL_USER=joni
      - MYSQL_PASSWORD=bigoud
    networks:
      - galaxie
    volumes:
      - db:/var/lib/mysql
```

```
volumes:
  db:
  wp:
networks:
  galaxie:
```

je relance

docker-compose up

on est revenu sur une installation

je fais une installation

je stoppe

docker-compose down

puis

docker volume ls

j'ai mes volumes

on a une combinaison du nom du répertoire et le volume

évite les collisions entre plusieurs environnements

docker-compose down -v supprime tous les volumes ATTENTION PAS DE CONFIRMATION

CREATION IMAGE SIMPLE

créer une application (exemple app node) et la tester

exemple d'application node

```
const express = require('express')
const app = express()
app.get('/',function(req,res) {
     res.send("slash");
})
app.get('/cool',function(req,res) {
     res.send("cool");
})
app.listen(8080, () => {
  console.log("Serveur prêt")
})
```

créer un Dockerfile

```
FROM node:12.4
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD [ "node", "app.js" ]
```

```
docker build -t unnom .
```

```
docker run -p 8090:8080 unnom
```

tester localhost:8090

ASSICIATION DE L’IMAGE CREE AVEC mongo

Créer un docker-compose.yml ainsi

```yaml
version: "3.8"
services:
  api:
      build: .
      ports:
        - 8080:8080
      links:
        - mongodb
      depends_on:
        - mongodb
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    expose:
      - "27017-27019"
    volumes:
      - ./data:/data/db
```

executer:

```
Supprimer tous les containers

docker system prune -a
```

Autre exemple

Créer une image à patrie d'un container existant

Installer ubuntu

```
docker run –ti ubuntu
```

ubuntu vient "à poil", c'est un linux de base sans web server sans node sans npm

installons des modules linux

apt-get update

```
apt–get install nginx
apt–get install systemctl

apt–get install vim
ou
apt–get install nano

systemctl status nginx
systemctl start nginx
```

Sur un autre terminal

```
docker ps –a

docker commit CONTAINER_ID IMAGE_NAME
```

Une nouvelle image est créé avec les options installées

instancier cette image

```
docker run –p 8080:80 IMAGE_NAME
```

ça donne sur localhost:8080 -> ?

Modification de index.html par

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome Jam Image</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx on openlab docker!</h1>
<p>
Application dockerisée de Jam.
<p>
<img src="jam.png" width=600>
<p><em> &copy; Openlab for dwwm AFPA 2021 </em></p>
</body>
</htm>
```

Copie d'un image à partir du host dans le container docker .

`docker cp path/de/image.png ID_CONTAINER:path/du/container`
`(exemple ici /var/www/html)` racine des sites nginx