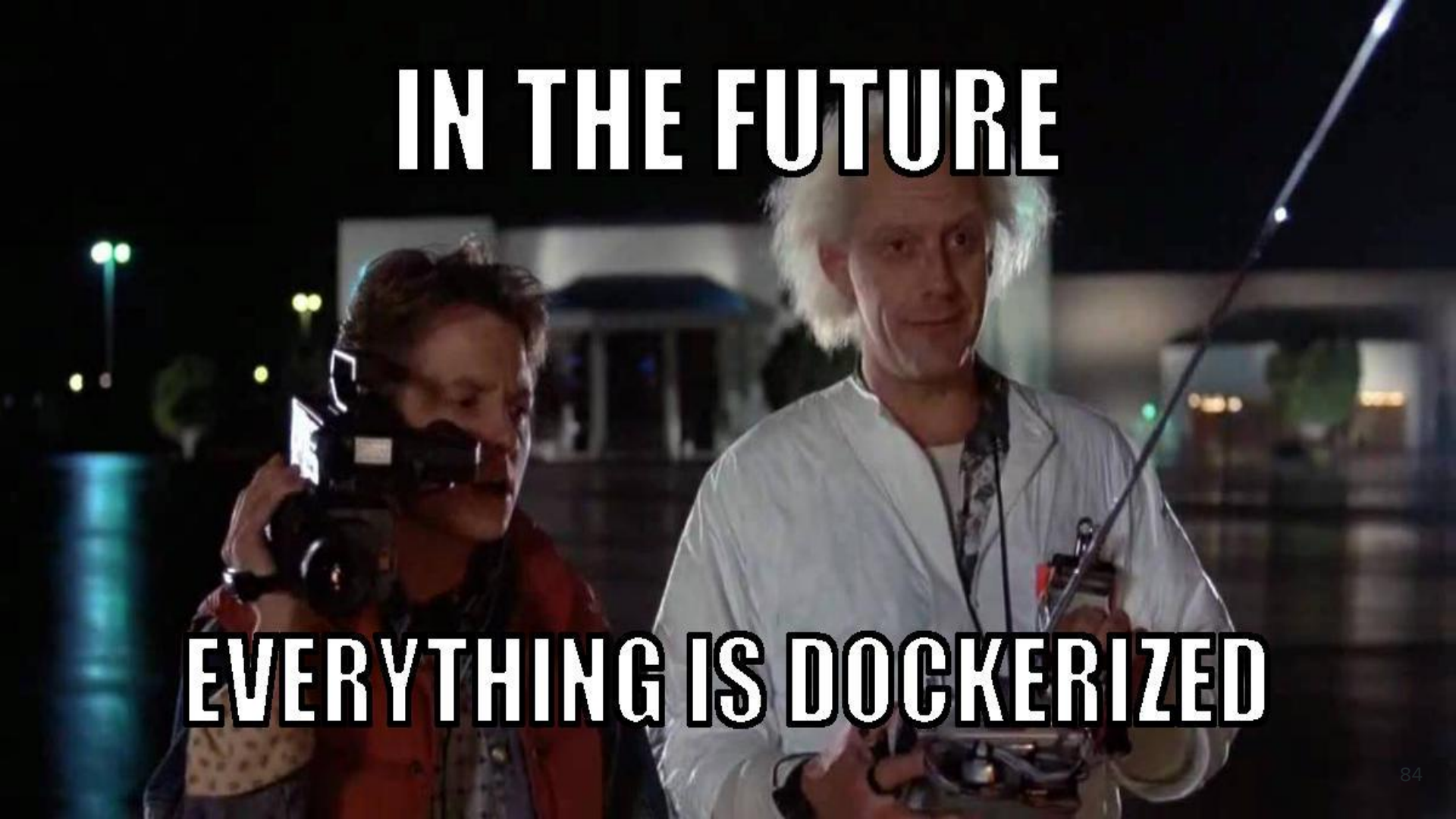
Environnement de travail collaboratif

ESGI - Semaine du 04/11/19

Christophe Villeger
Vincent Monjaret
Lucas Moisan


IN THE FUTURE
EVERYTHING IS DOCKERIZED

Installation

- Linux
  - Docker CE
- Mac
  - Docker for mac
- Windows 10 PRO
  - Docker for windows
- Windows 10 familial et en dessous
  - Docker Toolbox

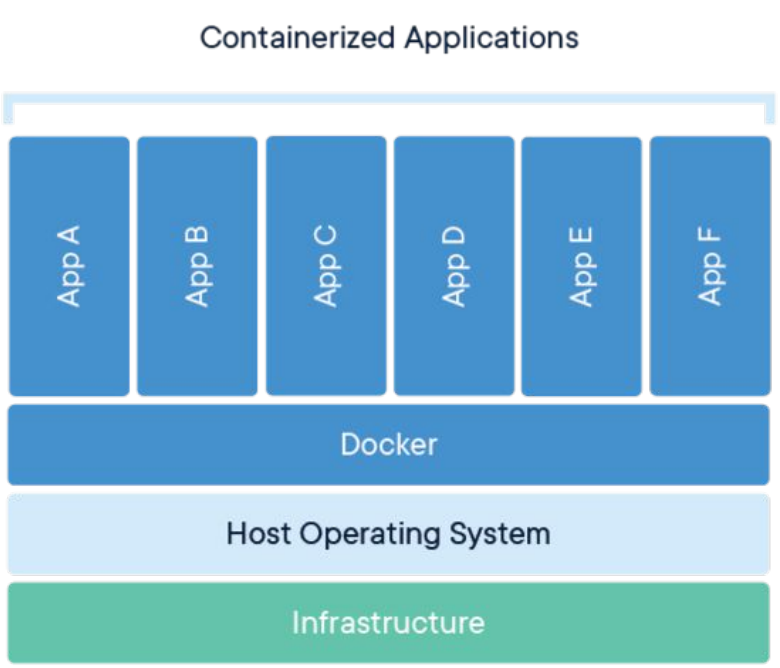
Docker : qu’est ce que c’est?

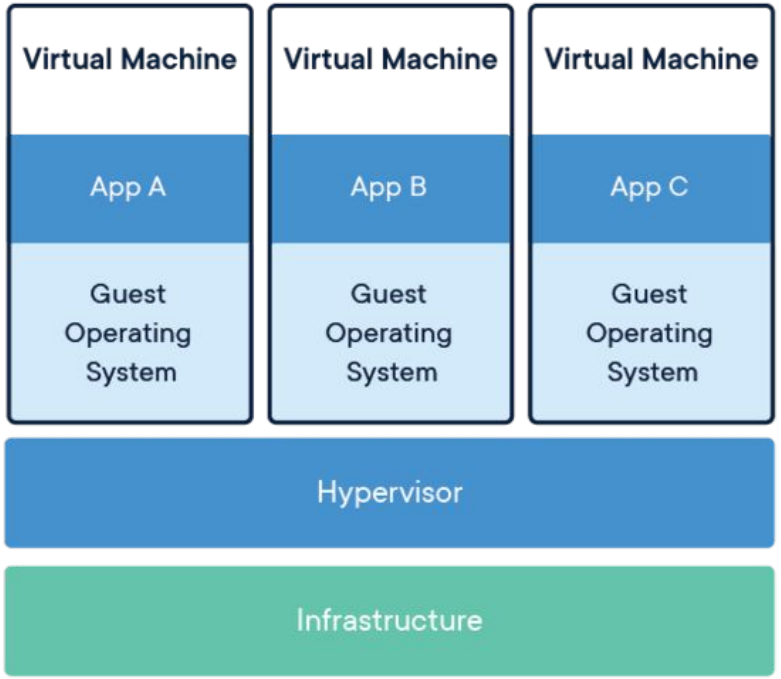
- Docker est un outil conçu pour faciliter la création, le déploiement et l'exécution d'applications à l'aide de conteneurs.
- Se rapproche de la virtualisation
  - A une différence...

Docker : Avantages multiple

- Les containers sont beaucoup plus légers
  - Pas besoin d'installer d'OS
  - Il est possible de mettre beaucoup plus de containers que de VMs par serveur
- Moins de CPU, RAM et disque utilisés
- Extrême portabilité

Docker : Container VS Virtual Machine





Docker : lexique

- Images
  - Modèle en lecture seule qui permet de créer des containers
  - Créés par vous ou par d'autres utilisateurs de docker
  - Stockées dans un registry ou localement sur votre machine
- Containers
  - "Entité" d'une application qui se suffit à elle-même
  - Contient tout ce qui est nécessaire pour faire tourner l'application
- Registry
  - Permet de stocker et distribuer des images
  - Docker Hub / Gitlab / Github

Docker commands

docker pull

Télécharge les images du docker hub ou d’un registre privé

- Est exécuté lorsque vous lancez docker run et que l’image n’est pas en local
- Par défaut, le tag latest est utilisé. Il est possible de spécifier le tag en rajoutant **:tag** (ex. ubuntu:18.04)
- Par défaut, télécharge l’image depuis docker hub

docker images

Liste et donne des détails sur les images

- Affichage LIFO
- Indique le repository, tag et taille de l’image
- Les images comportent des couches intermédiaires qui permettent la réutilisation rapide avec docker build, toutes les étapes sont mises en cache (mais n’apparaissent pas par défaut)
  - utiliser l’option -a pour tout afficher
- La taille indiquée est la taille de l’image + les images parents (pas forcément affichées)
- La même image est affichée autant de fois qu’elle a de tags

docker run

Créer un container depuis une image

docker run [options] image[:tag] [command] [arg1, arg2, ...]

ex : docker run -ti ubuntu /bin/bash

- **-d** : lancer en mode détacher
- **-v** : partager un dossier / fichier entre l'hôte et le container
- **-p** : créer un tunnel d'un port de l'hôte vers un port du container
- **-ti** : autoriser la simulation d'un terminal et la saisie dans celui-ci
- **--rm** : supprimer le container à son arrêt
- **--name** : Nommer le container

docker ps

Liste et détaille les containers (par défaut ceux lancés)

- **-a** : liste tous les containers (même arrêtés)
- **-q** : retourne seulement l'id
  - sert pour concaténer le résultat à d'autres commandes linux : *docker stop $(docker ps -q)*

docker exec

Lance une commande dans un container lancé

Attention : ne redéfinit pas la commande par défaut

docker exec -ti <container> /bin/sh

Lance un processus de shell et le retourne, et permet ainsi de se connecter en terminal au container

docker login / docker logout

Pour pouvoir travailler avec le hub (répertoire public d’images), vous devez créer un compte sur hub.docker.com

- L’authentification est obligatoire pour les repos privées

docker commit

Cette commande permet de créer une nouvelle image à partir de modifications effectuées depuis un shell interactif depuis une autre image

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

- ➢ Une des manières de créer une image (l'autre sera en utilisant un Dockerfile)
- ➢ Vous pouvez fournir le nom de la nouvelle image et du repo après le nom/id du container à utiliser
- ➢ Pour publier sur le docker hub vous devrez le préfixer <username>/

docker push

Envoi (partage) une image de votre hôte vers le docker hub ou un repo privé

docker push [OPTIONS] NAME[:TAG]

- ➢ l’image doit respecter une certaine nomenclature de nom et tag

docker start / docker stop

- La commande docker start lance un ou plusieurs containers arrêtés

- La commande docker stop arrête un ou plusieurs containers lancés

docker rm

Supprime un ou plusieurs containers

- Peut être supprimé par l’id ou le nom du container
- Un container lancé ne peut pas être supprimé
  - Pour forcer la suppression, utiliser l’option -f

docker rmi

Supprime une ou plusieurs images

- Aucun container ne doit utiliser l’image (sinon erreur)
- Sinon passer le paramètre -f pour forcer
  - les containers utilisant l’image seront orphelins, mais pourront continuer à tourner ou à être démarrés

TP

- Dans un premier terminal (terminal 1)
  - récupérer l'image alpine
  - vérifier que l'image alpine est bien présente en local
  - démarrer un container avec un shell de manière interactive à partir de l'image alpine
  - lister les processus en cours (2 processus)
- Dans un nouveau terminal (terminal 2)
  - lister les containers lancés
  - ajouter un nouveau processus (ping www.google.com) au container (attention il vous faut le nom du container)
- Dans votre container (terminal 1)
  - lister de nouveau les processus en cours (3 processus)
  - créer un fichier avec votre nom

- Dans le terminal 2
  - Créer une nouvelle image à partir du container que vous avez modifié (vous y avez ajouté un fichier lors de la précédente étape) et nommer là alpine-dm (attention à votre nom d'utilisateur)
  - se log au docker hub
  - envoyer votre image sur le hub
  - vérifier sur hub.docker.com que votre image est présente
- Dans le terminal 1
  - arrêter et supprimer votre container
  - retirer l'image alpine et votre alpine-dm de la liste des images sur votre ordinateur
  - démarrer un nouveau container alpine-dm depuis l'image du hub

Création d’une image

Dockerfile

Docker peut créer des images automatiquement à partir d’instructions dans un fichier Dockerfile

- Permet de spécifier un ensemble de commandes (ex.: installation de packages)
- Docker build se sert de ce fichier pour automatiser le build de l’image

docker build

La commande docker build permet de créer une image à partir d'un fichier dockerfile

docker build -t <user-dockerhub>/<image-name>:tag /path/

- ➢ l'utilisateur du hub n'est pas obligatoire, vous pouvez seulement passer le nom de l'image
- ➢ le tag par défaut sera latest et n'est pas obligatoire

Attention, ne pas fournir un path dans lequel il y a beaucoup de fichiers et de dossiers car il va tous les parcourir (Pensez à faire un dossier différent pour chaque Dockerfile)

Best practices

Ce fichier d’instructions doit respecter des bonnes pratiques

- Être le plus éphémère possible
- N’installer que le nécessaire
- Un container = un processus
- Limiter le nombre de couches
- Trier les arguments multi-lignes par ordre alphanumérique

FROM

FROM est la première instruction d’un Dockerfile

Toutes les images dérivent d’une autre image ou d’une image de base (ex.: debian, alpine)

FROM alpine

FROM debian:wheezy

Instructions

Les 3 instructions RUN, CMD et ENTRYPOINT peuvent être définies selon une forme exec ou shell

Shell form

- RUN apt-get install php-fpm
- CMD echo "Hello world"
- ENTRYPOINT echo "Hello world"

Exec form

- RUN ["apt-get", "install", "php-fpm"]
- CMD ["/bin/echo", "Hello world"]
- ENTRYPOINT ["/bin/echo", "Hello world"]

RUN

Pour exécuter une commande (ou suite de commandes) dans le container, il faut utiliser l'instruction RUN

- Un RUN = une couche (layer)
- Les couches sont mis en cache
- Préférable de séparer les longues lignes en plusieurs lignes
  - Possible avec \

CMD

L’instruction CMD spécifie la commande qui sera exécutée lors du démarrage de l’image

- Ne peut être utilisée qu’une seule fois dans le Dockerfile
- Si plusieurs CMD sont présentes, le dernier est utilisé
- La commande et les paramètres peuvent être surchargés depuis la ligne de commande

ENTRYPOINT

- L’instruction ENTRYPOINT vous permet de configurer un container qui s’exécutera en tant qu’exécutable.
- Il ressemble à CMD, car il vous permet également de spécifier une commande avec des paramètres à exécuter.
- La différence est la commande ENTRYPOINT et les paramètres ne sont pas ignorés lorsque le conteneur Docker est lancé avec des paramètres en plus.

EXPOSE

L’instruction EXPOSE permet d’indiquer à docker que le container écoute sur les ports indiqués

Attention : ne rend pas accessible par défaut les ports depuis l’hôte

- l’option -p de la commande docker run doit être utilisé pour publier un range de port (-p 35500:22) ou -P pour publier tous les ports exposés
- un numéro de port exposé dans le container peut être un numéro de port différent accessible sur l’hôte (ex.: exposé sur le 22, accessible sur le 34500 depuis l’hôte)

COPY

L’instruction COPY copie des fichiers ou des dossiers à l’intérieur du container

- ➢ les fichiers sources ne peuvent pas être en dehors du dossier de build (donc pas de ../)
- ➢ il est possible d’utiliser des wilcards et matchings (*, ?, ...) pour les noms de source (ex.: file*)
- ➢ info : tout est créé avec l’id et gid 0
- ➢ info : la destination doit finir par un / si c’est un dossier, ou que plusieurs fichiers vont y être copiés

TP

- créer un fichier Dockerfile (c’est son nom) dans un dossier de votre projet
- Démarrer votre image d’alpine
- Installer ssh
  - Pour installer un package sur alpine **apk add package_name**
  - Pour mettre à jour le gestionnaire de package d’alpine, vous pouvez utiliser l’option **--update** avec apk add
- Générer les clés nécessaires à ssh
  - commande : **ssh-keygen -A**
- Autoriser la connexion par mot de passe et changer le mot de passe root
  - Commande : **sed -i s/#PermitRootLogin.*/PermitRootLogin\ yes/ /etc/ssh/sshd_config && echo "root:dm" | chpasswd**

- Exposer le port 22 pour pouvoir se connecter en ssh au serveur
- Lancer le serveur sshd au démarrage du container
- Builder votre image
  - N’oubliez pas de lui donner un nom (afin d’éviter une image <none>), ce sera plus simple pour la lancer
- Démarrer votre image et tenter de vous connecter à l’utilisateur root
  - N’oubliez pas de publier le port dans votre commande run pour pouvoir vous connecter sur le port exposé au sein du container

Hadolint

- Linter pour docker
- Utilisation :
  - docker run --rm -i hadolint/hadolint < Dockerfile
  - https://hadolint.github.io/hadolint/
- Liste des règles

Orchestration

docker-compose.yml

Le fichier docker-compose.yml peut contenir plusieurs services

- Un service peut se baser sur une image du hub (**image**) ou une image à builder à partir d'un dockerfile (**build**)
- Spécification des dossiers partagés (**volumes**)
- Spécification des ports à publier (**ports**)
- La commande sur laquelle se lancera le container (**command**)
- Linker un container à un autre (**depends_on**)

Générateur de docker-compose

- Okty.io
- Depuis un template ou totalement custom
- Open-source
  - N’hésitez pas à contribuer pour voir vos images favorites ;)

Commandes

- docker-compose build
  - Pour builder les images de services pour lesquelles il a été spécifié un Dockerfile
  - Donc pas nécessaire si tous les services utilisent une image du hub
- docker-compose up
  - Pour démarrer tous les containers
  - L'option -d permet de tous les lancer en background et de rendre la main dans le terminal
  - L'option --build permet de builder les images de services pour lesquelles il a été spécifié un Dockerfile
- docker-compose down
  - permet d'arrêter et de supprimer tous les containers créés par le up
- docker-compose exec <service-name> <command>
  - Equivalent du docker exec
- docker-compose start / docker-compose stop

TP

Créer votre propre docker-compose.yml et ajouter 3 services

- mariadb
- phpmyadmin
- wordpress

docker-compose.override.yml

- Permet de redéfinir la définition des services
  - Changer le port d’un service
  - Changer une variable d'environnement