15/03/2023
Version 1

DEVOPS

JÉRÉMY PERROUAULT

DOCKER

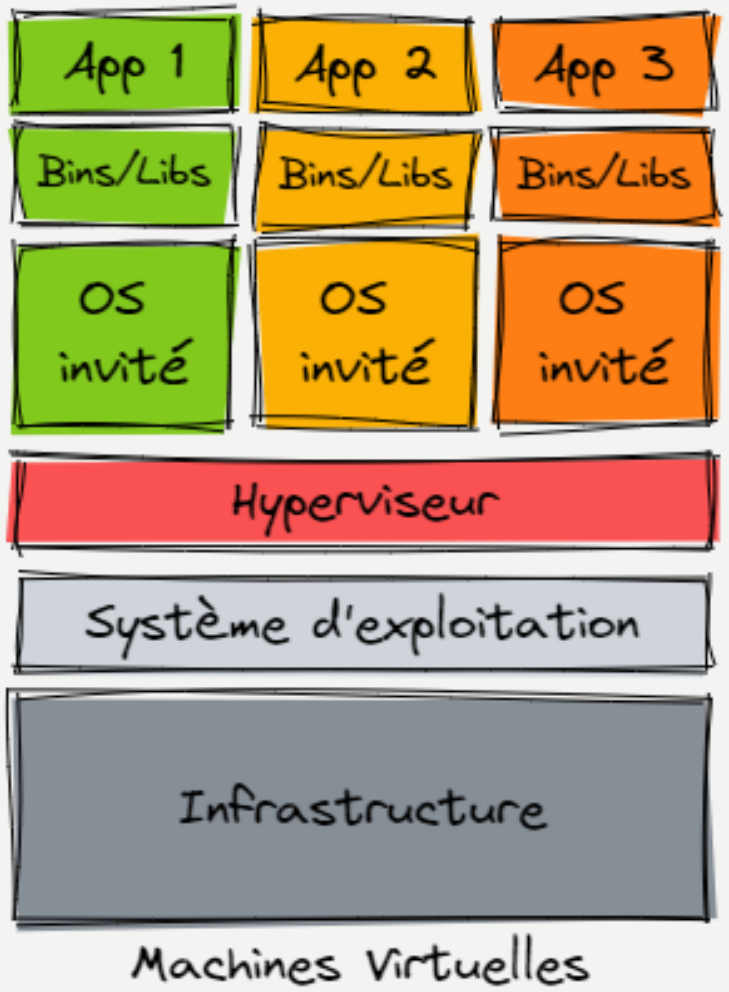
INTRODUCTION À DOCKER

DOCKER

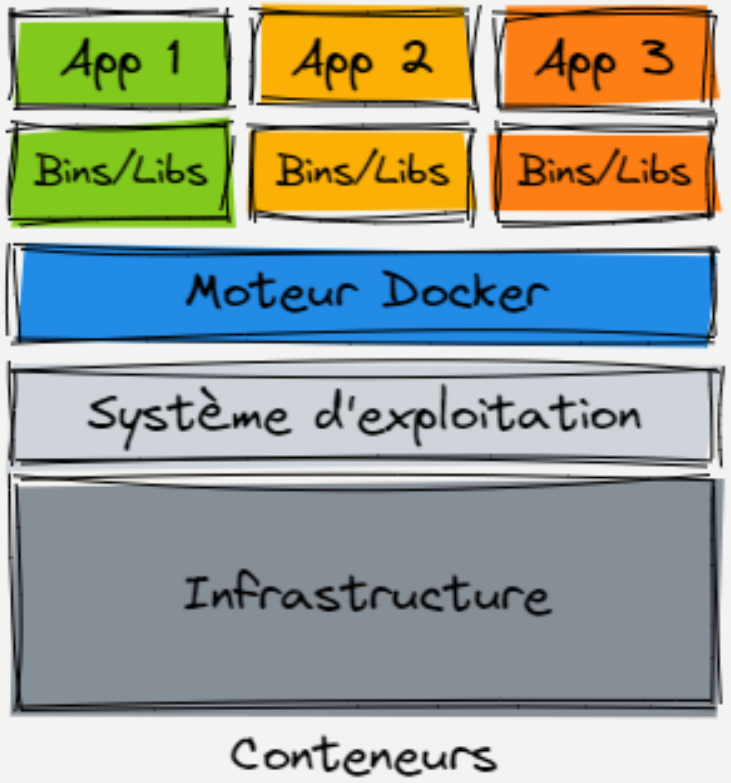
- Initialement pensé pour une société de **Platform as a Service** (PaaS), *DotCloud*
- Finalement placé en open source en 2013, projet nommé *Docker*

- Répond à plusieurs problématiques fortes
  - Problème de versions d'OS (Windows, Mac, Rocky, Ubuntu, etc.)
  - Problème de versions de dépendances ou de plateforme
  - Problème pour exécuter plusieurs versions d'un même composant sur la même machine
  - Installations diverses directement sur le poste de développement
  - Problème de déploiements
  - …
- Docker répond à toutes ces problématiques grâce aux conteneurs

DOCKER



Machines Virtuelles



Conteneurs

DOCKER

- **Docker** est décliné en trois versions
  - CE (Community Edition) – Linux uniquement
  - Enterprise – Linux uniquement
  - Desktop (Windows et Mac)
    - Sous Windows, il est préférable d'utiliser **WSL** (Windows Subsystem Linux)

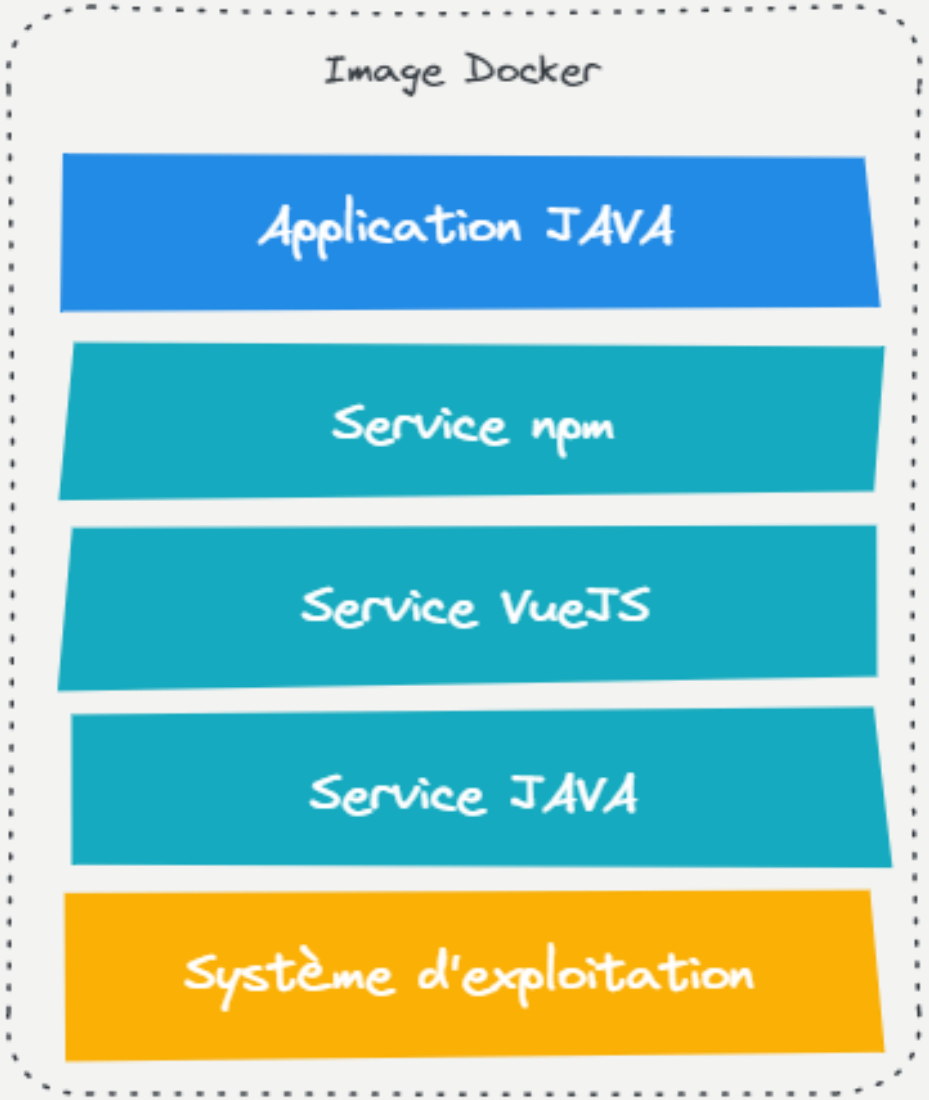
DOCKER

- Docker offre un **Registry** distant dans lequel on retrouve des *images*
  - **Le Docker Hub**
- Chaque image peut être récupérée sur la machine locale
- Chaque machine peut créer une nouvelle image
  - Et éventuellement l'envoyer sur le **Docker Hub** ou un autre **Registry**

- Les commandes `pull` et `push`, à l'instar de **git**, sont utilisées pour ces opérations

DOCKER

- Une *image*, c'est une application ou un ensemble d'applications packagées en un seul endroit
  - Pour être utilisée dans un ou plusieurs conteneur (*container*)
  - Peut être facilement partagée et déplacée (comme un fichier)
  - Inclue le code source de l'application finale, mais aussi l'environnement de configuration complet
  - A un *tag* qui permet d'identifier sa version ou sa déclinaison



DOCKER

- Plusieurs *containers* peuvent être démarrés à partir d'une même *image*
  - Il y a donc plusieurs instances de cette même *image*

```
docker run -d -p 8080:80 nginx:latest
```

- On utilise ici **Docker CLI** (Command Line Runner)
  - Le paramètre `run` permet de créer un nouveau *container* à partir de l'*image*
    - Si l'image n'a pas été trouvée pas en local, **Docker** ira la chercher sur le **Docker Hub** par défaut (`pull`)
    - Si un *container* existe déjà pour cette *image* … **Docker** en crée quand même un nouveau
  - L'option `d` permet d'exécuter le container en mode **d**`eamon`
  - L'option `p` permet de publier (**p**`ublish`) un port local vers un port du *container*
  - Le dernier paramètre est le nom de l'*image*, suivi de « : » puis d'un *tag*

https://docs.docker.com/engine/reference/commandline/run/

DOCKER

- Pour voir les *containers* qui sont exécutés

```
docker ps
```

- Pour voir tous les *containers*, y compris ceux qui ne sont pas démarrés

```
docker ps -a
```

- Pour arrêter un *container* (on précise son nom, ou les 3 premières caractères de son id)

```
docker stop 0f5
```

- Pour démarrer un *container* (on précise son nom, ou les 3 premières caractères de son id)

```
docker start nginx
```

- Pour voir la liste des images présentes sur la machine

```
docker images
```

```
docker image ls
```

DOCKER

- Pour voir les logs d'un *container* qui sont exécutés

```
docker logs nginx
```

- Pour inspecter un *container*

```
docker inspect nginx
```

- Pour exécuter une commande dans un *container*

```
docker exec nginx cmd
```

- Pour exécuter une commande interactive dans un *container*

```
docker exec -it nginx bash
```

EXERCICE

- Installer **Docker**
- Conteneuriser **nginx** et y accéder via un navigateur sur le port 5000
  - En mode `deamon`
  - Lui donner le nom « nginx »
  - Publier le port 5000 (sur le port 80)
  - Arrêter ce container, puis le relancer
- Conteneuriser **openjdk** (*tag* **17-bullseye**)
  - Publier le port 8080 (sur le port 8080)
  - Lier le volume où se trouve le projet **SPRING BOOT** sur le répertoire **/app**
  - Interagir avec le bash
    - Vérifier la version de **JAVA**
    - Exécuter le projet **SPRING BOOT**
  - Y accéder via un navigateur sur le port 8080

RÉSEAU

GÉRER LE RÉSEAU ENTRE CONTAINERS

RÉSEAU

- **Docker** met à disposition un *réseau* par défaut
  - Chaque *container* est branché sur ce *réseau*
  - Chaque *container* a sa propre adresse IP dans ce *réseau*
- Il est possible de créer d'autres *réseaux* et y connecter des *containers*

https://docs.docker.com/engine/reference/commandline/network/

DOCKER

- Pour voir les *réseaux*

```
docker network ls
```

- Pour créer un nouveau *réseau*

```
docker create -d bridge rso
```

- Pour créer un *container* sur un *réseau*

```
docker run --network rso ... image
```

- Pour connecter un *container* (existant) sur un *réseau*

```
docker network connect rso nginx
```

- Pour supprimer un *réseau*

```
docker network rm rso
```

DOCKER

- Pour créer un nouveau *réseau* avec un subnet

```
docker create --subnet 172.20.0.0/24 rso
```

- Pour connecter un *container* (existant) sur un *réseau* en spécifiant une adresse IP fixe

```
docker network connect --ip 172.20.0.1 rso nginx
```

- Pour inspecter un *réseau*

```
docker network inspect rso
```

- Pour déconnecter un *container* d'un *réseau*

```
docker network disconnect rso nginx
```

DOCKER

- On peut aussi lier des *containers* directement entre eux
  - Le *conteneur* **nginx** pourra dialoguer avec **postgres** en utilisateur **db** comme nom d'hôte

```
docker network connect --link postgres:db nginx
```

- On peut y penser dès la création du *contrainer*

```
docker run --name nginx --link postgres:db nginx:latest
```

EXERCICE

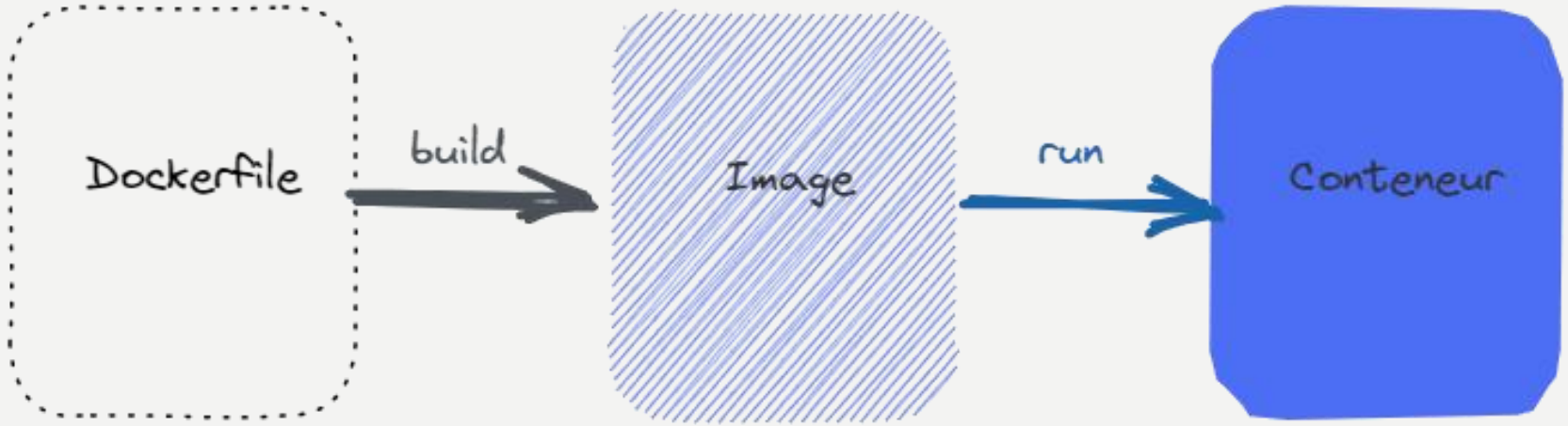
- Créer un nouveau *réseau* 172.20.0.0/24
- Créer un *container* **mysql** sur l'IP 172.20.0.2
  - En mode `deamon`
  - Lui donner le nom « mysql »
  - Attention, il faudra préciser au moins une variable d'environnement (consulter la documentation)
- Connecter le *container* **java** sur le *réseau* et l'adresse IP 172.20.0.1
  - Implémenter **SPRING DATA JPA** pour vous connecter à la base de données
    - Avec une entité Produit
    - Avec un contrôleur pour lister des Produits
    - Avec un contrôleur pour générer un nouveau Produit

BUILD

CRÉER SA PROPRE IMAGE

CRÉER SA PROPRE IMAGE

- Avec un fichier *Dockerfile*
  - On vient configurer l'image
  - Y copier le code source éventuel
  - Compiler / transpiler
  - Exposer un ou plusieurs ports
  - Exécuter l'environnement applicatif
  - …



CRÉER SA PROPRE IMAGE

- Un *Dockerfile* démarre souvent d'une autre *image* pour aider à fabriquer l'application
  - Attention, l'ordre a une importance
    - Non seulement parce que les tâches à exécuter doivent l'être dans un ordre …
    - Mais également parce que **Docker** cache par « couche »
      - Essayer de placer les lignes les moins souvent sujettes à modification au début
      - Et les autres vers la fin (le code source par exemple)

CRÉER SA PROPRE IMAGE

```
# Image de base pour la fabrication et/ou le déploiement
# On peut également lui donner un alias avec le mot-clé « AS » (voir multi-stage)
FROM image:tag

# Répertoire de travail
WORKDIR /repertoire

# Copie d'un fichier
# On peut également copier depuis une fabrique avec l'option « --from=nom »
COPY fichiersource ./fichierdestination

# Exécution d'une commande shell
RUN command shell

# Exécution du script pour exécuter l'application
CMD [ "cmd", "arg0" ]

# Exposition d'un port
EXPOSE 5000
```

https://docs.docker.com/build/building/packaging/

DOCKER

- A partir du répertoire dans lequel se trouve le fichier *Dockerfile*

```
docker build -t nomimage:tag
```

- Pour nettoyer un peu les *images* utilisées pendant le build

```
docker image prune -f
```

EXERCICE

- A partir du fichier **JS** et **JSON** (voir pages prochaines)
  - Créer l'*image* **Docker**
    - A l'aide de l'image node, version 19-alpine
    - Configurer un espace de travail sur */app*
    - Le fichier *package.json* est peu modifié
    - Le répertoire *src* peut être copié directement à la racine de l'espace de travail
    - La commande « npm install » doit être exécutée pour installer les dépendances **JS**
    - La commande « node server.js » sera exécutée pour lancer l'application **NodeJS**
    - Le port 8080 sera exposé
  - Créer un *container* à partir de cette *image*, et le publier sur le port local 5500

EXERCICE

- Fichier *src/server.js*

```
const express = require('express');
const app = express();

app.get('/', (req, resp) => {
  resp.send("Hello, World!");
});

app.listen(8080, () => {
  console.log("Application démarrée sur le port 8080");
});
```

EXERCICE

- Fichier *package.json*

```
{
  "name": "node-app",
  "version": "1.0.0",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

EXERCICE

- Procéder à la création d'une *image* pour le projet **JAVA**
  - L'image utilisera **Maven** pour packager (voir *multi-stage*)
  - Elle copiera le jar généré dans le répertoire /app
  - Elle démarrera l'application
  - Elle exposera le port 8080
- Créer un *container* à partir de cette *image*, et le publier sur le port 5600