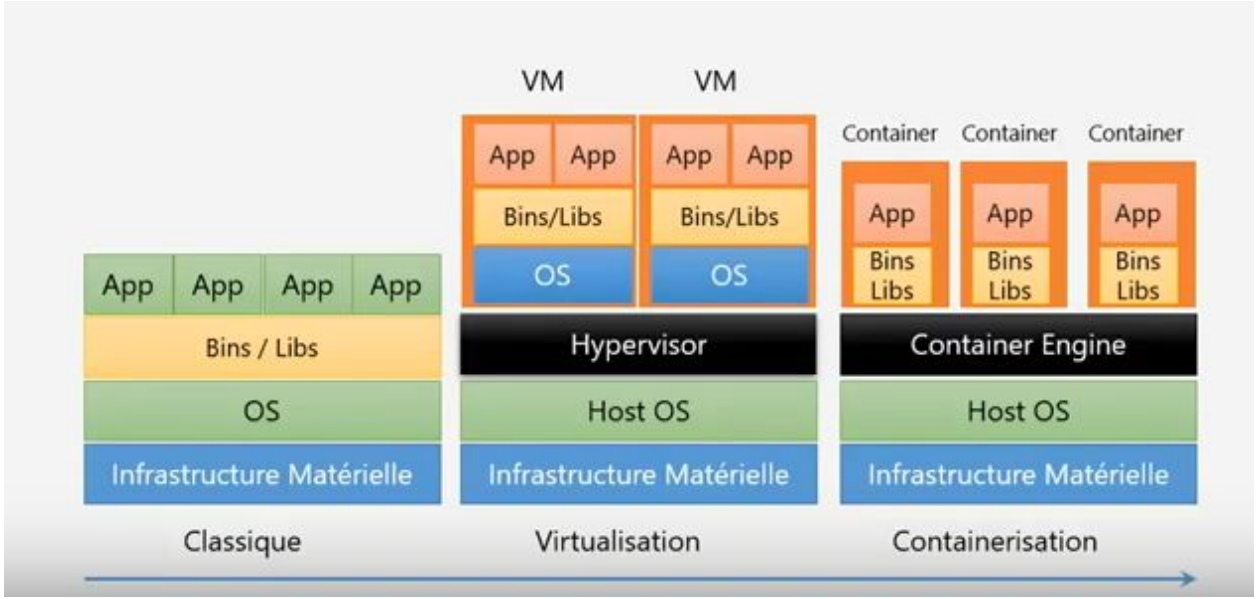
Introduction à docker

Plan de cours

- Containerisation vs virtualisation
- C’est quoi Docker ? c’est quoi un contenair ?
- Les commandes de bases
- Demo Project

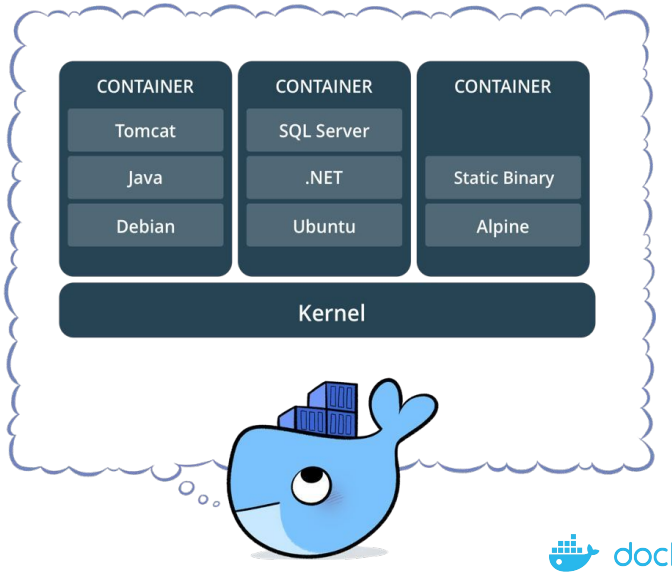
Containerisation vs virtualisation



docker

Conteneur: définition

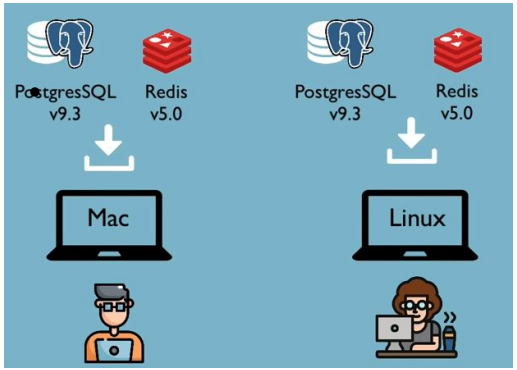
- Un conteneur enveloppe une application dans une boîte invisible avec tout ce dont elle a besoin pour s'exécuter.
- Docker exécute des conteneurs qui accueillent vos services : base de données, serveur web, test...
- Facilite le développement et le déploiement d'un service.
-

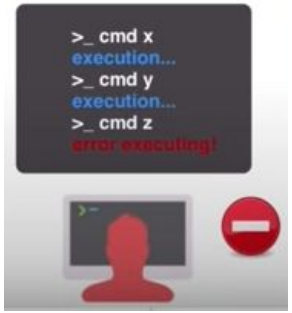


docker

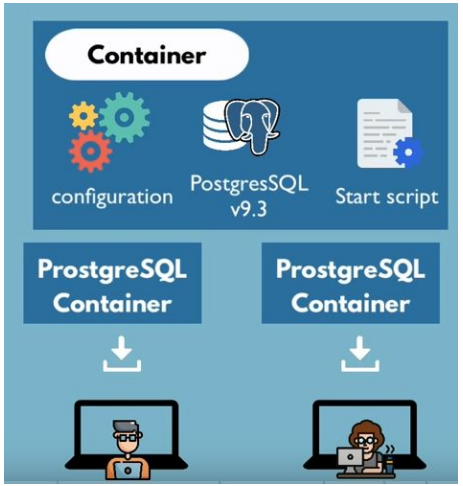
DEV-APP : before and after

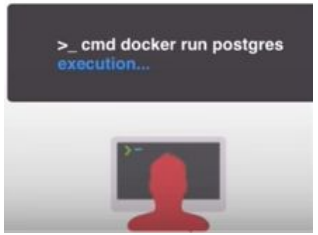
- Environnement logiciel à installer dépend de l'OS
- Conflits de dépendances





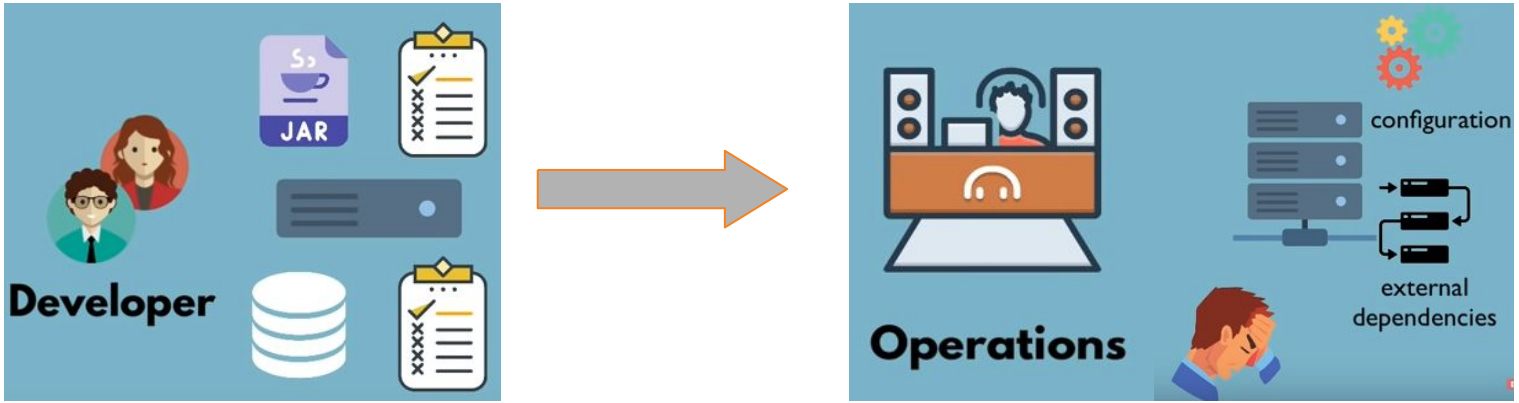
- Environnement logiciel isolé
- Une unique commande à exécuter pour installer l'application indépendamment de l'OS





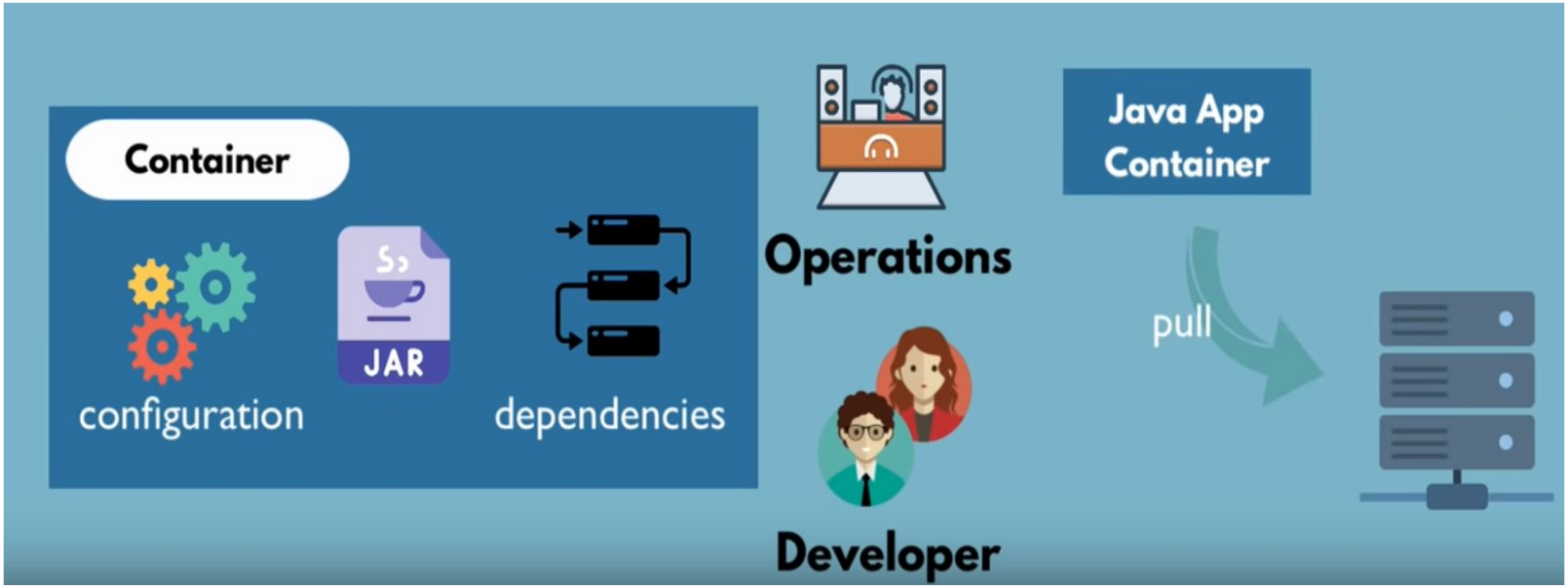
docker

DEPLOY-APP : before and after



- Configurer l'environnement au niveau du serveur
- Satisfaire les exigences de l'application => gérer les conflits de dépendances
- Perte de temps

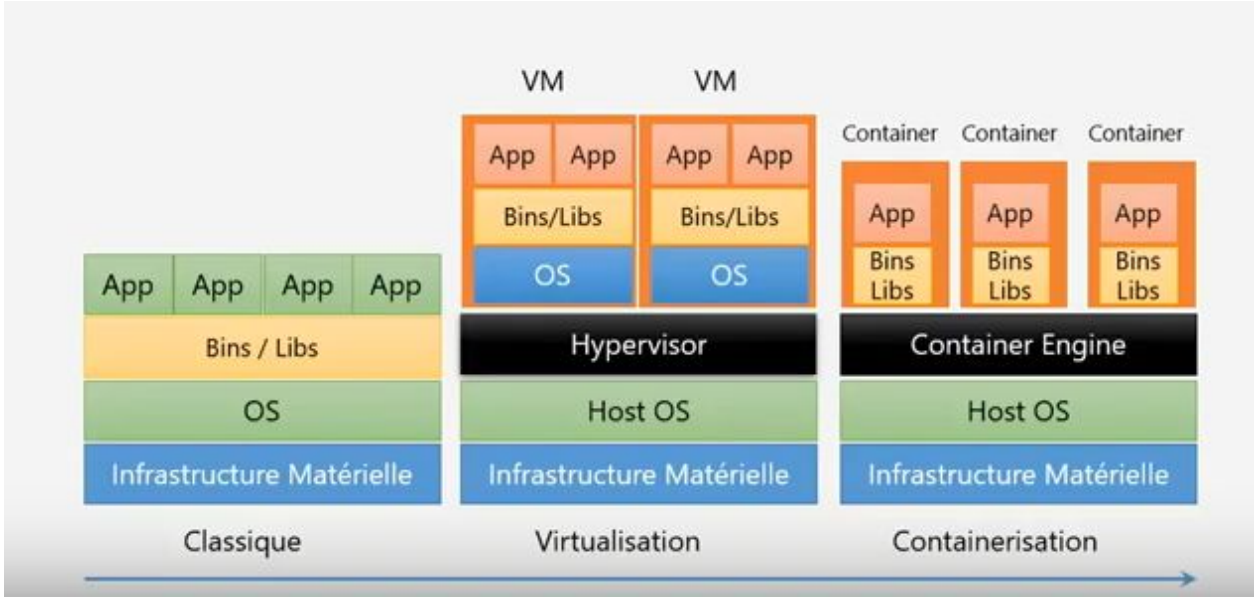
DEPLOY-APP : before and after



- Les DEV et les OPS travaillent ensemble pour empaquetter l'application dans un conteneur
- Pas de configuration d'environnement au niveau du serveur

docker

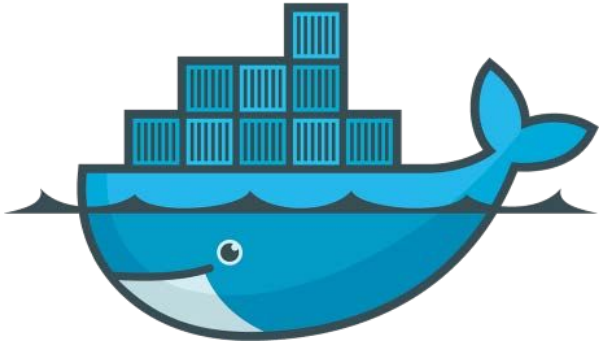
Containerisation vs virtualisation



docker

Docker: définition

What Is Docker?



- Docker est une plateforme permettant de créer, de déployer et de gérer des conteneurs d'applications virtualisées sur un système d'exploitation.
- Modèle avec architecture de type client / serveur
  - Client Docker (ligne de commande)
  - Moteur Docker (docker daemon)
- Docker hub : système de stockage pour les images de conteneurs.

docker

Docker: Ecosystème

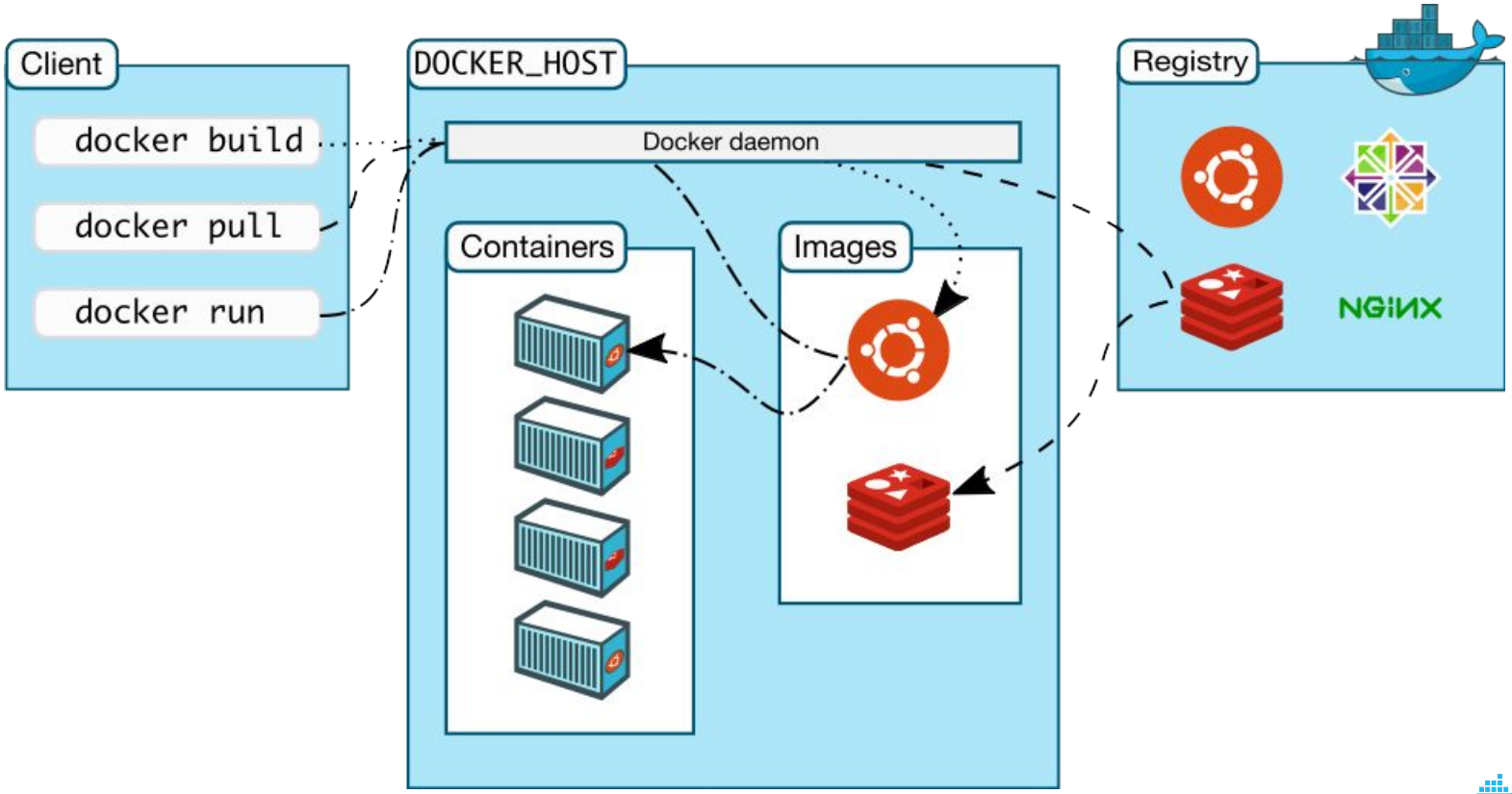


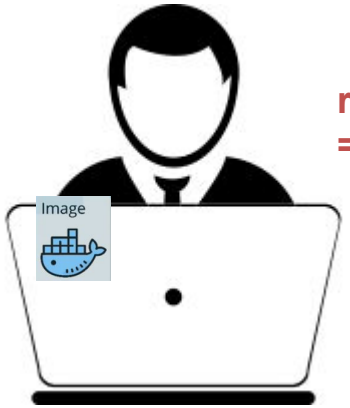
Image vs Conteneur

- Un package de toutes les dépendances et informations nécessaires pour créer un conteneur.

not running

- Chaque image commence par une image de base (ex: Ubuntu) ou plus « avancée » comme Node.js installé sur Ubuntu
- Chaque image est ensuite constituée d'une série de couches baptisées « layers »

- lancer l'application
- créer l'environnement du conteneur



running
=>conteneur

docker

Docker: installation sous Linux

- Vérifiez si Docker est déjà installé sur votre machine :

```
$ docker version      # il faudra le préfixer avec sudo
```

  Si vous obtenez une réponse sans erreur, il n'y a rien d'autre à faire.

- Deux alternatives pour installer Docker :
  - ❖ Installation depuis les paquets des distros

```
$ sudo yum  install docker
$ sudo apt−get install docker.io
```

  - ❖ Utilisez le script d'installation de Docker. Disponible pour Ubuntu, Debian, Fedora et Gentoo :

```
$ curl −s https://get.docker.com/ | sudo sh
```

Docker: configuration

- Si votre utilisateur n'est pas dans le groupe Docker, vous devrez préfixer chaque commande avec sudo; par exemple. sudo docker --version.

- Pour éviter cela, ajoutez votre utilisateur au groupe Docker :
    - Ajouter le groupe docker

```
$ sudo groupadd docker
```

    - Ajouter votre utilisateur au groupe docker

```
$ sudo usermod -aG docker $USER
```

    - Redémarrer le daemon docker

```
$ sudo service docker restart
```

docker

Docker: premier conteneur

Docker exécute une image dans un conteneur

```
docker run hello-world

   Unable to find image 'hello-world:latest' locally
   latest: Pulling from library/hello-world
   4276590986f6: Pull complete
   Status: Downloaded newer image for hello-world:latest

   Hello from Docker!
```

1. Le client docker se connecte au docker daemon,
2. Docker daemon a cherché dans son répertoire local pour savoir si l'image hello-world était déjà téléchargée,
3. Docker daemon va donc télécharger l'image du docker Hub et l'instancier.

Docker: deuxième conteneur

```
$ docker run -i -t ubuntu /bin/bash
```

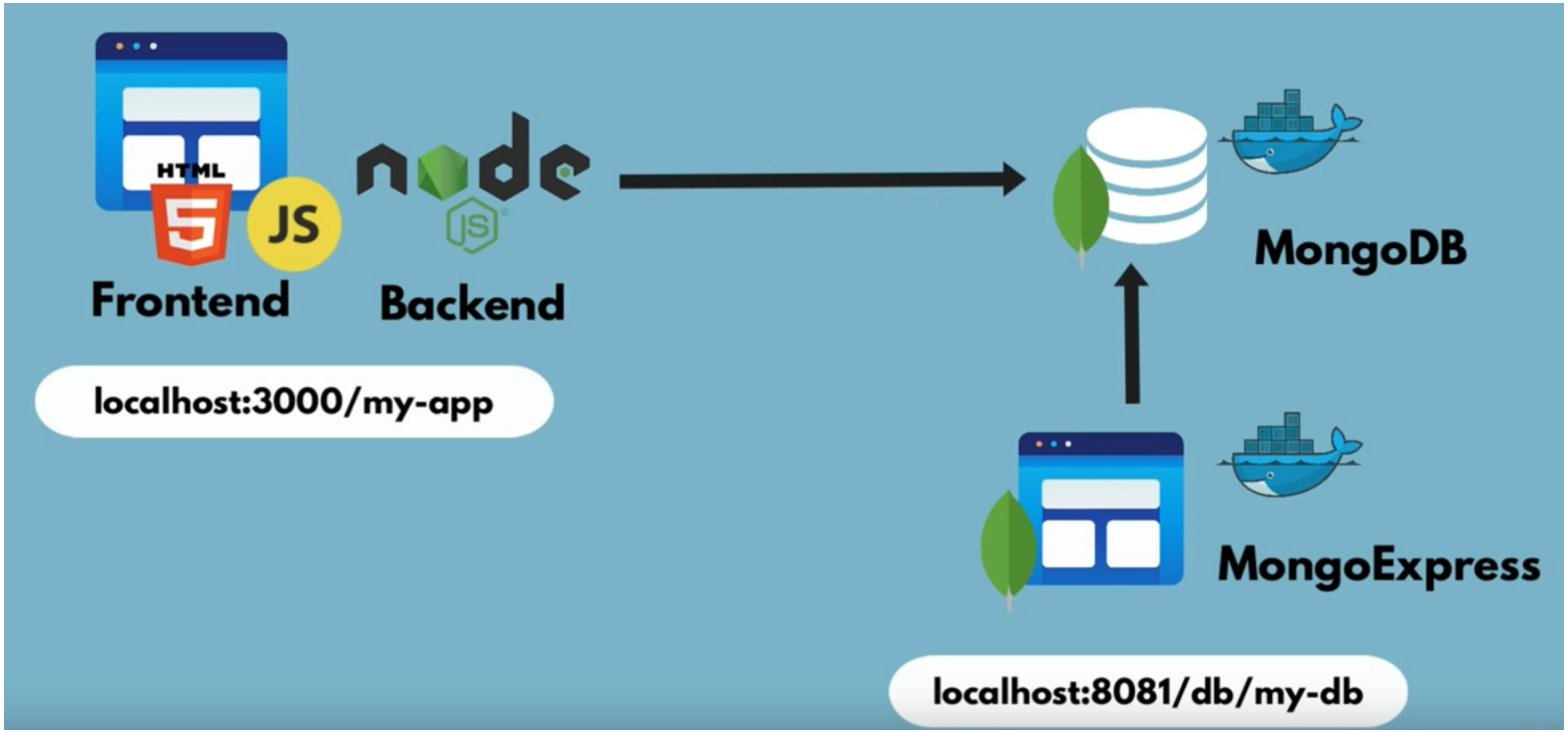
- **`run`** : lance un nouveau conteneur
- **`-i -t`** : demande une terminal en mode interactif dans une pseudo-terminal
- **`ubuntu`** : l'image à utiliser pour ce conteneur. Si elle n'est pas disponible sur votre ordinateur, elle sera téléchargée du *Docker Hub*.
- **`/bin/bash`** : exécute bash dans le conteneur

```
$ docker run -i -t ubuntu /bin/bash
root@0bc82356b52d9:/# cat /etc/issue
Ubuntu 14.04.2 LTS
root@0bc82356b52d9:/# exit
```

Docker: commandes de base

```
$ docker search ubuntu  #give ubuntu images from public index ( official /trusted )

$ docker pull stackbrew/ubuntu #pull latest stackbrew/ubuntu images

$ docker images # show local images

$ docker run -i -t stackbrew/ubuntu /bin/bash #run this image / create container

$ docker run -t -i -link redis :db -name webapp ubuntu bash #link 2 containers

$ docker ps #show active containers (-a to show all containers )

$ docker logs myUbuntu

$ docker attach myUbuntu #retake the hand on the container

$ docker run -d -p 8888:80 ubuntu #export 8888 on master
```

Docker demo project

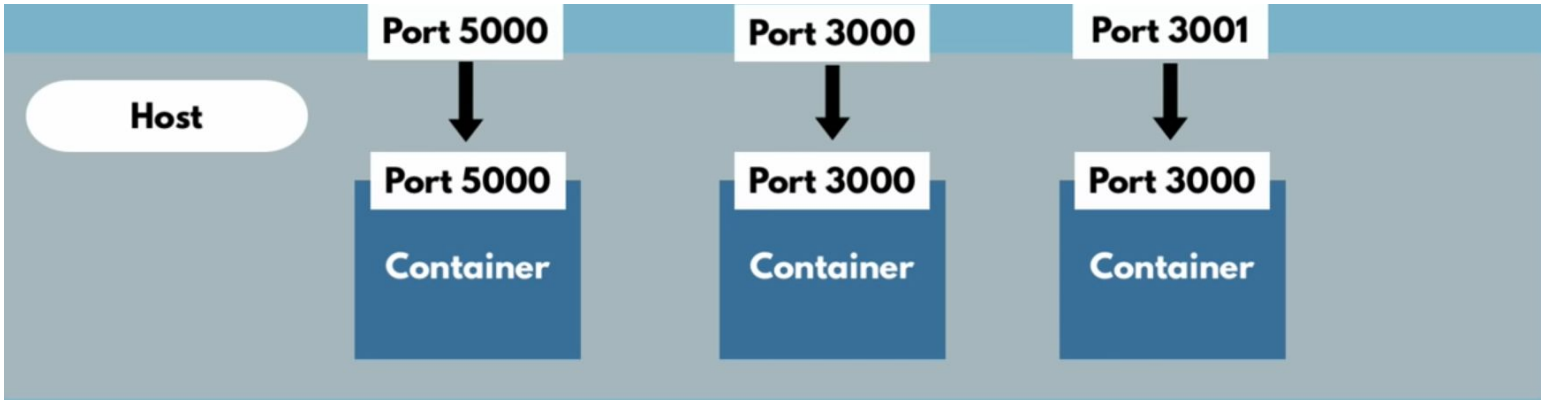


docker

Container port vs Host Port

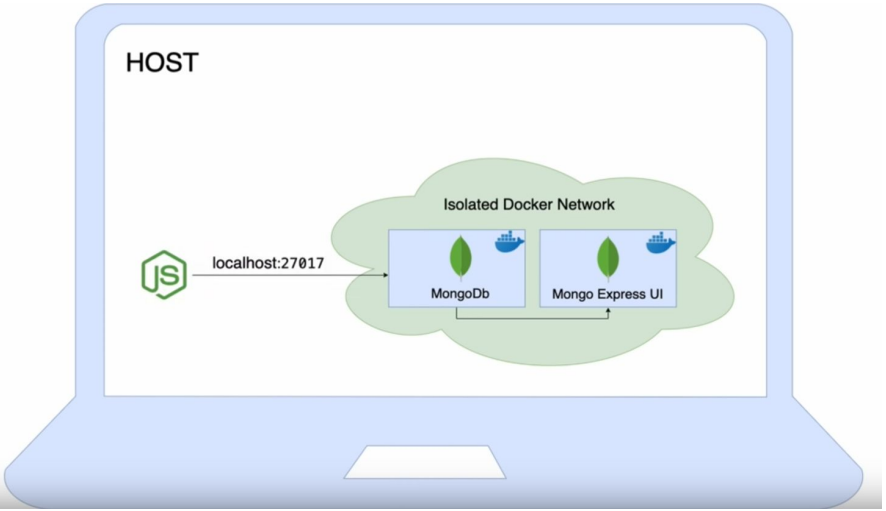
- Plusieurs conteneurs peuvent s'exécuter simultanément sur la machine HOST
- Votre machine dispose d'un nombre limité de ports disponibles
- Vous pouvez utiliser le même numéro de port pour deux conteneurs

some-app://localhost:3001



docker

Docker network & create container



$ docker network create mongo-network

```
docker run -d \
--name mongodb \
-p 27017:27017 \
-e MONGO_INITDB_ROOT_USERNAME=admin \
-e MONGO_INITDB_ROOT_PASSWORD=password \
--net mongo-network \
mongo

docker run -d \
--name mongo-express \
-p 8081:8081 \
-e ME_CONFIG_MONGODB_ADMINUSERNAME=admin \
-e ME_CONFIG_MONGODB_ADMINPASSWORD=password \
-e ME_CONFIG_MONGODB_SERVER=mongodb \
--net mongo-network \
mongo-express
```

Docker compose

Docker run command

```
docker run -d \
--name mongodb \
-p 27017:27017 \
-e MONGO_INITDB_ROOT_USERNAME=admin \
-e MONGO_INITDB_ROOT_PASSWORD=
password \
--net mongo-network
mongo

docker run -d \
--name mongo-express
-p 8081:8081
-e ME_CONFIG_MONGODB_ADMINUSERNAME=admin
-e ME_CONFIG_MONGODB_ADMINPASSWORD=password
-e ME_CONFIG_MONGODB_SERVER=mongodb
--net mongo-network
mongo-express
```

mongo-docker-compose.yaml

```
version: '3'
services:
  mongodb:
        image: mongo
        ports:
        - 27017:27017
        environment:
        - MONGO_INITDB_ROOT_USERNAME=admin
        - MONGO_INITDB_ROOT_PASSWORD= password
  mongo-express:
        image: mongo-express
        restart: always
        ports:
        - 8080:8081
        environment:
        - ME_CONFIG_MONGODB_ADMINUSERNAME= admin
        - ME_CONFIG_MONGODB_ADMINPASSWORD=password
        - ME_CONFIG_MONGODB_SERVER=mongodb
```

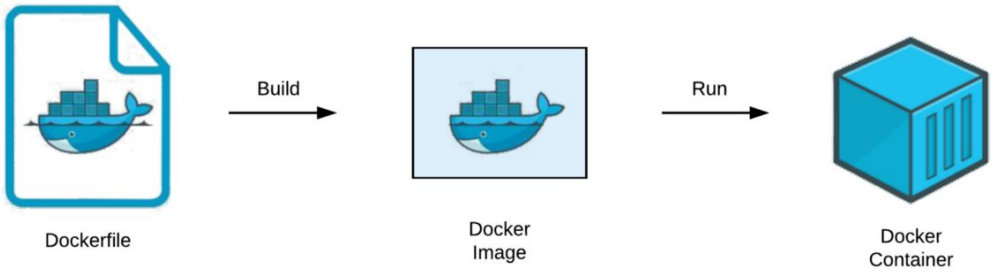
docker

Docker compose

```
$ docker-compose -f mongo-docker-compose.yaml up -d
 #start all containers
$ docker-compose -f docker-compose.yaml down
```

Dockerfile

- C'est un fichier texte avec un ensemble de commandes ou d'instructions.
- Ces commandes / instructions sont exécutées successivement pour effectuer des actions sur l'image de base afin de créer une nouvelle image docker.



Dockerfile

Image Environement

Install node
set MONGO_DB_USERNAME=admin
set MONGO_DB_PWD=password

create /home/app directory

copy current folder files to /home/pp

start the app with “node server.js”

Dockerfile

```
FROM node:13-alpine

ENV MONGO_DB_USERNAME=admin \
        MONGO_DB_PWD=password

RUN mkdir -p /home/app

COPY ./app /home/app

# set default dir so that next commands executes in /home/app dir
WORKDIR /home/app

# will execute npm install in /home/app because of WORKDIR
RUN npm install

# no need for /home/app/server.js because of WORKDIR
CMD ["node", "server.js"]
```

docker

Dockerfile

```
$ docker build -t myapp:1.0 .
$ docker run myapp:1.0
```

docker