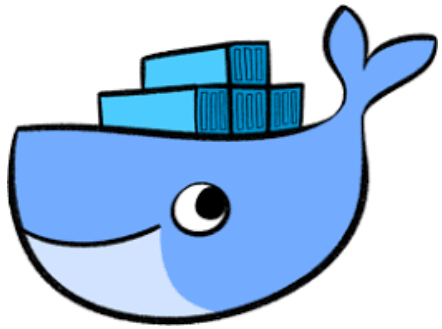
Docker

C. Brun

2018



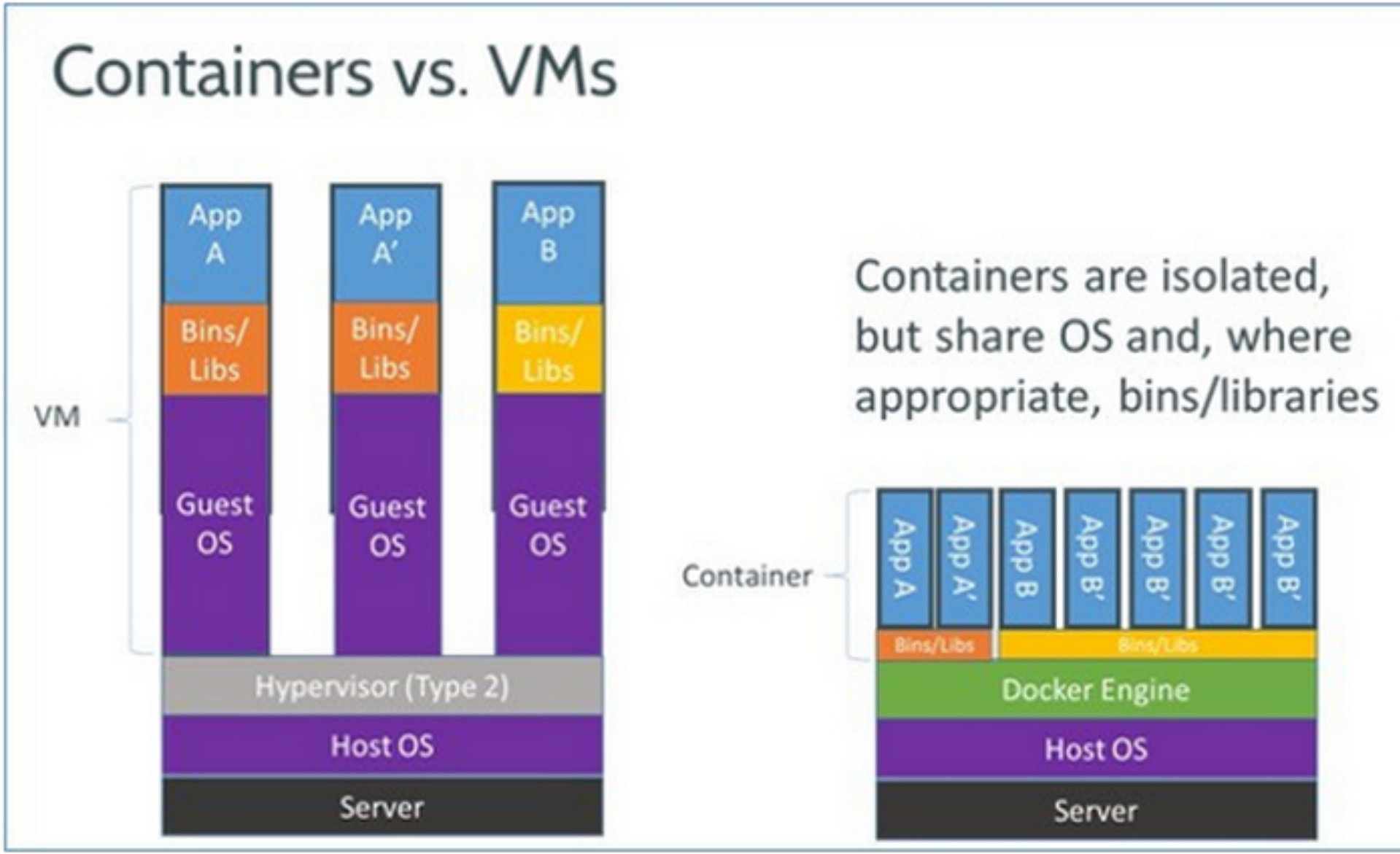
Introduction

- Pourquoi Docker
  - simplicité d'utilisation
  - Empreinte légère
  - Workflow de déploiement
  - Système isolé
- Un succès
  - Large communauté
  - des millions de containers
  - les gros de l'hébergement : AWS, Azure,
  - lié à la culture DevOps
  - GOlang

Vocabulaire

- Images
- Container
- Docker Engine : Daemon + CLI
- Couches
- Repository

VM vs. Docker


Containers vs. VMs
VM
App A
Bins/ Libs
Guest OS
App A'
Bins/ Libs
Guest OS
App B
Bins/ Libs
Guest OS
Hypervisor (Type 2)
Host OS
Server
Containers are isolated, but share OS and, where appropriate, bins/libraries
Container
App A
App A'
App B
App B'
App B'
App B'
App B'
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

Images

- image est à docker ce que la classe est à la POO.
- description de la structure du docker.
- Permet l'instanciation du container
- Composée de plusieurs couches (layers)
- la couche de base est une image
- les couches : delta au niveau filesystem avec la couche inférieure
- structure en oignon.
- une image peut toujours être une image de base

Attention

- à la cible : i386, arm
- à l'OS : couches manquantes > taille de l'image
- windows : FS a posé des pb aux dev –> performance

Dockerfile

- Recette de construction d'une image

```
FROM nodered/node-red-docker
MAINTAINER Christophe Brun <christophe@cbrun.fr>

RUN npm install node-red-dashboard; \
    npm install node-red-contrib-looptimer; \
    npm install node-red-contrib-loose; \
    npm install node-red-contrib-quotes; \
    npm install node-red-contrib-repeat; \
    npm install node-red-contrib-slackbot; \
    npm install node-red-contrib-soap; \
    npm install node-red-contrib-soapserver; \
    npm install node-red-node-dropbox; \
    npm install node-red-node-forecastio; \
    npm install node-red-node-google; \
    npm install node-red-node-random; \
    npm install node-red-node-pushover

VOLUME ["/data"]
EXPOSE 1880

ENV FLOWS=flows.json
ENV NODE_PATH=/usr/src/node-red/node_modules:/data/node_modules

CMD ["npm", "start", "--", "--userDir", "/data"]
```

Build/Run d'une image

```
docker build -t [ImageName] .
```

- attention au point en fin : Dockerfile dans le répertoire courant
- -t : permet de définir le tag de l'image = le nom

```
docker run --name [containerName] [ImageName]
```

- Lance l'image par création d'un container
- Container = instance d'image

Commandes dans un Dockerfile

- FROM : image de base pour la construction de notre image
- MAINTAINER
- RUN : lance la commande dans le shell de construction de l'image (et pas sur le host)
- COPY : permet de copier des fichiers du host sur le filesystem de l'image
- ADD : comme COPY mais prend en charge les archives
- VOLUME : expose une partie du filesystem
- EXPOSE : expose un port (network)
- ENV : expose une variable d'environnement et la valorise
- ENTRYPOINT : commande lancée lors de l'instanciation
- CMD : commande par défaut
- WORKDIR : Définition du répertoire de travail

Dockerfile du viewer

```
FROM elixir:latest
MAINTAINER Christophe Brun <christophe@cbrun.fr>

RUN mix local.hex --force ; \
    mix local.rebar --force; \
    mix archive.install https://github.com/phoenixframework/archives/raw/master/
     phoenix_new.ez --force

RUN curl -sL https://deb.nodesource.com/setup_6.x -o nodesource_setup.sh; \
    bash nodesource_setup.sh; \
    apt-get install nodejs

RUN mkdir /app
COPY .  /app
WORKDIR /app

ENV AWS_ACCESS_KEY_ID="AWS_ACCESS_KEY_ID" \
    AWS_SECRET_ACCESS_KEY="AWS_SECRET_ACCESS_KEY" \
    AWS_REGION="AWS_REGION" \
    S3_PATH="S3_PATH" \
    HOST="https://api-dev.cbrun.com" \
    ROLLBAR_TOKEN="ROLLBAR_TOKEN" \
    MIX_ENV="prod" \
    PORT="4000"

RUN rm -rf deps \
    rm -rf _build

RUN mix deps.get --only prod; \
    mix compile; \
    npm install gulp -g; \
```

Commandes utiles

- lister les images

```
docker images
```

- lister les containers

```
docker ps        # list les containers actifs
docker ps -a     # list tout les containers
```

- Start / Stop un container

```
docker start [ContainerName]     # redemarre une container (attention, different de run)
docker stop [ContainerName]      # stop et snapshot container
```

- supprimer un container

```
docker rm [ContainerName]    # destruction du snapshot
```

- Supprimer une image

```
docker rmi [ImageName]      # a condition de ne plus avoir d'instances de l'image
```

Repository / Hub

- docker/docker hub : git/github ??
- https://hub.docker.com/
- Repository d'image plus ou moins "utilisable"
- possibilité d'héberger vos images
- Permet de trouver des images de base
- Aide pour lancer les containers

```
docker pull [ImageName]       # recuperation d'une image
docker push [ImageName]       # push d'une image
```

- Exemple

```
docker run --name my-custom-nginx-container -v /host/path/nginx.conf:/etc/nginx/nginx.
    conf:ro -d nginx
docker run -it -p 1880:1880 --name mynodered nodered/node-red-docker
docker run -d -P -p 4444:4444 -p 5900:5900 --name selenium_firefox selenium/standalone-
    firefox-debug:2.53.0
docker run -d -e POSTGRES_USER=cbrun -e POSTGRES_PASSWORD=cbrun -e POSTGRES_DB=
    cbrun_elixir -p 5432:5432 --name cbrun_elixir postgres:latest
docker  run -d -e MYSQL_DATABASE=cbrunsite  -e MYSQL_USER=cbrunsite   -e MYSQL_PASSWORD=
    XXXXXXXX -p 3306:3306 --name cbrunsite mysql
```

Docker-compose

- Lancement d'une architecture de container
- Description de l'infrastructure des services (containers) pour avoir une application
- les commandes

```
docker-compose up        # Lancement de l'infra la 1ere fois
docker-compose start     # relance de l'infra
docker-compose stop      # stop l'infra : snapshot les containers
```

docker-compose.yml

- C'est le dockerfile de docker-compose

```
version: '3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Docker-compose exemple 2

```
version: '2'
services:
  scheduler:
    image: spiside/luigi:latest
    ports:
      - "8082:8082"
    environment:
      LUIGI_TASK_HISTORY_DB_CONNECTION: postgres://luigi@db/dev
      LUIGI_SCHEDULER_RECORD_TASK_HISTORY: "true"
    depends_on:
      -  db
    entrypoint:
      - ./wait-for-postgres.sh
      - db
  worker:
    build:
      dockerfile: docker/Dockerfile.worker
      context: .
    hostname: worker
    volumes:
      - .:/luigi
    environment:
      LUIGI_CORE_DEFAULT-SCHEDULER-URL: http://scheduler:8082
    depends_on:
      - scheduler
  db:
    image: postgres:9.5.4
    ports:
      - "5433:5432"
    environment:
      POSTGRES_USER: luigi
```

Exemple 3

```
version: '3'
services:
  db:
    image: postgres:9.6
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  web:
    build:
      context: rails/.
    working_dir: /mnt/rails
    env_file: .env
    command: bundle exec rails s -p 5000 -b '0.0.0.0'
    volumes:
      - ./rails:/mnt/rails
    ports:
      - "80:5000"
    links:
      - db
    depends_on:
      -db

volumes:
  postgres_data:
```

Lancement de l'infra

```
docker-compose start
Starting db        ... done
Starting scheduler ... done
Starting worker    ... done
docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED   STATUS
     PORTS                     NAMES
861ffa1e230c   dockerluigi_worker       "run_worker"             7 days    Up 2 seconds
                           dockerluigi_worker_1
60d9056a7266   spiside/luigi:latest     "./wait-for-postgres"    7 days    Up 7 seconds
     0.0.0.0:8082->8082/tcp   dockerluigi_scheduler_1
408ed778cea9   postgres:9.5.4           "/docker-entrypoint"     7 days    Up 12 seconds
     0.0.0.0:5433->5432/tcp   dockerluigi_db_1
```

Optims

- Influence des layers
- Taille de l'image : 750 Mo

Optims : Dockerfile

```
FROM fedora:21
MAINTAINER "Christophe Brun" <christophe.brun.cl194@gadz.org>

RUN yum -y update
RUN yum -y install httpd supervisor php php-imap php-mcrypt php-gd php-pear-Net-Curl php
     -mysqlnd php-pear
RUN yum clean all

ADD start.sh /start.sh
RUN chmod -v +x /start.sh
RUN echo "Apache" >> /var/www/html/index.html

EXPOSE 80
ADD supervisord.conf /etc/supervisord.conf

CMD ["/start.sh"]
```

Optims suite

```
FROM fedora:21
MAINTAINER "Christophe Brun" <christophe.brun.cl194@gadz.org>

RUN yum -y update && yum -y install httpd supervisor php php-imap php-mcrypt php-gd php-
     pear-Net-Curl php-mysqlnd php-pear && yum clean all

ADD start.sh /start.sh
RUN chmod -v +x /start.sh && echo "Apache" >> /var/www/html/index.html

EXPOSE 80
ADD supervisord.conf /etc/supervisord.conf

CMD ["/start.sh"]
```

- Taille de l'image 140 Mo (vs 750 Mo)

Warnings

- performances : locale > Docker > VM, mais fonction de l'OS
- Espace disk : attention lors de compilation failed (création d'image None)
- Optimisation : attention à l'image et la cible : préférable de recompiler sur le même os (de - en - vrai)
- Mise à jour : attention aux mise à jour de docker parfois obligé de réinstaller les images
- Monitoring : on commence à avoir des outils
- clustering : l'orchestrateur est important
- sous Windows : il est nécessaire de compiler l'image sous Windows ;-(
- La taille des images/containers sur le filesystem sur le host n'est pas représentative
- **DROITS** : attention, les users dans le container
- **PERSISTENCE** : simplicité du snapshot vs. volume

Tips

- travailler le docker non optimisé et optimiser à la fin (gains au build)
- Avoir plusieurs versions sur un host : Exemple posgtresql, des firefox (mais pas d'IE)
- tests plus poussés : écritures concurrentes
- tests crash sans effet : isolation
- cross-compiling :
  - compilation arm sur i386 sans installation de la toolchain complète
  - rapidité : compilation domoticz sur raspi2 (20 à 45mn) sur portable (4 à 7mn)
- test rapide d'un application : exemple graylog / luigi
- Faire un poc en utilisant des images existantes : T4E : openocr
- déploiement : push/pull
- pouvoir "lancer" d'anciennes applications : php4
- avoir un env de développement rapidement
- API Restful
- Wrappers

Tips suite

- commandes docker

```
docker exec -ti [tagname] bash    # lance un bash dans un container
docker inspect [tagname]          # detail du container (port mapping, volume)
docker logs [tagname]             # logs du container
docker inspect [tagname] | awk '/IPAddress/ {if ($2 != "null,")  print $2 }' | uniq |
     awk -F \" '{print $2}'
docker commit                     # creation d'image
```

- outils
  - beaucoup, beaucoup
  - cockpit, swarm visualizer
  - docker console
  - dockviz : alias dockviz="docker run -it –rm -v /var/run/docker.sock:/var/run/docker.sock nate/dockviz"
  - docker run -it clearlinux/machine-learning
  - docker pull terminus7/deep-learning