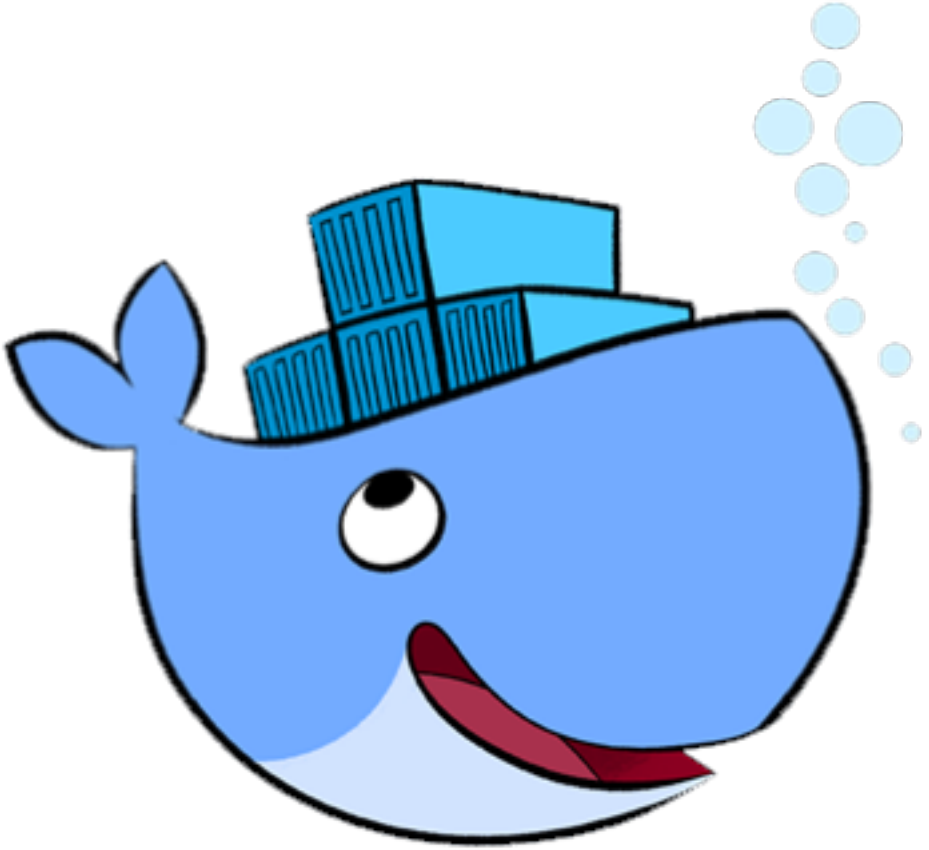


~~Une baleine bleue au LIAS~~

Docker, c'est quoi ?

Mickaël BARON – 2019
mailto:baron.mickael@gmail.com ou mailto:baron@ensma.fr
@mickaelbaron

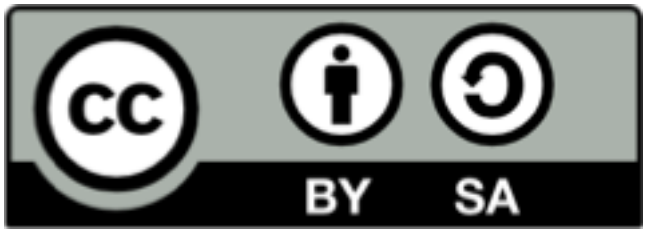
Licence

Creative Commons

Contrat Paternité

Partage des Conditions Initiales à l'Identique

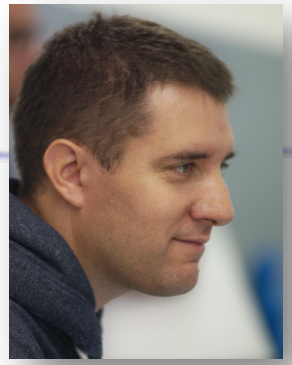
2.0 France



http://creativecommons.org/licenses/by-sa/2.0/fr

@mickaelbaron

A propos de l'auteur ...



- **Mickaël BARON**
- Ingénieur de Recherche au LIAS
  - *https://www.lias-lab.fr*
  - Equipe : Ingénierie des Données et des Modèles
  - Responsable des plateformes logicielles, « coach » technique
- Responsable Rubriques Java de Developpez.com
  - Communauté Francophone dédiée au développement informatique
  - *https://java.developpez.com*
  - 4 millions de visiteurs uniques et 12 millions de pages vues par mois
  - 750 00 membres, 2000 forums et jusqu'à 5000 messages par jour

@mickaelbaron

@mickaelbaron

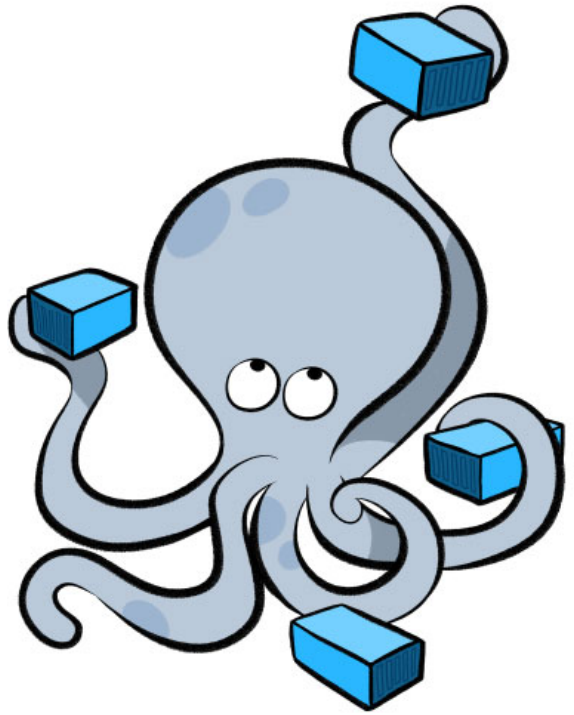
Disclaimer

- De nombreux raccourcis ont été utilisés
- Niveau : débutant (avoir créé une machine virtuelle)
- Présentation rapide de Docker (format 30 min)
  - Présentation des principaux concepts (juste pour démarrer)
  - Démonstrations en « live » (applications utilisées au laboratoire)
- Pour aller plus loin ...
  - *https://speakerdeck.com/mickaelbaron/architectures-microservices-mise-en-oeuvre* : support de cours sur les microservices et Docker
  - *https://github.com/mickaelbaron/javamicroservices-tutorial* : tutoriel
  - *https://mbaron.developpez.com/tutoriels/ros/environnement-developpement-ros-docker* : utilisation de Docker avec ROS

@mickaelbaron

Plan de la présentation

- Contexte
- Docker
  - Généralités
  - Image, conteneur, volume et redirection de ports
- Conclusion



@mickaelbaron

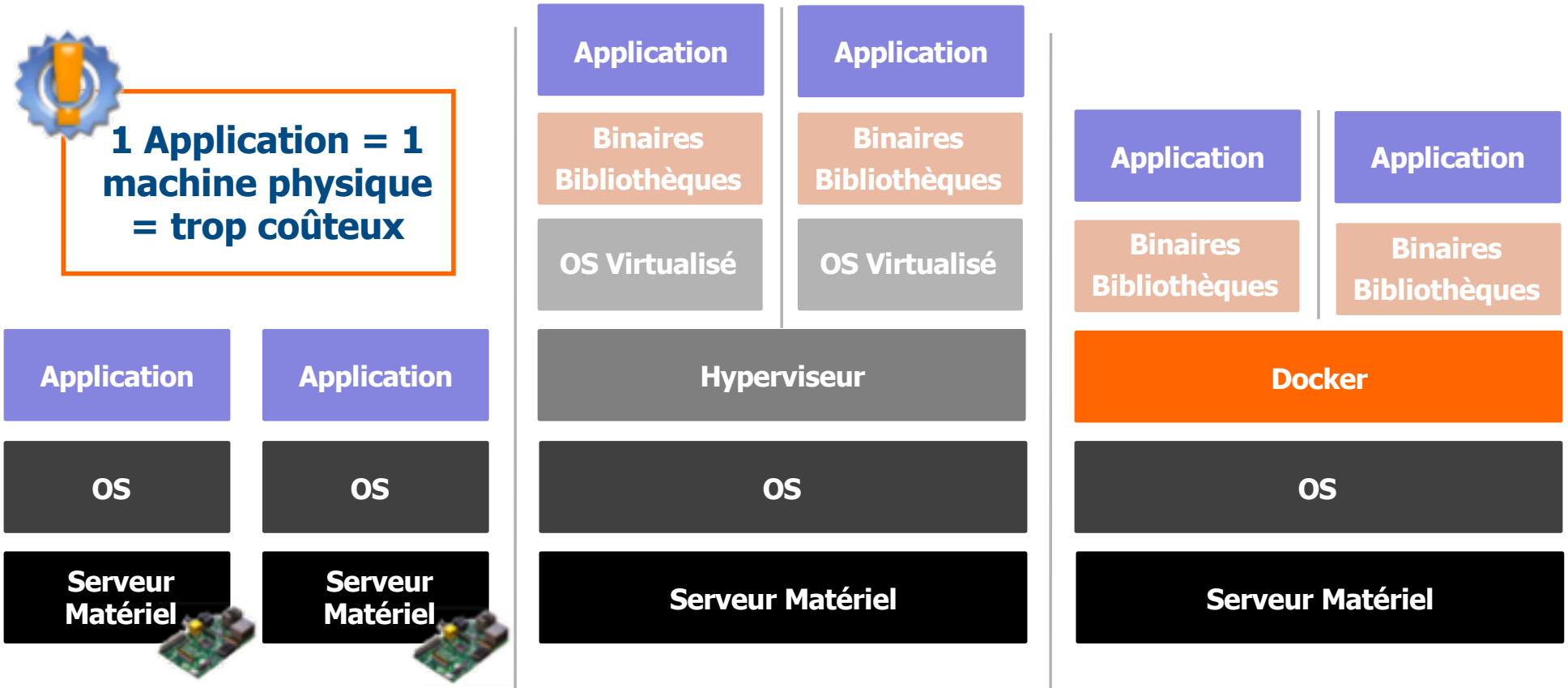
Contexte : besoin d'isoler une application

- Exécuter une application sur un OS non supporté
  - Exemple : ROS qui ne fonctionne que sous Ubuntu 14.04
- Multiplier le nombre d'instances d'une même application
  - Exemple : monter en charge une application web
- Exécuter une application en mode « bac à sable »
  - Exemple : tester un prototype de recherche
- Reproduire son environnement de production
  - Exemple : PHP, MySQL, compilateur…
- Tester plusieurs versions d'une application pour des tests
  - Exemple : différentes versions de javac (8, 9, 10, 11, 12…)

@mickaelbaron

Comment isoler : machine virtuelle ou conteneur ?

- Pour isoler une application => utilisation de trois techniques



Isoler : Machine Virtuelle VS Conteneur

Comparatif sur les performances

	Machine Virtuelle (574 Mo)	Conteneur (100 Ko)
Démarrage	~ 2 minutes	0.2 seconde
Mémoire	min 256 Mo	~ 0.3 Mo
Espace Disque	min 1 Go	~ 0.1 Mo

2500 conteneurs ont été créés dans un Raspberry PI 2

https://blog.docker.com/2015/09/update-raspberry-pi-dockercon-challenge/

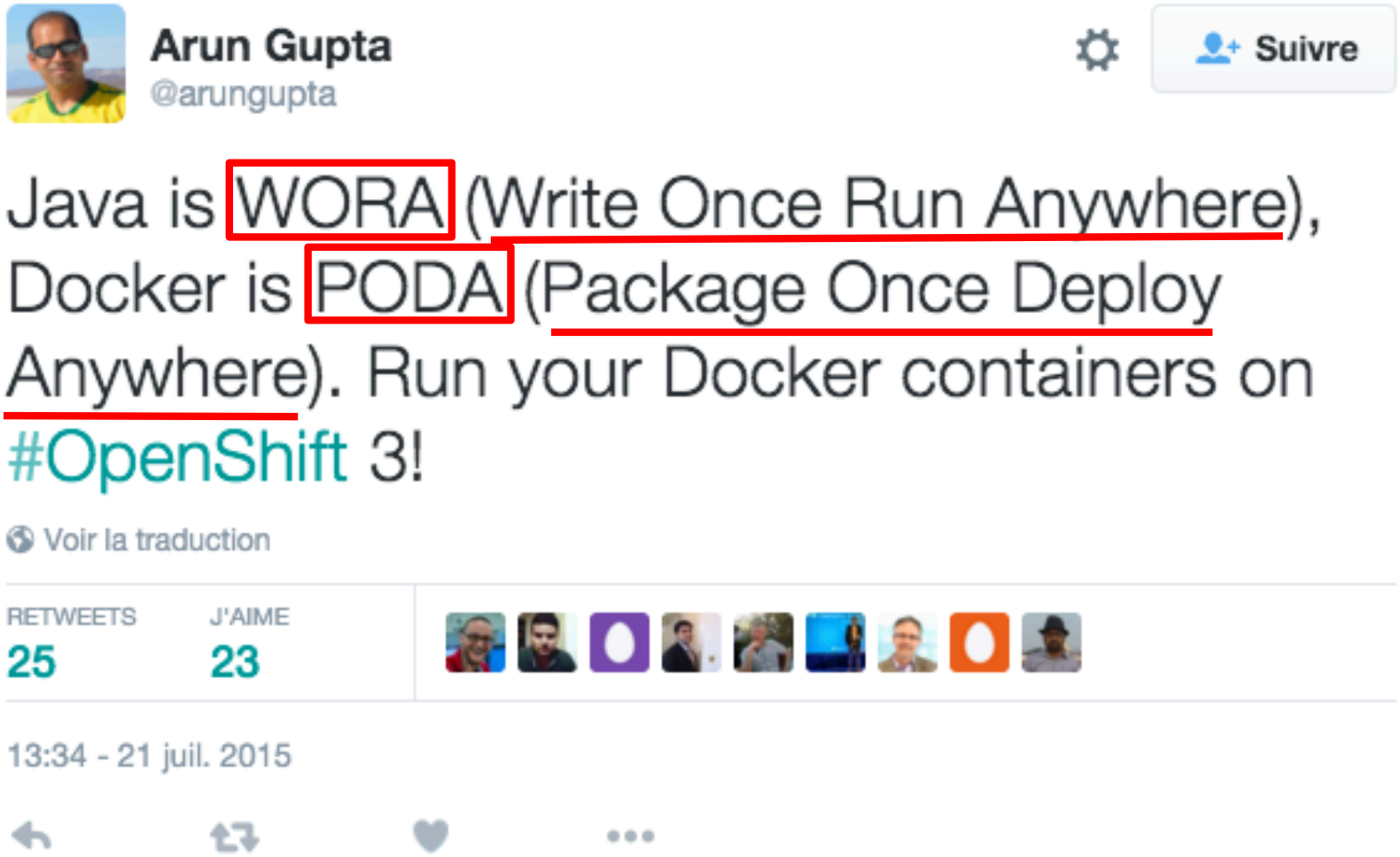
@mickaelbaron

Docker : généralités

- Docker Inc. est une jeune entreprise mars 2013
- Sites web de référence
  - *https://www.docker.com*
  - Documentation : *https://docs.docker.com/*
  - Installation : *https://docs.docker.com/engine/installation/*
- Propose une suite d'outils du même nom que la société
- Basé principalement sur
  - LinuX Containers (LXC) : *https://en.wikipedia.org/wiki/LXC*
  - AUFS : *https://en.wikipedia.org/wiki/Aufs*
- Outils sont principalement en ligne de commande
- Fonctionne sous Linux nativement et sous Windows et MacOS via une machine virtuelle minimaliste

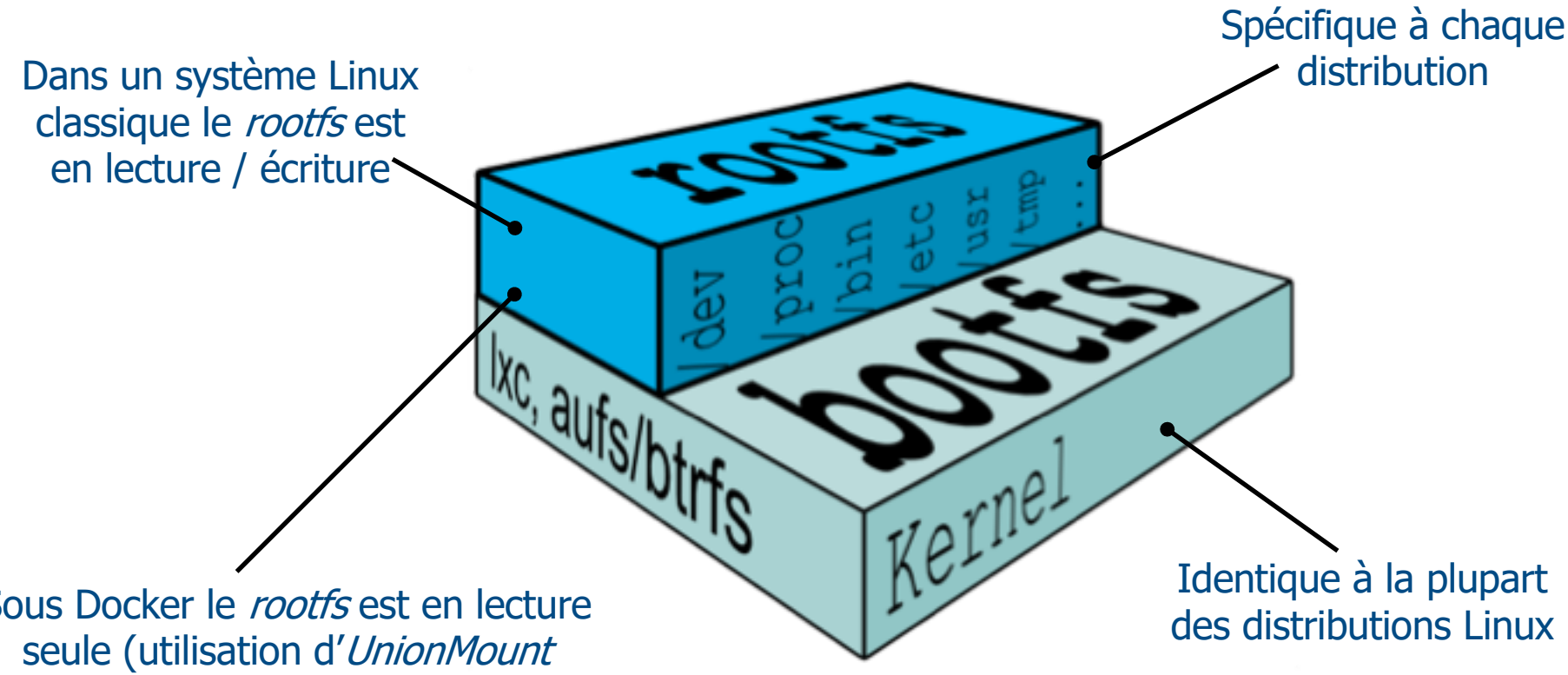
@mickaelbaron

Docker : un // avec Java



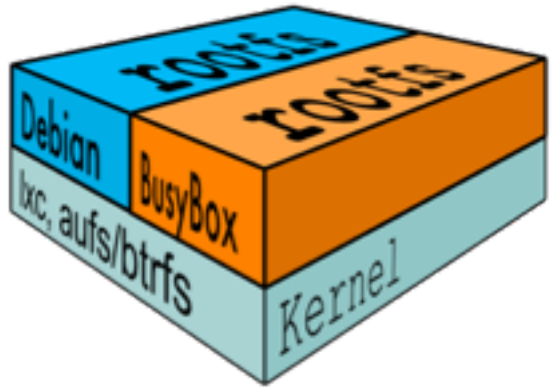
Docker : images

- Dans un système Linux classique, deux systèmes de fichiers
  - *bootfs* : contient le « boot loader » et le noyau (kernel)
  - *rootfs* : contient la structure des répertoires (*/usr*, */lib*, */bin…*)



Docker : images

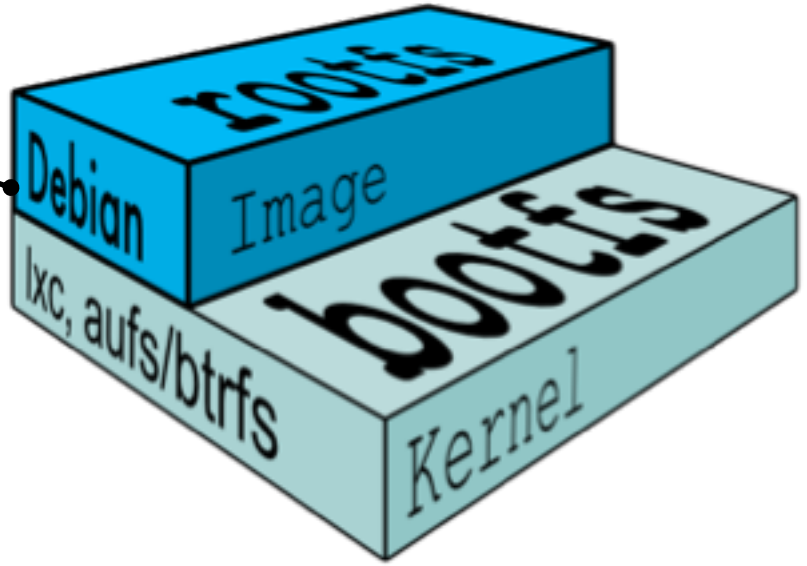
- Plusieurs *rootfs* ?
  - Sous un système Linux classique un seul *rootfs*
  - Sous Docker possibilité de plusieurs *rootfs*



- Le *rootfs* constitue l'**image de base** d'un conteneur Docker

Image de base

Sans image pas de conteneur
… patience le concept de conteneur est expliqué

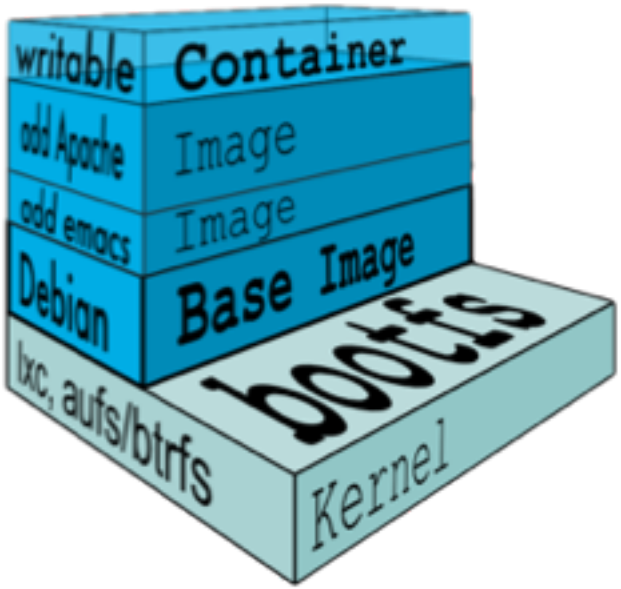


Une image ne peut être modifiée puisqu'elle est en lecture seule

@mickaelbaron

Docker : images

- Une image est décomposée d'un ensemble de couches appelé *layer* qui compose l'image finale
- Chaque couche est liée à la couche inférieure dite *parent*
- Exemple
  - Image finale = *Apache* + *Emacs* + *Debian*
  - Image de base = *Debian*
  - *Emacs* est une couche de l'image finale
  - *Emacs* est la couche parente de celle d'*Apache*

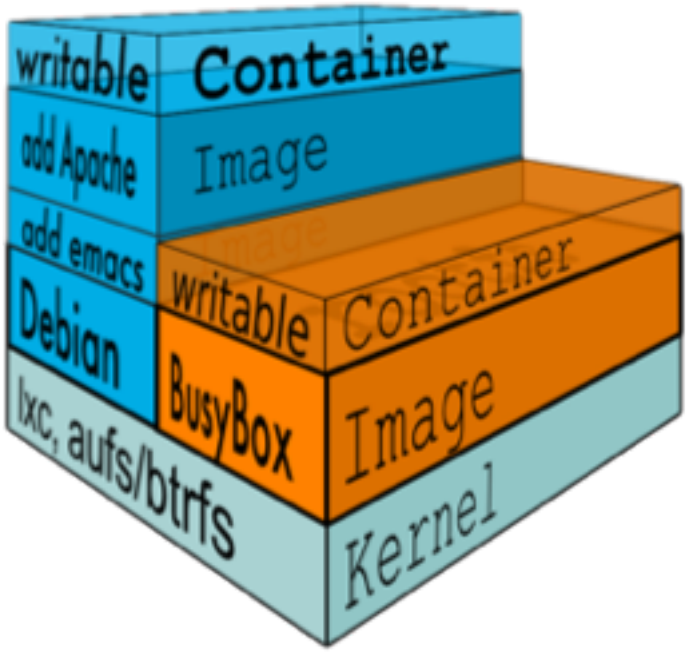


- Facilite la réutilisation des images
- Où trouver des images ?
  - *https://hub.docker.com*

@mickaelbaron

Docker : conteneur

- Un **conteneur** est une **couche modifiable** qui est liée à la couche inférieure
- Comparaison avec le monde « Java »
  - bootfs = JVM
  - image de base = classe Object
  - image = classe
  - relation parent = héritage
  - conteneur = instance d'une classe



- Seul le **conteneur** est en **lecture** et **écriture**

@mickaelbaron

Docker : conteneur interactif (démo 1)

➤ Commandes Docker : **pull**, **run**, **ps**

```
# Télécharge l'image 'busybox' depuis DockerHub
$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox

c00ef186408b: Pull complete
ac6a7980c6c2: Pull complete
Digest: sha256:e4f93f6ed15a0cdd342f5aae387886fba0ab98af0a102da6276eaf24d6e6ade0
Status: Downloaded newer image for busybox:latest

# Création d'un conteneur
$ docker run -it busybox /bin/sh # -i = interactif –t dans la console en cours
```

```
# Liste le contenu du 'rootfs' du conteneur
$ ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var

# Liste les processus du conteneur
# ps -ef

# Fermeture du conteneur (le conteneur se termine)
# exit
```

```
# Affichage des conteneurs (en cours, arrêtés, stoppés...)
$ docker ps –a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                      PORTS   NAMES
d2bc3ccd2ee1   busybox   "/bin/sh"   26 minutes ago   Exited (0) 23 minutes ago           goofy_bhabha
```

Docker : conteneur détaché (démo 2)

➤ Commandes Docker : **run, logs, ps, attach**

```
# Démarre un processus très long
$ JOB=$(docker run -d busybox /bin/sh -c 'while true; do echo Hello LIAS $(date); sleep 1; done')

# Affichage des informations
$ docker logs $JOB
...
Hello LIAS Thu Jan 7 15:09:01 UTC 2016
Hello LIAS Thu Jan 7 15:09:02 UTC 2016

# Vérification de l'état d'exécution
$ docker ps
CONTAINER ID  IMAGE    COMMAND              CREATED        STATUS         PORTS   NAMES
01d4dacd55e2  busybox  "/bin/sh -c 'while"  1 minutes ago  Up 25 seconds          reverent_franklin

# Attacher un conteneur
$ docker attach $JOB
...
Hello LIAS Thu Jan 7 15:17:22 UTC 2016
Hello LIAS Thu Jan 7 15:17:23 UTC 2016 # Pour terminer un CTRL-C

# Vérification de l'état d'exécution
$ docker ps -a
CONTAINER ID  IMAGE    COMMAND              CREATED        STATUS           PORTS  NAMES
01d4dacd55e2  busybox  "/bin/sh -c 'while"  9 minutes ago  Exited (0) 44 s         reverent_franklin

# Suppression d'un conteneur
$ docker rm $JOB # Un conteneur doit être arrêté pour le supprimer ou forcer par l'option -f
```


@mickaelbaron

Docker : conteneur sans état (démo 3)

➤ Commandes : **run, exec, diff, rm**

```
# Création de fichiers dans le répertoire /tmp
$ JOB=$(docker run -d busybox
          /bin/sh -c 'while true ; do /bin/touch /tmp/$(date +%H%M%S); sleep 60; done')

# Liste le contenu du répertoire /tmp
$ docker exec $JOB /bin/ls /tmp
074426
074526

# Changement sur le système de fichier – C = Changement, A = Ajout, D = Delete
$ docker diff $JOB
C /tmp
A /tmp/094126
A /tmp/094226

# Suppression du conteneur
$ docker rm -f $JOB
01dbf7a101abc13a80a95fae41c78a3997de90fa9636c26a4c6498738c631c26

# Si nouveau conteneur les anciens fichiers ne sont plus là
$ docker run -it busybox /bin/ls /tmp

# N'avez-vous jamais voulu faire cela ? ‘rm –rf /usr’
$ docker run -it busybox /bin/sh -c '/bin/rm -rf /usr;/bin/ls /'
```


@mickaelbaron

Docker : créer une image

- À chaque création d'un conteneur, le système de fichiers correspond à celui de l'image « parente »
- Lors de l'**arrêt du conteneur** les modifications apportées sont **perdues**
- Questions ? J'aimerais pouvoir conserver ce qui suit
  - Installations de programmes
  - Organisation de répertoires
  - Configuration (réseaux, programmes…)
- Solutions => créer une nouvelle image
- Pour créer une nouvelle image
  - Manuellement via la commande **commit**
  - Via l'utilisation d'un fichier **Dockerfile**

@mickaelbaron

Docker : créer une image (cas d'étude réel)

- Cas d'étude : rdf3x
  - Moteur de stockage de données RDF
  - Interpréteur SPARQL (en ligne de commande)
- Limitations de la version actuelle rdf3x
  - Ne compile (C++) que sous Linux (et problème si pas Ubuntu 14.04)
  - Pas de SPARQL Endpoint (interface web) pour l'interrogation
- Objectifs => configurer son poste de développeur
  - Proposer une image Docker de rdf3X
  - Créer des conteneurs Docker pour importer des données et exécuter des requêtes SPARQL

@mickaelbaron

Docker : créer une image « manuellement » (démo 4)

Commandes : **run, commit, rm, images**

```
# Créer un conteneur à partir d'une image de base
$ docker run –it --name gh_rdf3x ubuntu:16.04 /bin/sh

# Exécuter les commandes
$ apt-get update -y && apt-get install -y git g++ make
...
$ git clone https://github.com/gh-rdf3x/gh-rdf3x && cd gh-rdf3x && make
...

# S'assurer que la compilation s'est bien déroulée
$ ls bin
buildmonetdb  buildpostgresql  cts  infra  makeutil  rdf3xdump   rdf3xembedded  rdf3xload
rdf3xquery  rdf3xreorg  rdf3xupdate  rts  tools  translatesparql

# Création d'une image appelée mickaelbaron/gh_rdf3x_manual
$ docker commit gh_rdf3x mickaelbaron/gh_rdf3x_manual
C40410395aedf8cfb8f55469a974a519ee764dd3a765e3c78df8df60316870ad # => Id de l'image

# Suppression du conteneur actuel
$ docker rm -f gh_rdf3x
gh_rdf3x

# Affiche la liste des images en cache
docker images
REPOSITORY                    TAG        IMAGE ID        CREATED              VIRTUAL SIZE
mickaelbaron/gh_rdf3x_manual  latest     c40410395aed    About a minute ago   533 MB
Ubuntu                        latest     ac6a7980c6c2    5 weeks ago          123 MB

# Création d'un conteneur à partir de la nouvelle image
$ docker run –it --name gh_rdf3x mickaelbaron/gh_rdf3x_manual /bin/sh
```

@mickaelbaron

Docker : créer une image depuis un Dockerfile

- Un fichier *Dockerfile* pour créer automatiquement des images
  - Provisionner (installer des applications, copier des fichiers…)
  - Configuration (ouverture de port, commande en tâche de fond…)
- Command **build** pour construire une image

```
$ docker build -t imagename .
```

- *Dockerfile* composé d'instructions
- Format du fichier
  - # pour les commentaires
  - INSTRUCTION arguments
- **Chaque instruction** est exécutée **indépendament** et suivie par un **commit** => image à chaque instruction

@mickaelbaron

Docker : Dockerfile (démo 5)

➤ Préparation à la construction d'une image Docker RDF-3X

FROM => Indique l'image de base

```
FROM ubuntu:16.04
LABEL maintener="Mickael BARON"

RUN apt-get update -y && apt-get install -y git g++ make

RUN git clone https://github.com/gh-rdf3x/gh-rdf3x && cd gh-rdf3x && make

WORKDIR /output
```

RUN => Exécute des commandes Linux dans une nouvelle couche

WORKDIR => Précise le répertoire courant à la création du conteneur

Fichier *Dockerfile* du projet *docker-gh_rdf3x*

Docker : Dockerfile (démo 5)

➤ Commades : **build, images**

```
# Création d'une image à partir du précédent Dockerfile
$ docker build -t mickaelbaron/gh_rdf3x .

Sending build context to Docker daemon 223.7MB
Step 1/5 : FROM ubuntu:latest
---> 20c44cd7596f
Step 2/5 : LABEL maintener="Mickael BARON"
---> Using cache
---> 73f94671388f
Step 3/5 : RUN apt-get update -y && apt-get install -y git g++ make
---> Using cache
---> ffbd1b1fdfa2
Step 4/5 : RUN git clone https://github.com/gh-rdf3x/gh-rdf3x && cd gh-rdf3x && make
---> Using cache
---> 1b20588165a9
Step 5/5 : WORKDIR /output
---> Running in 72228244978a
Removing intermediate container 72228244978a
---> c91b6a7cb2e2
Successfully built c91b6a7cb2e2
Successfully tagged mickaelbaron/gh_rdf3x:latest



$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
mickaelbaron/gh_rdf3x_manual    latest              17d5fe7340f6        4 seconds ago       533MB
mickaelbaron/gh_rdf3x           latest              c91b6a7cb2e2        12 days ago         533MB
ubuntu                          latest              20c44cd7596f        18 months ago       123MB
```

Docker : persister les données avec les volumes

- Un conteneur est volatile : toutes les données produites sont perdues lors de la destruction du conteneur (logs, fichiers…)
- Comment s'assurer que si je recrée mon conteneur je retrouverai mes données ? => utilisation des **Volumes**
- Il s'agit d'un **dossier** qui se trouve **à la fois** sur le **hôte** et sur le **conteneur** (un peu comme un dossier partagé)
- Pour créer un volume, utilisation du paramètre `-v` lors de la création d'un conteneur
- Syntaxe de l'option : `-v hostdirectory:/containerdirectory`

@mickaelbaron

Docker : les volumes (démo 6)

➤ Commandes : **run**

```
$ ls $(pwd)/data
lubm-data.nt query.sparql

# Création d'une base RDF-3X via un conteneur Docker
# -v $(pwd)/data:/data : associe $(pwd)/data du hôte au répertoire /data du conteneur (où seront
stockées les données d'entrées .nt et requêtes SPARQL)
# -v $(pwd)/output:/output : associe $(pwd)/output du hôte au répertoire /output du conteneur (où
seront stockées les bases RDF-3X)
$ docker run --rm -v $(pwd)/data:/data -v $(pwd)/output:/output mickaelbaron/gh_rdf3x
          /gh-rdf3x/bin/rdf3xload /output/datadb /data/lubm-data.nt

$ ls $(pwd)/output
datadb

# Exécution d'une requête SPARQL à partir d'une base RDF-3X créée précédemment
$ docker run --rm -v $(pwd)/data:/data -v $(pwd)/output:/output mickaelbaron/gh_rdf3x
          /gh-rdf3x/bin/rdf3xquery /output/datadb /data/query.sparql
RDF-3X query interface
(c) 2008 Thomas Neumann. Web site: http://www.mpi-inf.mpg.de/~neumann/rdf3x
(c) 2013 Hancel Gonzalez and Giuseppe De Simone. Web site: http://github.com/gh-rdf3x/gh-rdf3x
<http://www.w3.org/2002/07/owl#Ontology> 190
<http://www.w3.org/2002/07/owl#Class> 43
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#Publication> 76529
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#AssistantProfessor> 1822
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#AssociateProfessor> 2291
<http://www.w3.org/2002/07/owl#Restriction> 8
...
```

@mickaelbaron

Docker : rediriger les ports

- Lors de la création d'un conteneur, une adresse IP lui est assignée dans un sous-réseau Docker
- Pour accéder à un port particulier du conteneur
  - en passant par l'adresse IP du conteneur:PORT
  - en passant par la redirection de port
- La redirection de port permet d'accéder au conteneur en utilisant l'accès réseau du système hôte
- Syntaxe
  - `-p portHôte:portConteneur` : mappe *portHôte* avec *portConteneur*
  - `-P` : mappe tous les ports du conteneur avec l'hôte

@mickaelbaron

Docker : rediriger les ports (démo 7)

➤ Commandes : **run**

```
# Créer un conteneur Postgres avec une redirection du port 5432 vers le 54320 de l'hôte
$ docker run --name postgres54320 -p 54320:5432 -d postgres
efee2388c2e8ed3c8b135731159a809a88f0681385a4ff70c145a0e690a2d57e

# Utilisation de l'outil psql installé depuis l'hôte
$ psql -U postgres -d postgres -h localhost -p 54320
psql (9.4.4, server 10.0)
WARNING: psql major version 9.4, server major version 10.0.
         Some psql features might not work.
Type "help" for help.

postgres=# select 1;
 ?column?
----------
        1
(1 row)

postgres=#

# Créer un deuxième conteneur Postgres avec une redirection du port 5432 vers le 54321 de l'hôte
$ docker run --name postgres54321 -p 54321:5432 -d postgres
77457b4a34d35409e556cf1c1b8366cf8e5624874b674e418119bfe57f93b5a7

# Utilisation de l'outil psql depuis l'hôte
$ psql -U postgres -d postgres -h localhost -p 54321
...
```

@mickaelbaron

Docker : aller plus loin

- Présentation rapide de Docker et de ses commandes
- À approfondir
  - La création des conteneurs
  - La création des images
  - La création des sous-réseaux pour faire communiquer des conteneurs
  - Le déport d'affichage X11
- À découvrir
  - La composition de conteneurs avec **docker-compose**
  - La gestion de conteneurs distants **docker-machine**
  - L'orchestration de conteneurs avec docker Swarm et Kubernetes

@mickaelbaron

Docker : à quoi cela peut me servir au laboratoire

- Pour configurer son poste de travail
  - En mode développement (triple Stores, bases de données, Hadoop, ROS, NoSQL…)
- Pour une publication
  - Fournir une image Docker de son expérimentation
  - Lien vers l'image sur la publication
- Pour un prototype logiciel => OntoWebStudio
  - Intégration de plusieurs composants (serveur web, base de données)
  - Données intégrées dans l'image
- Déployer des outils pour la recherche
  - Faciliter le déploiement et la mise à jour
  - Sharelatex, Redmine, Wekan, Git et bientôt Rocket.chat (Slack clone)

@mickaelbaron

Docker : les inconvénients

- Eco-système qui évolue rapidement (API Deprecated)
- Ne pas être « allergique » à la ligne de commande
- Nécessite d'être root dans le conteneur
- Problème de sécurité
- Déport d'affichage (application graphique)
- Limité aux systèmes Linux
- **Docker4Mac** et **Docker4Windows** pas un vrai Docker
  - Problème de redirection de port
  - Problème de performance sur les volumes

@mickaelbaron

Docker : tips

- Pour supprimer tous les conteneurs arrêtés
  - `docker rm $(docker ps -a -q)`
- Pour supprimer les images qui n'ont pas de nom
  - `docker rmi $(docker images -f "dangling=true" -q)`
- Connaître l'IP d'un conteneur
  - `docker inspect -f '{{.NetworkSettings.IPAddress }}' mycont`
- S'attacher à un conteneur en cours d'exécution
  - `docker attach mycont <=> docker exec -it mycont /bin/bash`
  - Pour se détache CTRL+p CTRL+q
- Dans Dockerfile, pour **run** utiliser un « exec form ["..",".."] » car toutes les images n'ont pas forcément de shell
- Liens
  - *https://www.ctl.io/developers/blog/post/15-quick-docker-tips*

@mickaelbaron