

DOCKER FEEDBACK

Mathilde Boissel

18/07/2023

FORMATION

La formation **Docker** (Mai 2023) réalisé via le Réseau Min2Rien (CNRS), animé par Benjamin LECHA & Mickaël MASQUELIN.

Le materiel (slides) sont disponible ici

`\\egid-partage2\BIBS\Biostatistics\Docker_2023`

Testé sous Docker version 20.10.12.

Dans ce feedback, nous allons voir

Motivations & introduction / Vocabulaire & Commandes

DockerHub / Persistance / Réseaux / Dockerfile & Mots clés / Build

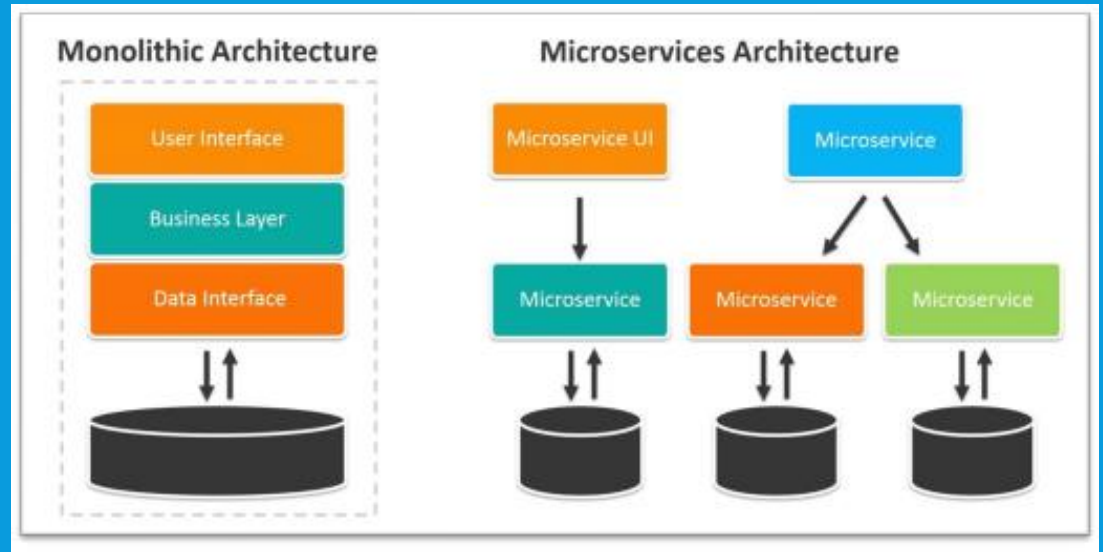
Biostat's infra avec Docker

Bonnes pratiques

MOTIVATIONS

Architecture monolithique vs microservices nécessitant

- Agilité
- Flexibilité
- Résilience
- Scalabilité

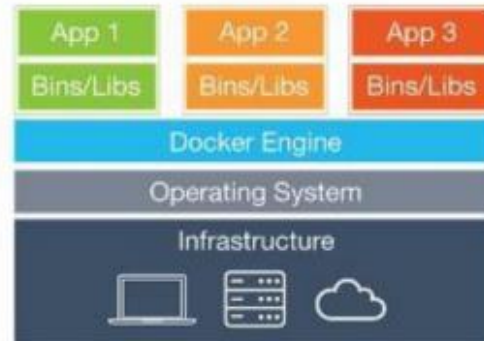


INTRODUCTION

hyperviseurs vs. emumateurs (container)



Virtual Machines



Containers

INTRODUCTION

Ingrédient pour arriver à Docker

- chroot
Modifier le répertoire racine pour le processus en cours d'exécution et ses enfants
- namespaces linux : Ce que l'on peut voir
Permet de segmenter les processus
user namespaces : UID différent de root
- cgroups (control groups) : Combien on peut utiliser
Definition de frontière, intégré dans le noyau linux

= A star is born : **Docker**

INTRODUCTION DOCKER

=> “separations of concerns”, “shipping software from A to B, reliably and automatically”

=> Personnes responsables du développement ! = Personnes responsables de la livraison

=> Docker - **CLI** = outil en ligne de commande

Pour l'installation de l'outil Docker confer pdf [Min2Rien](#)

Pour une installation sur nos servers confer [gitlab-wiki](#)

Pour tester : `docker run hello-world`

VOCABULAIRE

Docker, c'est la technologie de conteneurisation.

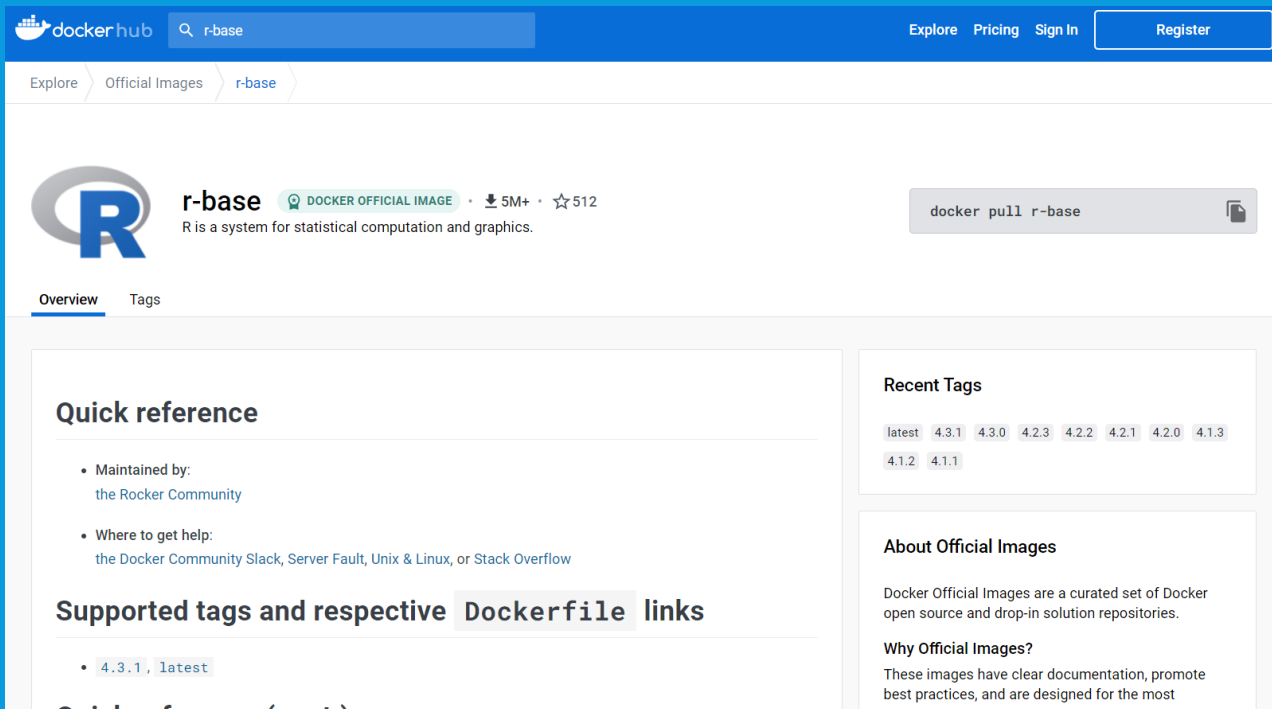
Une **image**, c'est une représentation statique de l'application ou du service, de leur configuration et de leurs dépendances.

Pour exécuter l'application ou le service, l'image de l'application est instanciée (`run`), créant ainsi un **conteneur** à exécuter sur l'hôte Docker.

Les développeurs doivent stocker les images dans un **registre**, comparable à une bibliothèque d'image. Docker gère un registre public via Docker Hub :

<https://hub.docker.com/>

DOCKERHUB




The screenshot shows the Docker Hub interface for the 'r-base' image. At the top, there's a navigation bar with the Docker Hub logo, a search bar containing 'r-base', and links for 'Explore', 'Pricing', 'Sign In', and a 'Register' button. Below the navigation bar, the breadcrumb trail shows 'Explore' > 'Official Images' > 'r-base'. The main content area features the 'r-base' logo, a 'DOCKER OFFICIAL IMAGE' badge, download statistics ('5M+' and '512 stars'), and a description: 'R is a system for statistical computation and graphics.' To the right of this information is a button with the command 'docker pull r-base' and a download icon. Below the main header, there are tabs for 'Overview' (selected) and 'Tags'. The 'Overview' tab contains a 'Quick reference' section with links to 'Maintained by: the Rocker Community' and 'Where to get help: the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow'. It also has a section for 'Supported tags and respective Dockerfile links' showing '4.3.1' and 'latest'. On the right side, there's a 'Recent Tags' section listing various version tags like 'latest', '4.3.1', '4.3.0', etc. Below that is an 'About Official Images' section explaining that these are curated, open-source, and drop-in solutions. The bottom of the page shows the start of a 'Quick summary (cont)' section.

dockerhub

Explore Pricing Sign In Register

Explore Official Images r-base

 **r-base** DOCKER OFFICIAL IMAGE · 5M+ · 512
R is a system for statistical computation and graphics.

docker pull r-base

Overview Tags

Quick reference

- Maintained by:
the Rocker Community
- Where to get help:
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

- 4.3.1, latest

Recent Tags

latest 4.3.1 4.3.0 4.2.3 4.2.2 4.2.1 4.2.0 4.1.3
4.1.2 4.1.1

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

Quick summary (cont)

Par défaut quand on contacte le registre, c'est la version latest de l'image qui est ciblée, sinon on peut préciser `image:version`.

COMMANDES

```
docker run -it r-base  
docker run -it ubuntu bash
```

docker : nom du logiciel
run : la commande
-i : interactif
-t : affichage du nom
ubuntu : le nom de l'image
bash : la commande

```
docker --help : pour avoir la doc general  
docker run --help : pour avoir la doc de la commande run  
docker ps -a : pour voir la liste des containers (-a pour all, qui tournent ou exited)
```

PERSISTANCE VOLUMES ET DETACH

- 1) `docker run -it ubuntu bash`
- 2) `touch bonjour.txt` command pour création d'un fichier
- 3) `apt update` et `apt install curl` pour installation d'une library linux
- 4) `exit` pour quitter le terminal (et donc le container) une fois quitté : tout l'environnement a disparu le fichier bonjour et l'installation de curl est à refaire!

Raison : pas de montage de volume (stockage persistant) et une fois quitté il y a un `destroy` du container.

Pour éviter le `destroy` automatique, l'option `-d` ou `--detach` permet de laisser tourner le container en arrière plan (on peut rentrer dedans en interactif, quitter et revenir).

Param Detach : `docker run -it -d -name myubuntu_mb ubuntu bash`

Montage de volume :

`docker run -it -v /Isiprodl/user/any_user:/home inside ubuntu bash`

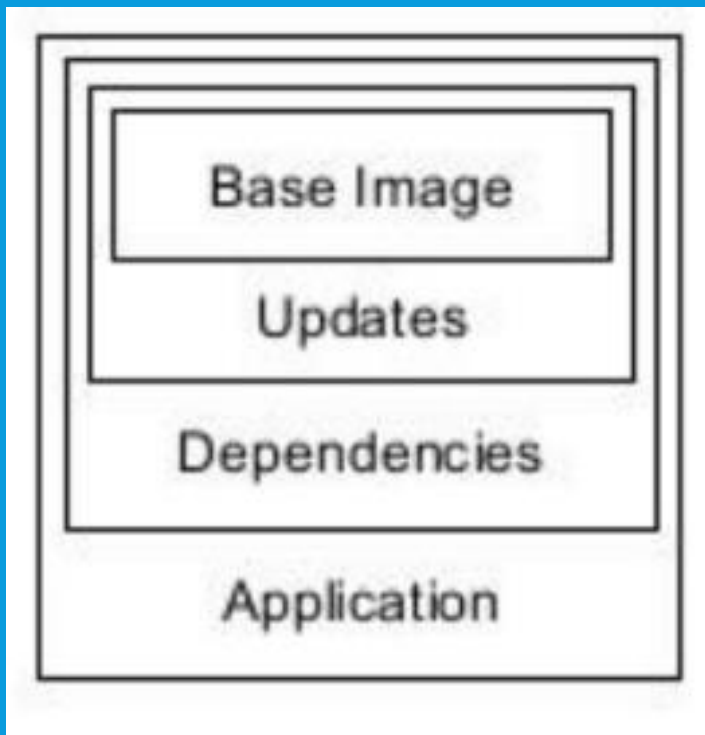
Montage avec droit "read only" (ro) : `docker run -it -v ./:/home:ro ubuntu bash`

RÉSEAUX

Chaque conteneur est connecté à un réseau privé virtuel nommé "bridge".
Chaque sous-réseau dans Docker peut communiquer avec l'adresse IP de la machine physique grâce à un routeur virtuel effectuant du NAT.
Tous les conteneurs situés sur le même réseau virtuel peuvent discuter entre eux (sans utiliser l'option `-p`) : confer `docker network`

Aussi pour rendre vos ports Docker accessibles par les services du monde extérieur, on peut exposé (ouvrir) notre docker avec l'instruction `EXPOSE` dans le Dockerfile ou le paramètre `-p` (publishing ports) dans la commande `run`.
Exemple `docker run -d -p 8080:80 nginx` avec un port de sa machine (8080) vers un port du conteneur (80).

DOCKERFILE



= La recette de cuisine utilisée
par le développeur pour la
création d'images
(puis le transporter de son appli)

FYI : reverse engineering possible mais approximatif...

<https://github.com/wagoodman/dive> Sert à explorer les couches du Dockerfile (va plus loin que `docker history`)

<https://github.com/mrhavens/Dedockify> Permet de reconstruire le Dockerfile d'origine partiellement (ne redonnera pas les fichiers issus pour un COPY)

DOCKERFILE MOTS CLÉS

FROM : Pour spécifier l'image de base du container (une boite noire à prendre comme telle)

RUN : equivalent de lancer des commandes dans le terminal
Il faut connaitre l'environnement pour connaitre le langage de base

COPY : copie colle dossier ou fichier de la machine vers l'image

ADD : COPY ou Téléchargement de fichiers externes et Décompresse automatiquement une archive à l'emplacement donné

ENV : variable d'environnement connu dans toute l'image.

Exemple l'heure : `ENV TZ="Europe/Paris"` (TZ pour time zone)

LABEL : juste pour donner des étiquettes, des petites notes.

Retrouvé via `docker inspect`, commande pour afficher les infos

CMD : commande executée une seule fois à la fin

DOCKERFILE EXAMPLE TOKYO

```
➤ ~ cat Dockerfile
```

```
FROM ubuntu
```

```
RUN apt update && apt install -y tzdata
```

```
ENV TZ="Asia/Tokyo"
```

```
RUN dpkg-reconfigure tzdata
```

```
CMD ["date"]
```

Image source Ubuntu

Installation des maj et du
paquet tzdata

Changement d'une variable
d'environnement

Applique le changement
d'environnement

Lance la commande 'date' au
démarrage du conteneur

BUILD

```
docker build --tag tokyo .
```

`docker` : Nom du logiciel

`build` : Nom de la commande

`-t / --tag` : Ajout un tag pour l'identifier plus facilement

`tokyo` : Nom du tag et donc de l'image (version avec `tokyo:`, défaut `latest`)

`.` : Contexte d'exécution, en l'occurrence mon dossier courant où se trouve le fichier

`Dockerfile` à utiliser par défaut.

Si le nom est différent, le build ne se fera pas sans préciser le nom du fichier explicitement.

```
docker images | grep tokyo
```

 pour retrouver l'existence de l'image localement

```
docker run tokyo
```

 démarre le conteneur basé sur l'image cuistruite "tokyo"

Astuces (quick and dirty patch) : `image > docker commit > new image > docker save > load it !`

BIOSTAT'S DOCKERFILE

```
1 from rocker/rstudio:4.2.1
2
3 ENV DEBIAN_FRONTEND noninteractive
4
5 # ENV R_HOME=/usr/local/lib/R
6 # ENV CRAN=https://cran.r-project.org
7 ENV LANG=en_GB.UTF-8
8 ENV TZ=Etc/UTC
9
10 # linux libs
11 RUN apt-get update && apt-get upgrade -y
12
13 # ssh connection (needed to git push)
14 RUN apt-get update && apt-get install -y openssh-server
15
16 # audit from https://github.com/rocker-org/docker-versioned/blob/main/assets/install_r_packages
17 # Library linux needed for R package
18 RUN apt-get update && apt-get install -y --no-install-recommends \
19     cmake \
20     libudunits2-dev \
21     libcurl4-openssl-dev \
22     libxml2-dev \
23     libfontconfig1-dev \
24     libssl-dev \
25     libsasl2-dev \
26     libharfbuzz-dev \
27     libfribidi-dev \
28     libfreetype6-dev \
29     libpng-dev \
30     libtiff5-dev \
31     libjpeg-dev \
32     libv8-dev \
33     libxt6 \
34     libgdal-dev \
35     libgmp-dev \
36     librsvg2-dev \
37     libfftw3-dev
```

1 - rocker image

2 - ENV

3 - Linux lib

4 - Linux lib needed for R pkg after

```
5 # install basic pkg R
6
7 # RUN R -e "install.packages(c('renv'), dependencies=TRUE)"
8 RUN R -e "install.packages(c('udunits2', 'units', 'devtools', 'usethis', 'here', 'renv', 'ragg',
9
10     'add_users', with the wanted function to copy before...
11 COPY add_userid.sh .
12 RUN rm /bin/sh && ln -s /bin/bash /bin/sh
13 # RUN source /add_userid.sh
14 # RUN cat add_userid.sh # check if the function is known
15 RUN bash -c 'source /add_userid.sh && add_userid mboissel 2146'
16 RUN bash -c 'source /add_userid.sh && add_userid lining 2145'
17 RUN bash -c 'source /add_userid.sh && add_userid adiallo 2144'
18 RUN bash -c 'source /add_userid.sh && add_userid istanchev 2149'
19 RUN bash -c 'source /add_userid.sh && add_userid akhamis 2150'
20
21 # install other tools
22 copy code from https://github.com/umr1283/docker-versioned/blob/main/02-dockerfiles/umr1283_4.2.
23
24 ENV PANDOC_VERSION=2.19.2
25 ENV BCFTOOLS_VERSION=1.15.1
26 ENV QUARTO_VERSION=1.3.34
27 ENV ODBC_VERSION=8.0.27
28 ENV S6_VERSION=v2.2.0.3
29
30 COPY assets /docker_scripts
31
32 RUN chmod --recursive +x /docker_scripts
33 RUN /docker_scripts/install_libs.sh
34 RUN /docker_scripts/install_pandoc.sh
35 # RUN /docker_scripts/install_odbc.sh
36 RUN /docker_scripts/install_s6v2.sh
37 RUN /docker_scripts/install_python.sh
38 RUN /docker_scripts/install_crossmap.sh
39 RUN /docker_scripts/install_bcftools.sh
40 RUN /docker_scripts/install_vcftools.sh
41 RUN /docker_scripts/install_qtlttools.sh
42 RUN /docker_scripts/install_r_packages.sh
43 RUN /docker_scripts/install_quarto.sh
```

5 - R pkg

6 - add user

7 - Install other tools (pandoc, bcftools, python...)

detail in [lab's wiki](#)

BIOSTAT'S RUN

```
docker run \  
  --name "rstudio421_intern" \  
  --detach \  
  --rm \  
  -p 8421:8787 \  
  --env "RENV_PATHS_CACHE=/renv_cache" \  
  --volume /tmp:/tmp \  
  --volume /Isiprod1/datatmp/dockertmp/renv_pkgs_cache:/renv_cache \  
  --volume /Isiprod1/project/SB_istanchev:/Isiprod1/project/SB_istanchev \  
  umr1283/rstudio:4.2.1  
# secondly here add the user in this container running  
sudo docker exec rstudio421_intern /Isiprod1/project/SB_istanchev/scripts/add_user.sh istanchev 2149  
## go on `http://docker15.egid.local:8421/` to connect as istanchev (mdp istanchev)
```

BIOSTAT'S DEV CONTAINERS (1/3)

***Avec un dockerfile
spécifique pour
chaque projet,
build le container
du projet en live
et sur mesure !***

```
devcontainer.json
1 {
2   "name": "UMR1283",
3   "runArgs": ["--name", "${localEnv:USER}--${localWorkspaceFolderBasename}--devcontainer"],
4   "remoteUser": "${localEnv:USER}",
5   "build": {
6     "dockerfile": "Dockerfile",
7     "args": {
8       "IMAGE": "umr1283/umr1283:4.1.3",
9       "USERNAME": "${localEnv:USER}"
10    }
11  },
12  "containerEnv": {
13    "RENV_PATHS_CACHE": "/renv_cache",
14    "PKG_CONFIG_PATH": "/usr/lib/x86_64-linux-gnu/pkgconfig"
15  },
16  "workspaceMount": "source=${localWorkspaceFolder},target=/Isiprod1/project/${localWorkspaceFolderBasename},type=bind",
17  "workspaceFolder": "/Isiprod1/project/${localWorkspaceFolderBasename}",
18  "mounts": [
19    "source=/Isiprod1/user,target=/home,type=bind,consistency=cached",
20    "source=/Isiprod1,target=/Isiprod1,type=bind,consistency=cached",
21    "source=/Isiprod1/datatmp/dockertmp/renv_pkgs_cache,target=/renv_cache,type=bind,consistency=cached",
22    "source=/etc/localtime,target=/etc/localtime,type=bind,consistency=cached"
23  ],
24  "customizations": {
25    "vscode": {
26      "settings": {
27        "editor.tabSize": 2,
28        "files.trimFinalNewlines": true,
29        "files.insertFinalNewline": true,
30        "files.watcherExclude": {
31          "**/.Rproj": true,
32          "**/_cache/**": true
33        }
34      }
35    }
36  }
37 }
```

[Clic for wiki procedure](#)

BIOSTAT'S DEV CONTAINERS (2/3)

/media/project/SB_mboissel/.devcontainer/

Name	Size
..	
devcontainer.json	2
Dockerfile	1


Avec un dockerfile spécifique pour chaque projet, build le container du projet en live et sur mesure !

Dockerfile

```
1 ARG IMAGE
2 FROM ${IMAGE}
3
4 RUN apt-get update && apt-get install -y openssh-client sudo
5 RUN apt-get install pkg-config
6 RUN apt-get install jags
7
8 ARG USERNAME=umr1283
9 ARG USERID=1000
10 RUN if [[ -d /home/${USERNAME} ]]; then CH="--no-create-home"; else CH="--create-home"; fi \
11     && useradd \
12         ${CH} \
13         --no-user-group \
14         --shell /bin/bash \
15         --uid ${USERID} \
16         --gid staff \
17         --groups staff \
18         ${USERNAME} \
19         && usermod -aG sudo ${USERNAME} \
20         && echo "${USERNAME} ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers
21
22 RUN Rscript -e 'pak::pkg_install(c("languageserver", "httpgd"))'
23
24 USER ${USERNAME}
25
```


BIOSTAT'S DEV CONTAINERS (3/3)

Extension: Dev Containers



Dev Containers

v0.299.0 Preview


Microsoft  microsoft.com | 19,199,197 | ★★★★★ (44)

Open any folder or repository inside a Docker container and take advantage of Visual Studio Code's full feature set.

Disable

Uninstall

Switch to Pre-Release Version



This extension is enabled globally.

DETAILS

FEATURE CONTRIBUTIONS

RUNTIME STATUS

Visual Studio Code Dev Containers

The **Dev Containers** extension lets you use a [Docker container](#) as a full-featured development environment. Whether you deploy to containers or not, containers make a great development environment because you can:

- Develop with a consistent, easily reproducible toolchain on the same operating system you deploy to.
- Quickly swap between different, separate development environments and safely make updates without worrying about impacting your local machine.
- Make it easy for new team members / contributors to get up and running in a consistent development environment.
- Try out new technologies or clone a copy of a code base without impacting your local setup.

The extension starts (or attaches to) a development container running a well defined tool and runtime stack. Workspace files can be mounted into the container from the local file system, or copied or cloned into it once the container is running. Extensions are installed and run inside the container where they have full access to the tools, platform, and file system.

Categories

Other

Extension Resources

[Marketplace](#)
[Repository](#)
[License](#)
[Microsoft](#)

More Info

Published

5/2/2019, 20:31:37

Last released

7/10/2023, 15:07:29

Last updated

7/7/2023, 09:12:13

Identifier

ms-vscode-remote-remote-containers

BONNES PRATIQUES

*(ce qu'on fait tous,
si possible...)*

- 1) Utiliser des conteneurs non privilégiés (non root). Voir commande `adduser`
- 2) Pour chaque application ou service, il est préférable de créer un nouveau réseau virtuel (monitoring +, firewall +)
- 3) Placez toujours les couches susceptibles de changer le plus bas possible dans le Dockerfile (cache +)
- 4) Entrypoint vs. CMD : CMD facilement contournable car peut être remplacé à l'appel, alors que ENTRYPOINT nécessite le paramètre `--entrypoint` à l'appel. (Avantage ENTRYPOINT et CMD sont combinables)
- 5) Syntaxe array ou string : Préférez la syntaxe array
(CMD ["unicorn", "-w", "4", "-k", "unicorn.workers.UvicornWorker", "main:app"])
car dans l'autre cas `--string` (CMD "unicorn -w 4 -k unicorn.workers.UvicornWorker main:app"),
le processus est appelé avec un shell. Il y aura une mauvaise gestion des signaux Unix (ex : SIGTERM)
- 6) Limiter la capacité d'utilisation `--cpus=2 -m 512m` (Limite à 2 coeurs (CPU) + 512 Mo de RAM)
- 7) Un service = un conteneur
- 8) Ne pas stocker de secret dans son image ... ENV DATABASE_PASSWORD "4Nd0u1113!"
utiliser ARG DATABASE_PASSWORD et au moment du build
`--build-arg "DATABASE_PASSWORD=4Nd0u1113!"` ou voir `--secret`



*9) Encore quelques autres
astuces et bonnes
pratiques abordées dans
la présentation (voir pdf
complet de minzrien)*

MERCI POUR VOTRE ATTENTION !

