Devops

Docker

2025/2026

- Présentation disponible à l'adresse: https://gounthar.github.io/cours-devops-docker/main
- Version PDF de la présentation : Cliquez ici
- Contenu sous licence Creative Commons Attribution 4.0 International License
- Code source de la présentation: https://github.com/gounthar/cours-devops-docker

Comment utiliser cette présentation ?

- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
  - Gauche/Droite: changer de chapitre
  - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "o" (pour "**Overview**")

Bonjour !

```
poddingue@my-machine:~$ docker
```

Bruno VERACHTEN

CloudBees.

- Sr Developer Relations chez CloudBees pour le projet Jenkins 👨🏻‍⚖️
- Me contacter :
  - gounthar@gmail.com
  - gounthar
  - https://bruno.verachten.fr/
  - Bruno Verachten
  - @poddingue

Et vous ?

A propos du cours

- Première itération d'une découverte Docker dans le cadre du DevOps
- Contenu entièrement libre et open-source
- Méchamment basé sur le travail de Damien Duportal et Amaury Willemant
  - N'hésitez pas ouvrir des Pull Request si vous voyez des améliorations ou problèmes: sur cette page (😉 wink wink)

Calendrier

🚧 Work in progress… 🚧

- 16 septembre après-midi
- 23 septembre après-midi
- 30 septembre après-midi
- …

📊 Évaluation

- Pourquoi ? s'assurer que vous avez acquis un minimum de concepts
- Quoi ? Sans doute une note sur 20, basée sur une liste de critères exhaustifs
- Comment ? Un projet Gitlab (fort probable) ou GitHub (peu probable) à me rendre (timing à déterminer ensemble)

Plan

- Intro
- Base
- Containers
- Images
- Fichiers, nommage, inspect
- Volumes
- Réseaux
- Docker Compose
- Bonus

La suite: vers la droite ➡️

Docker

"La Base"

Pourquoi ?

🤔 Quel est le problème ?

Pourquoi commencer par un problème ?

🤔 Commençons plutôt par une définition:

Docker c'est …

Pourquoi commencer par un problème ?

🤔 Définition quelque peu datée (2014):

> *Docker is …*

Pourquoi commencer par un problème ?

🤔 Définition quelque peu datée (2014):

> *Docker is a toolset for Linux containers designed to 'build, ship and run' distributed applications.*

https://www.infoq.com/articles/docker-future/

Linux containers ?

🤔 Nous voilà bien…

C'est quoi un container?

Linux containers ?

🤔 Nous voilà bien…

C'est quoi un container?

Vous voulez la version enfant de 5 ans? Je ne crois pas…

Docker est vieux

10 ans déjà…

Docker est vieux

Mais moi aussi…

Docker Pirates ARMed with explosive stuff

Roaming the seven seas in search for golden container plunder.

HOME
GETTING STARTED
DOWNLOADS
FAQ
COMMUNITY
ABOUT US
CREW
DONATE

© 2020 Hypriot

Legal Notice

Edit this blog on GitHub

Docker on Raspberry Pi Workshop in Brussels

Thu, Mar 17, 2016

A couple of weeks ago we have been invited by the fine folks over at the Docker User Group in Brussels to help conduct a workshop. And of course this workshop was about Docker. Still it was not your ordinary Docker workshop.

It was special because instead of being a workshop about Docker on big servers it was about Docker on really small ARM devices. The very same devices that power the upcoming IoT revolution.

Turns out it is really amazing what you can do with Docker on those tiny machines.

Docker on arm

Aucune raison que ça soit réservé aux puissants! La révolution vaincra!

LinuxGizmos.com

All News | Boards | Chips | Devices | Software | LinuxDevices.com Archive | About | Contact | Subscribe

Open source ResinOS adds Docker to ARM/Linux boards

Oct 17, 2016 — by Eric Brown 4,449 views

Resin.io has spun off the Yocto-based OS behind its Resin.io IoT framework as a ResinOS 2.0 distro for running Docker containers on Linux IoT devices.

Resin.io, the company behind the Linux/Javascript-based Resin.io IoT framework for deploying applications as Docker containers, began spinning off the Linux OS behind the framework as an open source project over a year ago. The open source ResinOS is now publicly available on its own in a stable 2.0.0-beta.1 version, letting other developers create their own Docker-based IoT networks. ResinOS can run on 20 different mostly ARM-based embedded Linux platforms including the Raspberry Pi, BeagleBone, and Odroid-C1, enabling secure rollouts of updated applications over a heterogeneous network.

ResinOS 2.0 architecture
(click image to enlarge)

When enterprise-level CIOs are asked to integrate embedded devices into their networks, their first question is usually "Can they run Docker?" The answer is probably not, especially if they run on ARM processors.

Follow LinuxGizmos:

* get email updates *

Search LinuxGizmos:

Search

Search LinuxGizmos.com + LinuxDevices Archive:

ENHANCED BY Google Search

LinuxGizmos Sponsor ads:

(advertise here)

Top 10 trending posts...

» Libre Computer showcases low-cost SBC with PoE support
» ASRock Industrial's present 4x4 BOX 7040 Series mini PCs
» (Updated) The ZimaBlade is an upcoming low-cost x86 personal server
» GOWIN & Andes Technologies collaborate and reveal 22nm SoC FPGA
» Espressif Systems presents ESP32-S3-BOX-3 AIoT kit
» Milk-V Duo is a $9.00 RISC-V tiny embedded computer
» SATA HATs support up to four drives on Raspberry Pi 4 or Rock Pi 4
» Sipeed to launch RISC-V based Lichee Cluster 4A
» New SBC powered by Allwinner T507-H processor
» Orange Pi introduces a new Rockchip based Computer Module

LinuxGizmos Sponsor ads:

(advertise here)

Follow LinuxGizmos or subscribe to our posts:

Docker on *

Linux everywhere… And then Docker to follow.

```
target "debian_jdk11" {
  dockerfile = "debian/Dockerfile"
  context = "."
  args = {
    JAVA_MAJOR_VERSION = "11"
    version = "${PARENT_IMAGE_VERSION}"
  }
  tags = [
    equal(ON_TAG, "true") ? "${REGISTRY}/${JENKINS_REPO}:${PARENT_IMAGE_VERSION}": "",
    equal(ON_TAG, "true") ? "${REGISTRY}/${JENKINS_REPO}:${PARENT_IMAGE_VERSION}-jdk11": "",
    "${REGISTRY}/${JENKINS_REPO}:jdk11",
    "${REGISTRY}/${JENKINS_REPO}:latest",
    "${REGISTRY}/${JENKINS_REPO}:latest-jdk11",
  ]
  platforms = ["linux/amd64", "linux/arm64", "linux/arm/v7", "linux/s390x", "linux/ppc64le"]
}
```

On n’avait pas parlé d’un problème?

Intégrateur Développeur Testeuse

DEV

On n’avait pas parlé d’un problème?

On n’avait pas parlé d’un problème?

On n’avait pas parlé d’un problème?

On n’avait pas parlé d’un problème?

On n’avait pas parlé d’un problème?

Histoire vraie.

```
Hey salut !
```

On n’avait pas parlé d’un problème?

Histoire vraie.

```
Hey salut !
```

cc

On n’avait pas parlé d’un problème?

Histoire vraie.

Hey salut !

cc

Tu peux mettre à jour les packages système steup ? j’ai un truc à tester.

On n’avait pas parlé d’un problème?

Histoire vraie.

```
Hey salut !
```

cc

```
Tu peux mettre à jour les
packages système steup ? j’ai
un truc à tester.
```

nan

On n’avait pas parlé d’un problème?

Histoire vraie.

Hey salut !

cc

Tu peux mettre à jour les packages système steup ? j’ai un truc à tester.

nan

???

On n’avait pas parlé d’un problème?

Histoire vraie.

Hey salut !

cc

Tu peux mettre à jour les packages système steup ? j’ai un truc à tester.

nan

???

pas standard dsl

On n’avait pas parlé d’un problème?

Histoire vraie.

Hey salut !

cc

Tu peux mettre à jour les packages système steup ? j’ai un truc à tester.

nan

???

pas standard dsl

On n’avait pas parlé d’un problème?

Static Website	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Production Cluster	Public cloud	Developer’s Laptop	Customer Servers

Source: https://blog.docker.com/2013/08/paas-present-and-future/

Problème de temps **exponentiel**

Déjà vu ?

L'IT n'est pas la seule industrie à résoudre des problèmes…

Solution: Le conteneur intermodal

"Separation of Concerns"

Comment ça marche ?

"Virtualisation **Légère**"

Comment ça marche ?

Virtualisation

Comment ça marche ?

Virtualisation

Comment ça marche ?

"Virtualisation **Légère**"

- Légère, vraiment?

Raspberry Pi DockerCon Challenge: We Have a Winner!

VICTOR COISNE
Oct 20 2015

At the end of DockerCon 2015, Dieter Reuter from Hypriot presented a demo running 500 Docker containers on a Raspberry Pi 2 device but he knew that number could be at least doubled.

TL;DR Dieter was right.

Post Tags

- dockercon
- DockerCon Europe
- raspberry pi

Categories

- All
- Products
- Community
- Engineering
- Company

Légère, vraiment!

http://web.archive.org/web/20200810061020/https://www.docker.com/blog/raspberry-pi-dockercon-challenge-winner/

A big congratulations to Damien Duportal, Nicolas de Loof and Yoann Dubreuil on running 2500 web servers in Docker containers on a single Raspberry Pi 2!

Damien, Nicolas and Yoann each win a complimentary pass to DockerCon EU 2015 and speaking slot during the conference to demo how they accomplished this.

#RpiDocker 2740 web servers running on a #Rpi, could have more. But using a patched docker daemon with a hack that isn't a valuable fix. — Nicolas De loof (@ndeloof) October 13, 2015

Aside from the above issues (and there are more caveats in the documentation), my installation is a pretty faithful reproduction of ESXi on a server or home lab. However, ESXi itself uses about 1.3GB of RAM, leaving only around 6.5GB usable. This could support perhaps 4-5 small VMs, but would be much more useful for running containerised applications and eliminating the individual VM overhead.

Légère, vraiment!

Proof of Concept

https://www.architecting.it/blog/esxi-on-raspberry-pi/

Remember that Flings are just proofs of concept and not always aimed at final production. VMware suggests a range of hardware options for ARM-based workloads and this is more likely where we will see the initial development of ESXi on ARM. The challenge for end users here is to determine whether the price/performance/power/cooling ratio works better on ARM than x86. Of course applications also need to be available, but it's not hard to find ARM-compiled versions of most open source software and operating systems.

Conteneur != VM

- Idée erronée mais citée trop fréquemment !

Conteneur != VM

Machine Physique

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

Conteneur != VM

"Separation of concerns": 1 "tâche" par conteneur

VM

Containers

VMs & Conteneurs

Non exclusifs mutuellement

🤡 Docker, c'est pas un peu une VM quand même?

🤡 Docker, c'est pas un peu une VM quand même?

- Docker sur Windows pour exécuter des conteneurs Linux :

- Docker sur Linux pour exécuter des conteneurs Linux :

Cas d’usage

Le bac à sable

Cas d’usage

Le bac à sable

- Un environnement tout propre tout neuf !

Cas d’usage

Le bac à sable

- Un environnement tout propre tout neuf !
- Le poste de travail n’est pas impacté

Cas d’usage

Le bac à sable

- Un environnement tout propre tout neuf !
- Le poste de travail n’est pas impacté
- La configuration reste également isolée sans effet sur le poste de travail

Cas d’usage

La machine de dév

Cas d’usage

La machine de dév

- Avoir un environnement reproductible pour rendre homogène le développement et les tests.

Cas d'usage

La machine de dév

Cas d'usage

La machine de dév

Cas d’usage

Déploiement en production

Cas d’usage

Déploiement en production

- Avec un container, si ca fonctionne en local, ca fonctionne en prod !

Cas d’usage

Outillage jetable

Cas d’usage

Outillage jetable

- On instancie des applications sans les installer

Cas d’usage

Outillage jetable

- On instancie des applications sans les installer
- On crée le container, on l’utilise puis on le jette

Beau boulot! 🤗

On a la base, remplissons là maintenant de containers!

La suite: vers la droite ➡️

Préparer votre environnement de travail

Particularités du réseau

Utiliser Docker derrière un proxy ou un cache universitaire

Pourquoi ? – Le Problème

Lorsque nous sommes sur le réseau de l'université, Docker a du mal à télécharger directement les images depuis Docker Hub à cause des restrictions réseau. Nous sortons tous avec la même adresse IP, et en peu de temps, elle sera blacklistée pour un moment, ce qui nous pénalisera tous. Pour résoudre ce problème, l'université met à disposition un serveur de cache local.

1. Concepts de base Docker

Avant toute configuration, introduisons les notions essentielles, que nous verrons en détail par la suite :

```
# Que sont les images Docker ?
docker pull hello-world
docker run hello-world

# Comment les images Docker sont nommées ?
[registry/][username/]image-name[:tag]

# Exemples :
library/ubuntu:22.04   # Image officielle (library est implicite)
ubuntu:22.04           # Identique à ci-dessus
user/mon-app:latest    # Image personnalisée
```

2. Le concept de cache de registre

Au lieu de récupérer les images directement depuis Docker Hub, nous utilisons le cache fourni par l'université.

2. Le concept de cache de registre

Au lieu de récupérer les images directement depuis Docker Hub, nous utilisons le cache fourni par l'université.

Avant (problème) :

```
docker pull ubuntu:22.04   # Échec sur le réseau universitaire, ou succès temporaire avant d'être blacklistés
```

2. Le concept de cache de registre

Au lieu de récupérer les images directement depuis Docker Hub, nous utilisons le cache fourni par l'université.

Avant (problème) :

```
docker pull ubuntu:22.04   # Échec sur le réseau universitaire, ou succès temporaire avant d'être blacklistés
```

Après (solution) :

```
docker pull cache-ili.univ-artois.fr:80/proxy_cache/library/ubuntu:22.04
```

3. Approche simplifiée de configuration

Option A : utiliser un **registry mirror**.

3. Approche simplifiée de configuration

Option A : utiliser un **registry mirror**.

Modifier `/etc/docker/daemon.json` :

```
{
  "registry-mirrors": ["http://cache-ili.univ-artois.fr:80/proxy_cache/"],
  "insecure-registries": ["cache-ili.univ-artois.fr:80"]
}
```

3. Approche simplifiée de configuration

Option A : utiliser un **registry mirror**.

Modifier `/etc/docker/daemon.json` :

```
{
  "registry-mirrors": ["http://cache-ili.univ-artois.fr:80/proxy_cache/"],
  "insecure-registries": ["cache-ili.univ-artois.fr:80"]
}
```

Désormais, les commandes Docker standards fonctionnent :

```
docker pull ubuntu:22.04   # Utilise automatiquement le cache !
docker run gradle          # Fonctionne de manière transparente
```

4. Script étape par étape

Étape 1 – Comprendre le problème

- Docker Hub nous voit tous avec la même adresse IP, qui mène au blacklist
- Nous devons utiliser le serveur de cache local.

4. Script étape par étape

Étape 1 – Comprendre le problème

- Docker Hub nous voit tous avec la même adresse IP, qui mène au blacklist
- Nous devons utiliser le serveur de cache local.

Étape 2 – Configurer Docker une seule fois

```
sudo nano /etc/docker/daemon.json
```

4. Script étape par étape

Étape 1 – Comprendre le problème

- Docker Hub nous voit tous avec la même adresse IP, qui mène au blacklist
- Nous devons utiliser le serveur de cache local.

Étape 2 – Configurer Docker une seule fois

```
sudo nano /etc/docker/daemon.json
```

Ajouter :

```
{
  "registry-mirrors": ["http://cache-ili.univ-artois.fr:80/proxy_cache/"],
  "insecure-registries": ["cache-ili.univ-artois.fr:80"]
}
```

4. Script étape par étape

Étape 1 – Comprendre le problème

- Docker Hub nous voit tous avec la même adresse IP, qui mène au blacklist
- Nous devons utiliser le serveur de cache local.

Étape 2 – Configurer Docker une seule fois

```
sudo nano /etc/docker/daemon.json
```

Ajouter :

```
{
  "registry-mirrors": ["http://cache-ili.univ-artois.fr:80/proxy_cache/"],
  "insecure-registries": ["cache-ili.univ-artois.fr:80"]
}
```

Étape 3 – Redémarrer Docker

```
sudo systemctl restart docker
```

4. Script étape par étape

Étape 3 – Redémarrer Docker

```
sudo systemctl restart docker
```

Étape 4 – Utiliser Docker normalement

```
docker pull ubuntu:22.04    # Utilise automatiquement le cache
docker run python:3.9       # Fonctionne sans problème
```

4. Script étape par étape

Étape 4 – Utiliser Docker normalement

```
docker pull ubuntu:22.04    # Utilise automatiquement le cache
docker run python:3.9       # Fonctionne sans problème
```

Étape 5 – Explication visuelle

Votre client Docker → Cache universitaire → Docker Hub (si nécessaire) (cache-ili.univ-artois.fr)

Avantages : - Téléchargements plus rapides (images déjà en cache) - Fonctionne derrière le pare-feu universitaire - Pas besoin de changer vos commandes

4. Script étape par étape

Étape 5 – Explication visuelle

Votre client Docker → Cache universitaire → Docker Hub (si nécessaire) (cache-ili.univ-artois.fr)

Avantages : - Téléchargements plus rapides (images déjà en cache) - Fonctionne derrière le pare-feu universitaire - Pas besoin de changer vos commandes

Étape 6 – Problèmes courants

```
# Permission refusée ? Ajoutez votre utilisateur au groupe docker :
sudo usermod -aG docker $USER
newgrp docker   # Ou déconnectez-vous / reconnectez-vous

# Toujours un souci ? Testez le cache :
docker pull cache-ili.univ-artois.fr:80/proxy_cache/library/hello-world
```

5. Configurations avancées pour les proxys (au-delà du cache)

Parfois, vous devrez configurer un proxy HTTP/HTTPS pour que Docker télécharge les dépendances lors de la construction d'images.

5. Configurations avancées pour les proxys (au-delà du cache)

Parfois, vous devrez configurer un proxy HTTP/HTTPS pour que Docker télécharge les dépendances lors de la construction d'images.

Option 1 – Utiliser ENV dans le Dockerfile (Recommandé)

```
FROM alpine:latest

# IMPORTANT: ENV doit être APRÈS FROM et AVANT RUN
ENV https_proxy=http://cache-etu.univ-artois.fr:3128/
ENV http_proxy=http://cache-etu.univ-artois.fr:3128/
ENV HTTPS_PROXY=http://cache-etu.univ-artois.fr:3128/
ENV HTTP_PROXY=http://cache-etu.univ-artois.fr:3128/

# Votre gestionnaire de paquets utilisera désormais le proxy
RUN apk update && apk add your-package
```

⚠️ **Ordre crucial:** `FROM` → `ENV` → `RUN` (réseau)

5. Configurations avancées pour les proxys (au-delà du cache)

Option 2 – Build multi-étapes (proxy uniquement quand nécessaire)

```
# Étape de build avec proxy
FROM alpine:latest as builder
ENV https_proxy=http://cache-etu.univ-artois.fr:3128/
ENV http_proxy=http://cache-etu.univ-artois.fr:3128/
ENV HTTPS_PROXY=http://cache-etu.univ-artois.fr:3128/
ENV HTTP_PROXY=http://cache-etu.univ-artois.fr:3128/
RUN apk update && apk add build-dependencies

# Étape finale sans proxy
FROM alpine:latest
COPY --from=builder /built-artifacts /app
```

5. Configurations avancées pour les proxys (au-delà du cache)

Option 3 – Variables d'environnement passées au build

```
docker build --build-arg https_proxy=http://cache-etu.univ-artois.fr:3128/ .
```

Dockerfile correspondant :

```
ARG https_proxy
ARG http_proxy

RUN apk update && apk add your-package
```

5. Configurations avancées pour les proxys (au-delà du cache)

Option 4 – Configuration du démon Docker (système entier)

Ajouter dans `/etc/docker/daemon.json` :

```
{
  "proxies": {
    "http-proxy": "http://cache-etu.univ-artois.fr:3128",
    "https-proxy": "http://cache-etu.univ-artois.fr:3128",
    "no-proxy": "localhost,127.0.0.1"
  }
}
```

5. Configurations avancées pour les proxys (au-delà du cache)

Option 4 – Configuration du démon Docker (système entier)

Ajouter dans `/etc/docker/daemon.json` :

```
{
  "proxies": {
    "http-proxy": "http://cache-etu.univ-artois.fr:3128",
    "https-proxy": "http://cache-etu.univ-artois.fr:3128",
    "no-proxy": "localhost,127.0.0.1"
  }
}
```

Puis redémarrer Docker :

```
sudo systemctl restart docker
```

5. Configurations avancées pour les proxys (au-delà du cache) (suite)

Option 5 – Configuration spécifique à l'utilisateur

Créer ou modifier `~/.docker/config.json` :

```
{
  "proxies": {
    "default": {
      "httpProxy": "http://cache-etu.univ-artois.fr:3128",
      "httpsProxy": "http://cache-etu.univ-artois.fr:3128",
      "noProxy": "localhost,127.0.0.1"
    }
  }
}
```

6. Pourquoi ENV est préférable à RUN export

- **Persistance** : `ENV` définit la variable pour toutes les commandes `RUN` suivantes.
- **Clarté** : la dépendance au proxy est explicitement indiquée dans le Dockerfile.
- **Arguments de build** : permettent de surcharger la configuration au moment du build.

6. Pourquoi ENV est préférable à RUN export

- **Persistance** : `ENV` définit la variable pour toutes les commandes `RUN` suivantes.
- **Clarté** : la dépendance au proxy est explicitement indiquée dans le Dockerfile.
- **Arguments de build** : permettent de surcharger la configuration au moment du build.

En résumé

- Utilisez l'option **#1 (ENV dans Dockerfile)** si le proxy est toujours nécessaire.
- Utilisez l'option **#3 (build args)** si vous souhaitez plus de flexibilité.

6. Pourquoi ENV est préférable à RUN export

- **Persistance** : `ENV` définit la variable pour toutes les commandes `RUN` suivantes.
- **Clarté** : la dépendance au proxy est explicitement indiquée dans le Dockerfile.
- **Arguments de build** : permettent de surcharger la configuration au moment du build.

En résumé

- Utilisez l'option **#1 (ENV dans Dockerfile)** si le proxy est toujours nécessaire.
- Utilisez l'option **#3 (build args)** si vous souhaitez plus de flexibilité.

L'approche `RUN export` ne fonctionne que pour une seule commande : c'est pourquoi `ENV` est plus fiable et conforme à l'esprit Docker.

Environnement Cloud : GitHub Codespaces ☁️

Outils Nécessaires 🛠️

- Un navigateur web récent (et décent)
- Un compte sur GitHub

GitHub Codespaces

GitHub Codespaces : Environnement de développement dans le ☁️ "nuage"

- **But:** reproductible et instantané
- Puissance de calcul sur un serveur distant
- Éditeur VSCode dans le navigateur
- Docker **déjà installé et configuré** 🐳

Pourquoi Codespaces ? 🤔

Prérequis 📋

Vous avez déjà ce qu'il faut ! ✓

Démarrer avec Codespaces 🚀

⚠️ Passez à la slide suivante pour voir l'interface

Interface Codespaces 💻

Gauche : Explorateur de fichiers
→ Arborescence, Git

Centre : Éditeur de code
→ Coloration, autocomplétion

Bas : Terminal intégré
→ Bash/Zsh, Docker

Terminal Codespaces 🖥️

Le terminal est votre outil principal pour Docker !

```
# Vérifier l'utilisateur
whoami
# Résultat attendu : codespace

# Vérifier Docker
docker --version
# Résultat attendu : Docker version XX.XX.X

# Tester Docker
docker run hello-world
```

⚠️ Passez à la slide suivante pour comprendre la configuration

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
/workspaces/cours-devops-docker/npm-packages
Welcome to Codespaces! You are on our default image.
  - It includes runtimes and tools for Python, Node.js, Docker, and more. See the full list here: https://aka.ms/ghcs-default-image
  - Want to use a custom image instead? Learn more here: https://aka.ms/configure-codespace

To explore VS Code to its fullest, search using the Command Palette (Cmd/Ctrl + Shift + P or F1).

Edit away, run your app as usual, and we'll automatically make it available for you to access.

● @gounthar ➜ /workspaces/cours-devops-docker (main) $ whoami
codespace
● @gounthar ➜ /workspaces/cours-devops-docker (main) $ docker --version
Docker version 28.3.1-1, build 38b7060a218775811da953650d8df7d492653f8f
● @gounthar ➜ /workspaces/cours-devops-docker (main) $ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:6dc565aa630927052111f823c303948cf83670a3903ffa3849f1488ab517f891
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

○ @gounthar ➜ /workspaces/cours-devops-docker (main) $
```

Configuration Codespaces (Optionnelle) ⚙️

Codespaces peut être personnalisé avec `.devcontainer/`

```
// .devcontainer/devcontainer.json
{
  "name": "Docker DevOps Course",
  "image": "mcr.microsoft.com/devcontainers/base:ubuntu",
  "features": {
    "ghcr.io/devcontainers/features/docker-in-docker:2": {}
  }
}
```

Checkpoint 🎯

Vérifiez que tout fonctionne :

Gestion de votre Codespace 🔧

Optimisez votre quota gratuit de 60h/mois :

💡 **Astuce** : 60h = 2h/semaine pendant 30 semaines (largement suffisant !)

Avantages de Codespaces 🌟

Pour vous :

Pour le cours :

Limites et Alternatives ⚠️

Limites de Codespaces :

💡 Vous pouvez combiner les deux approches !

Ressources et Aide 📚

Documentation officielle :

- GitHub Codespaces Docs
- VSCode Documentation
- Docker Documentation

Support :

- 🙋 Questions pendant le cours
- 💬 Forum/Discord du cours (si disponible)
- 🐛 GitHub Issues pour bugs

💡 N'hésitez jamais à poser des questions !

Recap : GitHub Codespaces ✅

Ce que vous devez retenir :

➡️ Prêts pour commencer à utiliser Docker !

🐳 Installation locale de Docker

Ubuntu 25.04 et le conflit Podman ⚠️

Problème : Ubuntu 25.04+ est livré avec Podman par défaut

Symptôme : `Cannot connect to the Docker daemon`

⚠️ Conflit entre `/var/run/docker.sock` et `/run/podman/podman.sock`

Solutions au conflit

Solution 1 : Installer Docker

```
sudo apt install docker.io
sudo systemctl enable --now docker
sudo usermod -aG docker $USER
```

⚠️ Logout/login requis pour groupe docker

Solution 2 : Désinstaller Podman ✅

```
sudo apt remove podman
sudo apt install docker.io
sudo systemctl enable --now docker
sudo usermod -aG docker $USER
```

⚠️ Logout/login requis

Environnement prêt ! ✅

Votre environnement Docker est maintenant configuré 🎉

- ✅ GitHub Codespaces : Docker déjà installé
- ✅ Ubuntu 25.04 : Conflit Podman résolu
- ✅ Prêt pour la suite !

Prochaine étape 🚀

Avant de découvrir Docker, un détour essentiel…

Pourquoi la CLI ? 💻

Ligne de commande

Guide de survie

🤔 Problématique

- Communication Humain <→ Machine
- Base commune de TOUS les outils

CLI

- 🇬🇧 CLI == "Command Line Interface"
- 🇫🇷 "Interface de Ligne de Commande"

REPL

Pour les théoriciens et curieux :

- 🇬🇧 REPL == "Read–eval–print loop"

https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop

Anatomie d'une commande

```
ls --color=always -l /bin
```

- Séparateur : l'espace
- Premier élément (`ls`) : c'est la commande
- Les éléments commençant par un tiret − sont des "options" et/ou drapeaux ("flags")
  - "Option" == "Optionnel"
- Les autres éléments sont des arguments (`/bin`)
  - Nécessaire (par opposition)

Manuel des commande

- Afficher le manuel de `<commande>` :

```
# Commande 'man' avec comme argument le nom de ladite commande
man <commande>
```

  - Navigation avec les flèches haut et bas
    - Tapez `/` puis une chaîne de texte pour chercher
    - Touche `n` pour sauter d'occurrence en occurrence
  - Touche `q` pour quitter

🎓 Essayez avec `ls`, chercher le mot `color`

- 💡 La majorité des commandes fournit également une option (`--help`), un flag (`-h`) ou un argument (`help`)
- Google c'est pratique aussi hein !

Raccourcis

Dans un terminal Unix/Linux/WSL :

- `CTRL + C` : Annuler le process ou prompt en cours
- `CTRL + L` : Nettoyer le terminal
- `CTRL + A` : Positionner le curseur au début de la ligne
- `CTRL + E` : Positionner le curseur à la fin de la ligne
- `CTRL + R` : Rechercher dans l'historique de commandes

🎓 Essayez-les !

Commandes de base 1/2

- `pwd` : Afficher le répertoire courant
  - 🎓 Option `-P` ?
- `ls` : Lister le contenu du répertoire courant
  - 🎓 Options `-a` et `-l` ?
- `cd` : Changer de répertoire
  - 🎓 Sans argument : que se passe t'il ?
- `cat` : Afficher le contenu d'un fichier
  - 🎓 Essayez avec plusieurs arguments
- `mkdir` : créer un répertoire
  - 🎓 Option `-p` ?

Commandes de base 2/2

- `echo` : Afficher un (des) message(s)
- `rm` : Supprimer un fichier ou dossier
- `touch` : Créer un fichier
- `grep` : Chercher un motif de texte

Arborescence de fichiers 1/2

- Le système de fichier a une structure d'arbre
  - La racine du disque dur c'est `/` : 🎓 `ls -l /`
  - Le séparateur c'est également `/` : 🎓 `ls -l /usr/bin`
- Deux types de chemins :
  - Absolu (depuis la racine): Commence par `/` (Ex. `/usr/bin`)
  - Sinon c'est relatif (e.g. depuis le dossier courant) (Ex `./bin` ou `local/bin/`)

Source

Arborescence de fichiers 2/2

- Le dossier "courant" c'est `.` : 🎓 `ls -l ./bin # Dans le dossier /usr`
- Le dossier "parent" c'est `..` : 🎓 `ls -l ../ # Dans le dossier /usr`
- ~ (tilde) c'est un raccourci vers le dossier de l'utilisateur courant : 🎓 `ls -l ~`
- Sensible à la casse (majuscules/minuscules) et aux espaces : 🎓

```
ls -l /bin
ls -l /Bin
mkdir ~/"Accent tué"
ls -d ~/Accent\ tué
```

Un language (?)

- Variables interpolées avec le caractère "dollar" `$` :

```
echo $MA_VARIABLE
echo "$MA_VARIABLE"
echo ${MA_VARIABLE}

# Recommendation
echo "${MA_VARIABLE}"

MA_VARIABLE="Salut tout le monde"

echo "${MA_VARIABLE}"
```

- Sous commandes avec `$(<command>)` :

```
echo ">> Contenu de /tmp :\n$(ls /tmp)"
```

- Des `if`, des `for` et plein d'autres trucs (https://tldp.org/LDP/abs/html/)

Codes de sortie

- Chaque exécution de commande renvoie un code de retour (🇬🇧 "exit code")
  - Nombre entier entre 0 et 255 (en POSIX)
- Code accessible dans la variable **éphémère** `$?` :

```
ls /tmp
echo $?

ls /do_not_exist
echo $?

# Une seconde fois. Que se passe-t'il ?
echo $?
```

Entrée, sortie standard et d'erreur

```
ls -l /tmp
echo "Hello" > /tmp/hello.txt
ls -l /tmp
ls -l /tmp >/dev/null
ls -l /tmp 1>/dev/null

ls -l /do_not_exist
ls -l /do_not_exist 1>/dev/null
ls -l /do_not_exist 2>/dev/null

ls -l /tmp /do_not_exist
ls -l /tmp /do_not_exist 1>/dev/null 2>&1
```

Pipelines

- Le caractère "pipe" `|` permet de chaîner des commandes
  - Le "stdout" de la première commande est branchée sur le "stdin" de la seconde
- Exemple : Afficher les fichiers/dossiers contenant le lettre `d` dans le dossier `/bin` :

```
ls -l /bin

ls -l /bin | grep "d" --color=auto
```

Exécution 1/2

- Les commandes sont des fichier binaires exécutables sur le système :

```
command -v cat # équivalent de "which cat"

ls -l "$(command -v cat)"
```

- La variable d'environnement `$PATH` liste les dossiers dans lesquels chercher les binaires
  - 💡 Utiliser cette variable quand une commande fraîchement installée n'est pas trouvée

Exécution 2/2

- Exécution de scripts :
  - Soit appel direct avec l'interprétateur : `sh ~/monscript.txt`
  - Soit droit d'exécution avec un "shebang" (e.g. `#!/bin/bash`)

```
$ chmod +x ./monscript.sh

$ head -n1 ./monscript.sh
#!/bin/bash

$ ./monscript.sh
# Exécution
```

Git

Guide de survie

🤔 Problématique

- Support de communication
  - Humain → Machine
  - Humain <→ Humain
- Besoins de traçabilité, de définition explicite et de gestion de conflits
  - Collaboration requise pour chaque changement (revue, responsabilités)

Tracer le changement dans le code

avec un **VCS** : 🇬🇧 Version Control System

également connu sous le nom de SCM (🇬🇧 Source Code Management)

Pourquoi un VCS ?

- Pour conserver une trace de **tous** les changements dans un historique
  - "Source unique de vérité" (🇬🇧 *Single Source of Truth*)
- Pour **collaborer** efficacement sur un même référentiel

Concepts des VCS

Source : https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Quel VCS utiliser ?

Nous allons utiliser **Git**

Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

https://git-scm.com/

Les 3 états avec Git

- L'historique ("Version Database") : dossier `.git`
- Dossier de votre projet ("Working Directory") - Commande
- La zone d'index ("Staging Area")

Source : https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#*les_trois*%C3%A9tats

🎓 Exercice avec Git - 1.1

- Dans le terminal de votre environnement GitPod:
  - Créez un dossier vide nommé `projet-vcs-1` dans le répertoire `/workspace`, puis positionnez-vous dans ce dossier

```
mkdir -p /workspace/projet-vcs-1/
cd /workspace/projet-vcs-1/
```

  - Est-ce qu'il y a un dossier `.git/` ?
  - Essayez la commande `git status` ?
- Initialisez le dépôt git avec `git init`
  - Est-ce qu'il y a un dossier `.git/` ?
  - Essayez la commande `git status` ?

✅ Solution de l’exercice avec Git - 1.1

```
mkdir -p /workspace/projet-vcs-1/
cd /workspace/projet-vcs-1/
ls -la # Pas de dossier .git
git status # Erreur "fatal: not a git repository"
git init ./
ls -la # On a un dossier .git
git status # Succès avec un message "On branch main No commits yet"
```

🎓 Exercice avec Git - 1.2

- Créez un fichier `README.md` dedans avec un titre et vos nom et prénoms
  - Essayez la commande `git status` ?
- Ajoutez le fichier à la zone d'indexation à l'aide de la commande `git add (...)`
  - Essayez la commande `git status` ?
- Créez un commit qui ajoute le fichier `README.md` avec un message, à l'aide de la commande `git commit -m <message>`
  - Essayez la commande `git status` ?

✅ Solution de l’exercice avec Git - 1.2

```
echo "# Read Me\n\nObi Wan" > ./README.md
git status # Message "Untracked file"

git add ./README.md
git status # Message "Changes to be committed"
git commit -m "Ajout du README au projet"
git status  # Message "nothing to commit, working tree clean"
```

Terminologie de Git - Diff et changeset

diff: un ensemble de lignes "changées" sur un fichier donné

changeset: un ensemble de "diff" (donc peut couvrir plusieurs fichiers)

Terminologie de Git - Commit

commit: un changeset qui possède un (commit) parent, associé à un message

"HEAD": C'est le dernier commit dans l'historique

🎓 Exercice avec Git - 2

- Afficher la liste des commits
- Afficher le changeset associé à un commit
- Modifier du contenu dans `README.md` et afficher le diff
- Annulez ce changement sur `README.md`

✅ Solution de l’exercice avec Git - 2

```
git log

git show # Show the "HEAD" commit
echo "# Read Me\n\nObi Wan Kenobi" > ./README.md

git diff
git status

git checkout -- README.md
git status
```

Terminologie de Git - Branche

- Abstraction d’une version "isolée" du code
- Concrètement, une **branche** est un alias pointant vers un "commit"

🎓 Exercice avec Git - 3

- Créer une branche nommée `feature/html`
- Ajouter un nouveau commit contenant un nouveau fichier `index.html` sur cette branche
- Afficher le graphe correspondant à cette branche avec `git log --graph`

✅ Solution de l’exercice avec Git - 3

```
git switch --create feature/html
# Ou alors: git branch feature/html && git switch feature/html
echo '<h1>Hello</h1>' > ./index.html
git add ./index.html && git commit --message="Ajout d'une page HTML par défaut" # -m / --message

git log
git log --graph
git lg # cat ~/.gitconfig => regardez la section section [alias], cette commande est déjà définie!
```

Terminologie de Git - Merge

- On intègre une branche dans une autre en effectuant un **merge**
  - Un nouveau commit est créé, fruit de la combinaison de 2 autres commits

🎓 Exercice avec Git - 4

- Merger la branche `feature/html` dans la branche principale
  - ⚠️ Pensez à utiliser l'option `--no-ff`
- Afficher le graphe correspondant à cette branche avec `git log --graph`

✅ Solution de l’exercice avec Git - 4

```
git switch main
git merge --no-ff feature/html # Enregistrer puis fermer le fichier 'MERGE_MSG' qui a été ouvert
git log --graph

# git lg
```

Feature Branch Flow

- **Une seule** branche **par** fonctionnalité

Exemple d’usages de VCS

- "Infrastructure as Code" :
  - Besoins de traçabilité, de définition explicite et de gestion de conflits
  - Collaboration requise pour chaque changement (revue, responsabilités)
- Code Civil:
  - https://github.com/steeve/france.code-civil
  - https://github.com/steeve/france.code-civil/pull/40
  - https://github.com/steeve/france.code-civil/commit/b805ecf05a86162d149d3d182e04074ecf72c066

Checkpoint 🎯

- `git` est un des (plus populaires) de système de contrôle de versions
- Cet outil vous permet:
  - D'avoir un historique auditable de votre code source
  - De collaborer efficacement sur le code source (conflit `git` == "PARLEZ-VOUS")

  ⇒ C'est une ligne de commande (trop?) complète qui nécessite de pratiquer

Containers

🎓 Exercice : Votre premier conteneur

C’est à vous (ouf) !

- Allez dans Gitpod
- Dans un terminal, tapez la commande suivante :

```
docker container run hello-world
# Equivalent de l'ancienne commande 'docker run'
```

Anatomie

- Un service "Docker Engine" tourne en tâche de fond et publie une API REST
- La commande `docker container run` ... a lancé un client docker qui a envoyé une requête `POST` au service docker
- Le service a téléchargé une **Image** Docker depuis le registre **DockerHub**,
- Puis a exécuté un **conteneur** basé sur cette image

Anatomie

$

$

Anatomie

```
$ docker container run busybox echo hello world
```

```
$
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

```
$
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

```
$ docker container run centos date
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

```
$ docker container run centos date
Mon Sep  25 19:33:19 UTC 2023
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

Lancement d'un container

```
$ docker container run centos date
Mon Sep  25 19:33:19 UTC 2023
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

Lancement d'un container

Image

```
$ docker container run centos date
Mon Sep  25 19:33:19 UTC 2023
```

Anatomie

```
$ docker container run busybox echo hello world
hello world
```

Lancement d'un container

Image

Commande lancée

```
$ docker container run centos date
Mon Sep  25 19:33:19 UTC 2023
```

Anatomie

Anatomie

ON LANCE UNE COMMANDE DANS UN
SHELL DANS UNE COMMANDE DANS UN SHELL
TOUT CA POUR AFFICHER LA DATE ???

🎓 Exercice : Où est mon conteneur ?

C'est à vous !

```
docker container ls --help
# ...
docker container ls
# ...
docker container ls --all
```

⇒ 🤔 comment comprenez vous les résultats des 2 dernières commandes ?

✅ Solution : Où est mon conteneur ?

Le conteneur est toujours présent dans le "Docker Engine" même en étant arrêté

```
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
109a9cdd3ec8   hello-world   "/hello"   33 seconds ago   Exited (0) 17 seconds ago             festive_faraday
```

- Un conteneur == une commande "conteneurisée"
  - cf. colonne "**COMMAND**"
- Quand la commande s’arrête : le conteneur s’arrête
  - cf. code de sortie dans la colonne "**STATUS**"

Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

🫁 Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

- Le moteur Docker crée un container à partir de l'image "busybox".

Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.

Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l'intérieur du container.

🫁 Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l'intérieur du container.
- On obtient le résultat dans la sortie standard de la machine hôte.

🫁 Rappels d’anatomie

```
$ docker container run busybox echo hello world
```

- Le moteur Docker crée un container à partir de l’image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l’intérieur du container.
- On obtient le résultat dans la sortie standard de la machine hôte.
- Le container est stoppé.

Cas d’usage

Outillage jetable

- Tester une version de Maven, de JDK, de NPM, …

🎓 Exercice : Cycle de vie d'un conteneur simple

- Lancez un nouveau conteneur nommé `bonjour`
  - 💡 `docker container run --help` ou Documentation en ligne
- Affichez les "logs" du conteneur (==traces d'exécution écrites sur le stdout + stderr de la commande conteneurisée)
  - 💡 `docker container logs --help` ou Documentation en ligne
- Lancez le conteneur avec la commande `docker container start`
  - Regardez le résultat dans les logs
- Supprimez le container avec la commande `docker container rm`

✅ Solution : Cycle de vie d'un conteneur simple

```
docker container run --name=bonjour hello-world
# Affiche le texte habituel

docker container logs bonjour
# Affiche le même texte : pratique si on a fermé le terminal

docker container start bonjour
# N'affiche pas le texte mais l'identifiant unique du conteneur 'bonjour'

docker container logs bonjour
# Le texte est affiché 2 fois !

docker container ls --all
# Le conteneur est présent
docker container rm bonjour
docker container ls --all
# Le conteneur n'est plus là : il a été supprimé ainsi que ses logs

docker container logs bonjour
# Error: No such container: bonjour
```

🤔 Que contient "hello-world" ?

- C’est une "image" de conteneur, c’est à dire un modèle (template) représentant une application auto-suffisante.
  - On peut voir ça comme un "paquetage" autonome
- C’est un système de fichier complet:
  - Il y a au moins une racine `/`
  - Ne contient que ce qui est censé être nécessaire (dépendances, librairies, binaires, etc.)

Docker Hub

- https://hub.docker.com/ : C’est le registre d’images "par défaut"
  - Exemple : Image officielle de conteneur "Ubuntu"
- 🎓 Cherchez l’image `hello-world` pour en voir la page de documentation
  - 💡 pas besoin de créer de compte pour ça
- Il existe d’autre "registres" en fonction des besoins (GitHub GHCR, Google GCR, etc.)

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
```

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
/ $
```

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
/ $
```

- On lance un container à partir de l’image "alpine"

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
/ $
```

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
/ $
```

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container
- On redirige l'entrée standard avec -i

🏓 Lancer un container interactif

```
docker container run  --interactive --tty alpine
/ $
```

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container
- On redirige l'entrée standard avec -i
- On déclare un pseudo-terminal avec -t

🎓 Exercice : conteneur interactif

- Quelle distribution Linux est utilisée dans le terminal Gitpod ?
  - 💡 Regardez le fichier `/etc/os-release`
- Exécutez un conteneur interactif basé sur `alpine:3.18.3` (une distribution Linux ultra-légère) et regardez le contenu du fichier au même emplacement
  - 💡 `docker container run --help`
  - 💡 Demandez un `tty` à Docker
  - 💡 Activez le mode interactif
- Exécutez la même commande dans un conteneur basé sur la même image mais en **NON** interactif
  - 💡 Comment surcharger la commande par défaut ?

✅ Solution : conteneur interactif

```
$ cat /etc/os-release
# ... Ubuntu ....

$ docker container run --tty --interactive alpine:3.18.3
/ $ cat /etc/os-release
# ... Alpine ...
# Notez que le "prompt" du terminal est différent DANS le conteneur
/ $ exit
$ docker container ls --all

$ docker container run alpine:3.18.3 cat /etc/os-release
# ... Alpine ...
```

🏓 Utiliser un container interactif

Revenons dans notre container interactif de tout à l'heure…

```
/ $ curl google.fr
```

🏓 Utiliser un container interactif

```
/ $ curl google.fr
/bin/sh: curl: not found
```

🏓 Utiliser un container interactif

```
/ $ curl google.fr
/bin/sh: curl: not found
```

`cURL` n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable

🏓 Utiliser un container interactif

```
/ $ curl google.fr
/bin/sh: curl: not found
```

`cURL` n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable

```
/ $ apk update && apk add curl
```

🏓 Utiliser un container interactif

```
/ $ curl google.fr
/bin/sh: curl: not found
```

`cURL` n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable

```
/ $ apk update && apk add curl
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
v3.18.3-169-gd5adf7d7d28 [https://dl-cdn.alpinelinux.org/alpine/v3.18/main]
v3.18.3-171-g503977088b4 [https://dl-cdn.alpinelinux.org/alpine/v3.18/community]
OK: 20063 distinct packages available
(1/7) Installing ca-certificates (20230506-r0)
(2/7) Installing brotli-libs (1.0.9-r14)
(3/7) Installing libunistring (1.1-r1)
(4/7) Installing libidn2 (2.3.4-r1)
(5/7) Installing nghttp2-libs (1.55.1-r0)
(6/7) Installing libcurl (8.2.1-r0)
(7/7) Installing curl (8.2.1-r0)
Executing busybox-1.36.1-r2.trigger
Executing ca-certificates-20230506-r0.trigger
OK: 12 MiB in 22 packages
```

🏓 Utiliser un container interactif

```
/ $ curl google.fr
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.fr/">here</A>.
</BODY></HTML>
```

C’est bon, on a `cURL`

🏓 Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ $ exit
```

🏓 Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ $ exit
```

Si on veut réutiliser `cURL` sur Alpine, c'est simple, on relance le shell, non? 🤔

🏓 Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ $ exit
```

Si on veut réutiliser `cURL` sur Alpine, c'est simple, on relance le shell, non? 🤔

```
docker container run  --interactive --tty alpine
```

🏓 Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ $ exit
```

Si on veut réutiliser `cURL` sur Alpine, c'est simple, on relance le shell, non? 🤔

```
docker container run  --interactive --tty alpine
```

On relance `cURL`:

```
/ $ curl google.fr
```

🏓 Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ $ exit
```

Si on veut réutiliser `cURL` sur Alpine, c'est simple, on relance le shell, non? 🤔

```
docker container run  --interactive --tty alpine
```

On relance `cURL`:

```
/ $ curl google.fr
/bin/sh: curl: not found
```

🏓 Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

- Cette commande instancie un "nouveau container à chaque fois" !

🏓 Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

- Cette commande instancie un "nouveau container à chaque fois" !
- Chaque container est différent.

🏓 Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

- Cette commande instancie un "nouveau container à chaque fois" !
- Chaque container est différent.
- Aucun partage entre les containers à part le contenu de base de l'image.

🏓 Utiliser un container interactif

" On m'a vendu un truc qui permet de lancer des tonnes de microservices… mais là, on télécharge nimps

⛅ Conteneur en tâche de fond

Lançons un container bien particulier…

```
docker container run  --interactive --tty jpetazzo/clock
```

⛅ Conteneur en tâche de fond

Lançons un container bien particulier…

```
docker container run  --interactive --tty jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
```

⛅ Conteneur en tâche de fond

Lançons un container bien particulier…

```
docker container run  --interactive --tty jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
```

Ce container va tourner indéfiniment sauf si on le stoppe avec ⌨️ `Ctrl` + `C` …

⛅ Conteneur en tâche de fond

Lançons un container bien particulier…

```
docker container run  --interactive --tty jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
```

Ce container va tourner indéfiniment sauf si on le stoppe avec ⌨️ `Ctrl` + `C` …

… mais ca va stopper le container !

⛅ Conteneur en tâche de fond

Lançons un container bien particulier…

```
docker container run  --interactive --tty jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
```

Ce container va tourner indéfiniment sauf si on le stoppe avec ⌨️ `Ctrl` + `C` …

… mais ca va stopper le container !

Oui car quand on stoppe le processus principal d'un container, ce dernier n'a plus de raison d'exister et s'arrête naturellement.

⛅ Conteneur en tâche de fond

La solution : le flag `--detach`

```
docker container run --detach jpetazzo/clock
```

⛅ Conteneur en tâche de fond

La solution : le flag `--detach`

```
docker container run --detach jpetazzo/clock
399f3b23bc0585991afa80dfee854cf0a953d782b99153b4e2cbc74ab6b07770
```

Le retour de cette commande correspond à l'identifiant unique du container.

Cette fois-ci, le container tourne, mais en arrière plan !

⛅ Conteneur en tâche de fond

" Okay, maintenant il n'écrit la date nulle part… mais toutes les secondes… à l'aide ! "

⛅ Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard… du container !

Comment retrouver le contenu de la sortie standard du container ?

⛅ Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard… du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
```

⛅ Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard… du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
Tue Sep 12 12:37:01 UTC 2023
Tue Sep 12 12:37:02 UTC 2023
Tue Sep 12 12:37:03 UTC 2023
Tue Sep 12 12:37:04 UTC 2023
Tue Sep 12 12:37:05 UTC 2023
Tue Sep 12 12:37:06 UTC 2023
Tue Sep 12 12:37:07 UTC 2023
Tue Sep 12 12:37:08 UTC 2023
Tue Sep 12 12:37:09 UTC 2023
Tue Sep 12 12:37:10 UTC 2023
Tue Sep 12 12:37:11 UTC 2023
```

⛅ Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard… du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
Tue Sep 12 12:37:01 UTC 2023
Tue Sep 12 12:37:02 UTC 2023
Tue Sep 12 12:37:03 UTC 2023
Tue Sep 12 12:37:04 UTC 2023
Tue Sep 12 12:37:05 UTC 2023
Tue Sep 12 12:37:06 UTC 2023
Tue Sep 12 12:37:07 UTC 2023
Tue Sep 12 12:37:08 UTC 2023
Tue Sep 12 12:37:09 UTC 2023
Tue Sep 12 12:37:10 UTC 2023
Tue Sep 12 12:37:11 UTC 2023
```

Ouf ! On n’est pas obligé de saisir l’identifiant complet ! Il suffit de fournir le nombre suffisant de caractères pour que ce soit discriminant.

📖 Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

💡 Commande vue un peu plus tôt…

📖 Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
```

📖 Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
CONTAINER ID   IMAGE                            COMMAND                  CREATED        STATUS       PORTS
02cbf2fb3721   cours-devops-docker-serve        "/sbin/tini -g gulp …"   3 hours ago    Up 3 hours   0.0.0.0:8000->8000/tc
ebfbe695b2ec   moby/buildkit:buildx-stable-1    "buildkitd"              2 months ago   Up 3 hours
```

📖 Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
CONTAINER ID   IMAGE                           COMMAND                  CREATED        STATUS       PORTS
02cbf2fb3721   cours-devops-docker-serve       "/sbin/tini -g gulp …"   3 hours ago    Up 3 hours   0.0.0.0:8000->8000/tc
ebfbe695b2ec   moby/buildkit:buildx-stable-1   "buildkitd"              2 months ago   Up 3 hours
```

On obtient un tableau de tous les containers en cours d'exécution.

🛑 🏁 Stop / Start

Il est possible de stopper un container.

```
docker container stop 399f3
```

🛑 🏁 Stop / Start

Il est possible de stopper un container.

```
docker container stop 399f3
```

Pour redémarrer un container :

```
docker container start 399f3
```

📖 Lister tous les containers

Même les 💀 .

📖 Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

```
docker container ls --all
```

📖 Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

```
docker container ls --all
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                          PO
90725f661d4e   hello-world                    "/hello"                 13 seconds ago   Exited (0) 12 seconds ago
9d0a6586b9e1   busybox                        "echo hello world"       22 seconds ago   Exited (0) 21 seconds ago
368ed08a35e3   jpetazzo/clock                 "/bin/sh -c 'while d…"   6 minutes ago    Up 5 minutes
c036e57bbf05   jpetazzo/clock                 "/bin/sh -c 'while d…"   17 minutes ago   Exited (130) 17 minutes ago
02cbf2fb3721   cours-devops-docker-serve      "/sbin/tini -g gulp …"   3 hours ago      Up 3 hours                      0.
ebfbe695b2ec   moby/buildkit:buildx-stable-1  "buildkitd"              2 months ago     Up 4 hours
```

📖 Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

```
 docker container ls --all
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                        PO
90725f661d4e   hello-world                    "/hello"                 13 seconds ago   Exited (0) 12 seconds ago
9d0a6586b9e1   busybox                        "echo hello world"       22 seconds ago   Exited (0) 21 seconds ago
368ed08a35e3   jpetazzo/clock                 "/bin/sh -c 'while d…"   6 minutes ago    Up 5 minutes
c036e57bbf05   jpetazzo/clock                 "/bin/sh -c 'while d…"   17 minutes ago   Exited (130) 17 minutes ago
02cbf2fb3721   cours-devops-docker-serve      "/sbin/tini -g gulp …"   3 hours ago      Up 3 hours                    0.
ebfbe695b2ec   moby/buildkit:buildx-stable-1  "buildkitd"              2 months ago     Up 4 hours
```

Avec le flag `--all` , on obtient un tableau de tous les containers quel que soit leur état.

🧽 Nettoyage

Tout container stoppé peut être supprimé.

🧽 Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e
```

🧽 Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e
90725f661d4e
```

🧽 Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e
90725f661d4e
```

Container "auto-nettoyant" 🗑️

🧽 Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e
90725f661d4e
```

Container "auto-nettoyant" 🗑️

```
docker container run  --interactive --tty --rm jpetazzo/clock
```

Aussitôt stoppé, aussitôt supprimé !

⏰ Rappel: cycle de vie d'un container

⏰ Rappel: cycle de vie d'un container

⏰ Rappel: cycle de vie d'un container

⏰ Rappel: cycle de vie d'un container

⏰ Rappel: cycle de vie d'un container

🔄 Reprendre le contrôle

Sur un container en arrière-plan

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container exec <containerID> echo "hello"
```

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container ls
```

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container ls
CONTAINER ID   IMAGE                          COMMAND                  CREATED        STATUS       PORTS
368ed08a35e3   jpetazzo/clock                 "/bin/sh -c 'while d…"   4 hours ago    Up 4 hours
02cbf2fb3721   cours-devops-docker-serve      "/sbin/tini -g gulp …"   7 hours ago    Up 7 hours   0.0.0.0:8000->8000/tc
ebfbe695b2ec   moby/buildkit:buildx-stable-1  "buildkitd"              2 months ago   Up 7 hours
```

🔄 Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container ls
CONTAINER ID   IMAGE                           COMMAND                  CREATED        STATUS       PORTS
368ed08a35e3   jpetazzo/clock                  "/bin/sh -c 'while d…"   4 hours ago    Up 4 hours
02cbf2fb3721   cours-devops-docker-serve       "/sbin/tini -g gulp …"   7 hours ago    Up 7 hours   0.0.0.0:8000->8000/tc
ebfbe695b2ec   moby/buildkit:buildx-stable-1   "buildkitd"              2 months ago   Up 7 hours

docker container exec 368ed08a35e3 echo hello
hello
```

🔄 Reprendre le contrôle

Sur un container en arrière-plan

```
docker container exec --interactive --tty <containerID> bash
```

Ca fonctionne aussi en interactif !

🎓 Exercice : conteneur en tâche de fond

Étapes

- Lancer un container "daemon" `jpetazzo/clock`
- Utiliser l'équivalent de `tail -f` pour lire la sortie standard du container 💡
- Lancer un second container "daemon" 👹
- Stocker l'identifiant de ce container dans une variable du shell, en une seule commande et en jouant avec `docker container ls`
- Stopper le container avec cet identifiant
- Afficher les containers lancés 🏃📦
- Afficher les containers arrêtés 🛑📦

✅ Solution : conteneur en tâche de fond

```
# Lancer un container "daemon" `jpetazzo/clock`
docker container run --detach --name first-clock jpetazzo/clock

# Utiliser l'équivalent de `tail -f` pour lire la sortie standard du container 💡
docker container logs -f first-clock

# * Lancer un second container "daemon" 👹
docker container run --detach --name second-clock jpetazzo/clock

# Stocker l'identifiant de ce container dans une variable du shell, en une seule commande et en jouant avec `docker conta
# --filter "name=second-clock" filters the list of containers to include only those with the name "second-clock."
container_id=$(docker container ls -q --filter "name=second-clock")

# Stopper le container avec cet identifiant
docker container stop "$container_id"

# Afficher les containers lancés  🏃📦
docker container ls

# Afficher les containers arrêtés 🛑📦
docker container ls --filter "status=exited"
```

🎓 Exercice : conteneur en tâche de fond

- Relancer un des containers arrêtés.
- Exécuter un `ps –ef `dans ce container
- Quel est le PID du process principal ?
- Vérifier que le container "tourne" toujours
- Supprimer l’image (tip : `docker rmi`)
- Supprimer les containers
- Supprimer l’image (pour de vrai cette fois)

✅ Solution : conteneur en tâche de fond

```
# Relancer un des containers arrêtés.
docker container start second-clock

# Exécuter un `ps -ef `dans ce container
docker container exec second-clock ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c while date; do sleep 1; done
   56 root      0:00 sleep 1
   57 root      0:00 ps -ef

# Quel est le PID du process principal ?
# 1

# Vérifier que le container "tourne" toujours
docker container ls
CONTAINER ID   IMAGE                            COMMAND                  CREATED         STATUS              PORTS
7d7085809ad2   cours-devops-docker-serve        "/sbin/tini -g gulp …"   6 minutes ago   Up 5 minutes        0.0.0.0:8000-
edea0c46c6d8   jpetazzo/clock                   "/bin/sh -c 'while d…"   9 minutes ago   Up About a minute
ebfbe695b2ec   moby/buildkit:buildx-stable-1    "buildkitd"              2 months ago    Up 11 hours

# Supprimer l'image (tip : `docker rmi`)
docker rmi jpetazzo/clock
Error response from daemon: conflict: unable to remove repository reference "jpetazzo/clock" (must force) - container ede

# Supprimer les containers
docker stop second-clock
second-clock

docker rm second-clockl
```

🎓 Exercice : conteneur en tâche de fond

- Exécutez un conteneur, basé sur l'image `nginx` en tâche de fond ("Background"), nommé `webserver-1`
  - 💡 On parle de processus "détaché" (ou bien "démonisé") 👹
  - ⚠️ Pensez bien à `docker container ls`
- Regardez le contenu du fichier `/etc/os-release` dans ce conteneur
  - 💡 `docker container exec`
- Essayez d'arrêter, démarrer puis redémarrer le conteneur
  - ⚠️ Pensez bien à `docker container ls` à chaque fois
  - 💡 `stop`, `start`, `restart`

✅ Solution : conteneur en tâche de fond

```
docker container run --detach --name=webserver-1 nginx
# <ID du conteneur>

docker container ls
docker container ls --all

docker container exec webserver-1 cat /etc/os-release
# ... Debian ...

docker container stop webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
```

Checkpoint 🎯

Vous savez désormais:

- Maîtriser le cycle de vie des containers
- Interagir avec les containers existants

Checkpoint 🎯

- Docker essaye de résoudre le problème de l’empaquetage le plus "portable" possible
  - On n’en a pas encore vu les effets, ça arrive !
- Vous avez vu qu’un containeur permet d’exécuter une commande dans un environnement "préparé"
  - Catalogue d’images Docker par défaut : Le Docker Hub
- Vous avez vu qu’on peut exécuter des conteneurs selon 3 modes :
  - "One shot"
  - Interactif
  - En tâche de fond

⇒ 🤔 Mais comment ces images sont-elles fabriquées ? Quelle confiance leur accorder ?

Docker Images

🤔 Pourquoi des images ?

- Un **conteneur** est toujours exécuté depuis une **image**.
- Une **image de conteneur** (ou "Image Docker") est un modèle ("template") d'application auto-suffisant.

⇒ Permet de fournir un livrable portable (ou presque).

🤔 C’est quoi une image ?

C’est une collection de fichiers et de metadonnées.

🤔 C’est quoi une image ?

C’est une collection de fichiers et de metadonnées.

C’est une suite de couches superposées.

🤔 C’est quoi une image ?

C’est une collection de fichiers et de metadonnées.

C’est une suite de couches superposées.

🤔 C’est quoi une image ?

C’est une collection de fichiers et de metadonnées.

C’est une suite de couches superposées.

Détail des couches

Exemple d’une image Apache HTTPd "custom"

Détail des couches

Exemple d’une image Apache HTTPd "custom"

Détail des couches

Exemple d’une image Apache HTTPd "custom"

Détail des couches

Exemple d’une image Apache HTTPd "custom"

Détail des couches

Exemple d'une image Apache HTTPd "custom"

Détail des couches

Exemple d'une image Apache HTTPd "custom"

Dicton du jour

"L'image est à la classe ce que le container est à l'objet"

Amaury W.
14 septembre 2023

🤔 Application Auto-Suffisante ?

C’est quoi le principe ?

🐳 📖 🍳 Le livre de recettes

🐳 📖 🍳 Le livre de recettes

🐳 📖 🍳 Le livre de recettes

🐳 📖 🍳 Le livre de recettes

🐳 📖 🍳 Le livre de recettes

🤔 Pourquoi fabriquer sa propre image ?

❗ Problème :

```
cat /etc/os-release
# ...
git --version
# ...

# Même version de Linux que dans GitPod
docker container run --rm ubuntu:22.04 git --version
# docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable t

# En interactif ?
docker container run --rm --tty --interactive ubuntu:22.04 git --version
```

🎓 Fabriquer sa première image

- **But :** fabriquer une image Docker qui contient `git`
- Dans votre workspace Gitpod, créez un dossier nommé `docker-git/`
- Dans ce dossier, créer un fichier `Dockerfile` avec le contenu ci-dessous :

```
FROM alpine:3.23.2

RUN apk update && apk add --no-cache git
```

- Fabriquez votre image avec la commande `docker image build --tag=docker-git chemin/vers/docker-git/`
- Testez l'image fraîchement fabriquée
  - 💡 `docker image ls`

✅ Fabriquer sa première image

```
cat <<EOF >Dockerfile
FROM alpine:3.23.2

RUN apk update && apk add --no-cache git
EOF

docker image build --tag=docker-git ./

docker image ls | grep docker-git

# Doit fonctionner
docker container run --rm docker-git:latest git --version
```

🐳 📖 🍳 Fabriquer son image

Un peu de cuisine…

🐳 📖 🍳 Fabriquer son image

💪🏁 DÉFI

Créer une image alpine avec un JRE installé.

Dockerfile

```
FROM alpine:3.18
LABEL maintainer="Tony Stark"

RUN apk update

RUN apk add openjdk17-jre-headless
```

couche de base

Dockerfile

```
FROM alpine:3.18
LABEL maintainer="Tony Stark"

RUN apk update

RUN apk add openjdk17-jre-headless
```

Dockerfile

couche de base

metadata

```
FROM alpine:3.18
LABEL maintainer="Tony Stark"

RUN apk update

RUN apk add openjdk17-jre-headless
```

Dockerfile

couche de base

metadata

```
FROM alpine:3.18
LABEL maintainer="Tony Stark"

RUN apk update

RUN apk add openjdk17-jre-headless
```

commandes d'installation

🐳 📖 🍳 Fabriquer son image

```
FROM alpine:3.23.2
LABEL maintainer="Tony Stark"

RUN apk update

RUN apk add --no-cache openjdk17-jre-headless
```

🐳 📖 🍳 Cuistot, au boulot!

```
docker image build -t myjava:1.42 .
```

🐳 📖 🍳 Cuistot, au boulot!

```
docker image build -t myjava:1.42 .
```

nom de l'image à créer

🐳 📖 🍳 Cuistot, au boulot!

```
docker image build -t myjava:1.42 .
```

nom de l'image à créer

Répertoire contenant le Dockerfile

🐳 📖 🍳 Cuistot, au boulot!

```
docker image build -t myjava:1.42 .
```

nom de l'image à créer

Répertoire contenant le Dockerfile

```
docker image build -t myjava:1.42 .
```

🪜 Étapes de construction

```
Sending build context to Docker daemon  9.728k
Step 1/4 : FROM alpine:3.18
 ---> 7e6257c9f8d8
Step 2/4 : LABEL maintainer="Tony Stark"
 ---> Running in 8dfc06b5f18b
Removing intermediate container 8dfc06b5f18b
 ---> e7584e63d1fe
Step 3/4 : RUN apk update
 ---> Running in 01c16b79b7e6
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
...
```

🪜 Étapes de construction

```
Sending build context to Docker daemon  9.728kB
Step 1/4 : FROM alpine:3.18
 ---> 7e6257c9f8d8
Step 2/4 : LABEL maintainer="Tony Stark"
 ---> Using cache
 ---> e7584e63d1fe
Step 3/4 : RUN apk update
 ---> Running in 876b9c10017a
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
...
```

🪜 Étapes de construction

```
Sending build context to Docker daemon  9.728kB
Step 1/4 : FROM alpine:3.18
 ---> 7e6257c9f8d8
Step 2/4 : LABEL maintainer="Tony Stark"
 ---> Using cache
 ---> e7584e63d1fe
Step 3/4 : RUN apk update
 ---> Running in 876b9c10017a
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
...
```

On gagne du temps grâce aux builds précédents !

Et après ?

L’image est dispo !

Registre local, mais encore?

On les trouve où, ces images ?

Les registres Docker

Ce sont des plates-formes qui hébergent les images.

Il est possible de créer ses propres registres (ex : registre privé d'entreprise)

DockerCon 2023: Our annual developer event is back — online & in person. Learn more.

node

Docker Official Image · 1B+ · 10K+

Node.js is a JavaScript-based platform for server-side and networking applications.

```
docker pull node
```

Overview Tags

Quick reference

- **Maintained by**:
  The Node.js Docker Team
- **Where to get help**:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `20-alpine3.17`, `20.7-alpine3.17`, `20.7.0-alpine3.17`, `alpine3.17`, `current-alpine3.17`
- `20-alpine`, `20-alpine3.18`, `20.7-alpine`, `20.7-alpine3.18`, `20.7.0-alpine`, `20.7.0-alpine3.18`, `alpine`, `alpine3.18`, `current-alpine`, `current-alpine3.18`
- `20`, `20-bookworm`, `20.7`, `20.7-bookworm`, `20.7.0`, `20.7.0-bookworm`, `bookworm`, `current`, `current-bookworm`, `latest`
- `20-bookworm-slim`, `20-slim`, `20.7-bookworm-slim`, `20.7-slim`, `20.7.0-bookworm-slim`, `20.7.0-slim`, `bookworm-slim`, `current-bookworm-slim`, `current-slim`, `slim`
- `20-bullseye`, `20.7-bullseye`, `20.7.0-bullseye`, `bullseye`, `current-bullseye`
- `20-bullseye-slim`, `20.7-bullseye-slim`, `20.7.0-bullseye-slim`, `bullseye-slim`, `current-

Recent Tags

lts-hydrogen lts-buster lts-bu
latest hydrogen-buster hydro
hydrogen-bookworm hydrogen

About Official Images

Docker Official Images are a
open source and drop-in solu

Why Official Images?

These images have clear doc
best practices, and are desig
common use cases.

🖼️ Les images

Avant d'instancier un container, il faut récupérer l'image en local.

Un téléchargement par couches

🖼️🏷️ Conventions de nommage des images

REGISTRE

Le registre sur lequel on souhaite récupérer l'image.

Optionnel quand on récupère depuis le Docker Hub.

🖼️🏷️ Conventions de nommage des images

REGISTRE	NAMESPACE
Le registre sur lequel on souhaite récupérer l'image. Optionnel quand on récupère depuis le Docker Hub.	Une référence au projet voire à un compte utilisateurs. Optionnel quand il s'agit d'une image officielle poussée par Docker

🖼️🏷️ Conventions de nommage des images

Le registre sur lequel on souhaite récupérer l'image.

Optionnel quand on récupère depuis le Docker Hub.

Une référence au projet voire à un compte utilisateurs.

Optionnel quand il s'agit d'une image officielle poussée par Docker

Le nom de l'image, il n'est pas optionnel.

🖼️🏷️ Conventions de nommage des images

REGISTRE	NAMESPACE	NOM	TAG
Le registre sur lequel on souhaite récupérer l'image. Optionnel quand on récupère depuis le Docker Hub.	Une référence au projet voire à un compte utilisateurs. Optionnel quand il s'agit d'une image officielle poussée par Docker	Le nom de l'image, il n'est pas optionnel.	Un marqueur ajouté au nom pour donner des précisions sur une version par exemple. Si le tag n'est pas fourni, Docker recherche "latest"

reg.mycompany.com/prj/myapp:12-4

reg.mycompany.com	prj	myapp	12-4

`reg.mycompany.com/prj/myapp:12-4`

reg.mycompany.com	prj	myapp	12-4

`foo/bar:baz`

Docker Hub	foo	bar	baz

`reg.mycompany.com/prj/myapp:12-4`

reg.mycompany.com	prj	myapp	12-4

`foo/bar:baz`

Docker Hub	foo	bar	baz

`alpine`

Docker Hub	Image Officielle	alpine	latest

reg.mycompany.com/prj/myapp:12-4

reg.mycompany.com	prj	myapp	12-4

foo/bar:baz

Docker Hub	foo	bar	baz

alpine

Docker Hub	Image Officielle	alpine	latest

hello-world:42

Registre local		hello-world	42

📷🏷️ Conventions de nommage des images

Explore > jenkins/jenkins

jenkins/jenkins

Sponsored OSS

By Jenkins • Updated a day ago

The leading open source automation server

Image

Pulls 1B+

Overview | Tags

Sort by Newest

jdk21

TAG

jdk21

Last pushed a day ago by jenkinsinfraadmin

```
docker pull jenkins/jenkins:jdk…
```

DIGEST	OS/ARCH	COMPRESSED SIZE
9b23ced2b7e4	linux/amd64	282.07 MB
7579a7d3caeb	linux/arm/v7	390.1 MB
b52835c4edf7	linux/arm64	280.53 MB

+2 more...

TAG

latest-jdk21-preview

Last pushed a day ago by jenkinsinfraadmin

```
docker pull jenkins/jenkins:lat…
```

DIGEST	OS/ARCH	COMPRESSED SIZE
9b23ced2b7e4	linux/amd64	282.07 MB
7579a7d3caeb	linux/arm/v7	390.1 MB
b52835c4edf7	linux/arm64	280.53 MB

+2 more...

TAG

2.424-jdk21

Last pushed a day ago by jenkinsinfraadmin

```
docker pull jenkins/jenkins:2.4…
```

DIGEST	OS/ARCH	COMPRESSED SIZE
9b23ced2b7e4	linux/amd64	282.07 MB
7579a7d3caeb	linux/arm/v7	390.1 MB
b52835c4edf7	linux/arm64	280.53 MB

+2 more...

📷🏷️ Conventions de nommage des images

Une même image peut avoir plusieurs noms et tags !

Le tag "latest" est régulièrement réaffecté sur les registres distants.

Plusieurs noms pour une signature

Tags disponibles pour MySQL

- `8.1.0`, `8.1`, `8`, `innovation`, `latest`, `8.1.0-oracle`, `8.1-oracle`, `8-oracle`, `innovation-oracle`, `oracle`
- `8.0.34`, `8.0`, `8.0.34-oracle`, `8.0-oracle`
- `8.0.34-debian`, `8.0-debian`
- `5.7.43`, `5.7`, `5`, `5.7.43-oracle`, `5.7-oracle`, `5-oracle`

📷🏷️ Conventions de nommage des images

Pour résumer…

📷🏷️ Conventions de nommage : Exemples

- `ubuntu:22.04` ⇒ `registry.docker.com/library/ubuntu:22.04`
- `dduportal/docker-asciidoctor` ⇒ `registry.docker.com/dduportal/docker-asciidoctor:latest`
- `ghcr.io/dduportal/docker-asciidoctor:1.3.2@sha256:xxxx`

🎓 Utilisons les tags

- Rappel : ⚠️ Friends don't let friends use `latest` 👭
- Il est temps de "taguer" votre première image !

```
docker image tag docker-git:latest docker-git:1.0.0
```

- Testez le fonctionnement avec le nouveau tag
- Comparez les 2 images dans la sortie de `docker image ls`

✅ Utilisons les tags

```
docker image tag docker-git:latest docker-git:1.0.0

# 2 lignes
docker image ls | grep docker-git
# 1 ligne
docker image ls | grep docker-git | grep latest
# 1 ligne
docker image ls | grep docker-git | grep '1.0.0'

# Doit fonctionner
docker container run --rm docker-git:1.0.0 git --version
```

🎓 Mettre à jour votre image (1.1.0)

- Mettez à jour votre image en version `1.1.0` avec les changements suivants :
  - Ajoutez un `LABEL` dont la clef est `description` (et la valeur de votre choix)
  - Configurez `git` pour utiliser une branche `main` par défaut au lieu de `master` (commande `git config --global init.defaultBranch main`)
- Indices :
  - 💡 Commande `docker image inspect <image name>`
  - 💡 Commande `git config --get init.defaultBranch` (dans le conteneur)
  - 💡 Ajoutez des lignes **à la fin** du `Dockerfile`
  - 💡 Documentation de référence des `Dockerfile`

✅ Mettre à jour votre image (1.1.0)

```
cat ./Dockerfile
FROM ubuntu:22.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
LABEL description="Une image contenant git préconfiguré"
RUN git config --global init.defaultBranch main

docker image build -t docker-git:1.1.0 ./docker-git/
# Sending build context to Docker daemon  2.048kB
# Step 1/4 : FROM ubuntu:22.04
#  ---> e40cf56b4be3
# Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git
#  ---> Using cache
#  ---> 926b8d87f128
# Step 3/4 : LABEL description="Une image contenant git préconfiguré"
#  ---> Running in 0695fc62ecc8
# Removing intermediate container 0695fc62ecc8
#  ---> 68c7d4fb8c88
# Step 4/4 : RUN git config --global init.defaultBranch main
#  ---> Running in 7fb54ecf4070
# Removing intermediate container 7fb54ecf4070
#  ---> 2858ff394edb
Successfully built 2858ff394edb
Successfully tagged docker-git:1.1.0

docker container run --rm docker-git:1.0.0 git config --get init.defaultBranch
docker container run --rm docker-git:1.1.0 git config --get init.defaultBranch
# main
```

Cache d'images & Layers

```
Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git
  ---> Using cache
```

🤔 En fait, Docker n'a PAS exécuté cette commande la seconde fois ⇒ ça va beaucoup plus vite !

🎓 Essayez de voir les layers avec (dans Gitpod) `dive <image>:<tag>`

🎓 Cache d'images & Layers

- **But :** manipuler le cache d'images
- Commencez par vérifier que le cache est utilisé : relancez la dernière commande `docker image build` (plusieurs fois s'il le faut)
- Invalidez le cache en ajoutant le paquet APT `make` à installer en même temps que `git`
  - ⚠️ Tag `1.2.0`
- Vérifiez que le cache est bien présent de nouveau

✅ Cache d'images & Layers

```
# Build one time
docker image build -t docker-git:1.1.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.1.0 ./docker-git/

cat Dockerfile
# FROM ubuntu:22.04
# RUN apt-get update && apt-get install --yes --no-install-recommends git make
# LABEL description="Une image contenant git préconfiguré"
# RUN git config --global init.defaultBranch main

# Build one time
docker image build -t docker-git:1.2.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.2.0 ./docker-git/

## Vérification
# Renvoie une erreur
docker run --rm docker-git:1.1.0 make --version
# Doit fonctionner
docker run --rm docker-git:1.2.0 make --version
```

Comportement par défaut des containers

Deux instructions permettent de définir la commande à lancer au démarrage du container.

Comportement par défaut des containers

Deux instructions permettent de définir la commande à lancer au démarrage du container.

CMD

ENTRYPOINT

CMD vs ENTRYPOINT

Cas d’usage : énoncé du besoin.

CMD vs ENTRYPOINT

Cas d'usage. On va s'outiller!

CMD vs ENTRYPOINT

Cas d’usage.

CMD vs ENTRYPOINT

Cas d'usage.

```
$ docker container run -it my-curl:1.0 sh
/ $
```

override du CMD

CMD vs ENTRYPOINT

Cas d'usage: un cran plus loin.

L'image actuelle, c'est bien mais pas hyper flexible !

CMD vs ENTRYPOINT

Cas d'usage, un cran plus loin

Paramétrable, et hop!

CMD vs ENTRYPOINT

Cas d'usage, un cran plus loin

Paramétrable, et hop!

```
FROM alpine:3.23
LABEL maintainer="John Doe"
RUN apk update && apk add curl
# Si vous utilisez le proxy de l'Université
ENTRYPOINT ["curl","-x", "http://prx:3128", "-L", "--conne
# Si vous utilisez votre propre proxy docker
# CMD ["curl","-x", "http://host.docker.internal:3128", "-
```

```
docker image build -t my-curl:2.0 .
```

```
docker container run --rm my-curl:2.0 http
<!doctype html><html [...]
google blahblahblah [...]
</html>
```

En présence d'un ENTRYPOINT, l'override du CMD vient se concaténer et devient ainsi un paramètre !

Checkpoint 🎯

- Une image Docker fournit un environnement de système de fichier auto-suffisant (application, dépendances, binaries, etc.) comme modèle de base d'un conteneur
- Les images Docker ont une convention de nommage permettant d'identifier les images très précisément
- On peut spécifier une recette de fabrication d'image à l'aide d'un `Dockerfile` et de la commande `docker image build`

⇒ 🤔 et si on utilisait Docker pour nous aider dans l'intégration continue ?

Nos fichiers dans les images

📂 Le répertoire de travail

WORKDIR

```
FROM alpine:3.23
WORKDIR /repertoire/travail
RUN echo hello > ./world.txt
```

📂 WORKDIR

```
FROM alpine:3.23
WORKDIR /repertoire/travail
RUN echo hello > ./world.txt
```

```
$ docker image build --tag mon-img .
...
$ docker container run mon-img cat
/repertoire/travail/world.txt
hello
$ docker container run mon-img pwd
/repertoire/travail
$ docker container run --workdir /home mon-img pwd
/home
```

📂 WORKDIR

```
FROM alpine:3.23
WORKDIR /repertoire/travail
RUN echo hello > ./world.txt
```

```
$ docker image build --tag mon-img .
...
$ docker container run mon-img cat
/repertoire/travail/world.txt
hello
$ docker container run mon-img pwd
/repertoire/travail
$ docker container run --workdir /home mon-img pwd
/home
```

override du WORKDIR

WORKDIR

```
FROM alpine:3.23
WORKDIR /repertoire/travail
RUN echo hello > ./world.txt
```

```
docker image build --tag mon-img .
...
```

```
$ docker container run mon-img cat /repertoire/travail/world.txt
hello
```

```
$ docker container run mon-img pwd
/repertoire/travail
```

```
$ docker container run --workdir /home mon-img pwd
/home
```

📁 Embarquer des fichiers

Cas d'usage.

📁 Embarquer des fichiers

Copie de fichiers.

📁 Embarquer des fichiers

Le mot clé `ADD`.

```
FROM image
ADD https://mon-nexus/foo/bar/1.0/package.tar /uncompressed/
```

📁 Embarquer des fichiers

Le mot clé `ADD`

source : https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#add-or-copy

Because image size matters, using `ADD` to fetch packages from remote URLs is strongly discouraged; you should use `curl` or `wget` instead. That way you can delete the files you no longer need after they've been extracted and you don't have to add another layer in your image. For example, you should avoid doing things like:

```
ADD https://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all
```

And instead, do something like:

```
RUN mkdir -p /usr/src/things \
    && curl -SL https://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

For other items (files, directories) that do not require `ADD` 's tar auto-extraction capability, you should always use `COPY` .

📁 Embarquer des fichiers

Le mot clé `ADD`.

source : https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#add-or-copy

Because image size matters, using `ADD` to fetch packages from remote URLs is strongly discouraged; you should use `curl` or `wget` instead. That way you can delete the files you no longer need after they've been extracted and you don't have to add another layer in your image. For example, you should avoid doing things like:

```
ADD https://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all
```

And instead, do something like:

```
RUN mkdir -p /usr/src/things \
    && curl -SL https://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

📂 Embarquer des fichiers

Oui, mais pas la terre entière !

```
docker image build --tag myjava:1.42 ./
Sending build context to Docker daemon  172.8MB
```

📁 Embarquer des fichiers

Oui, mais pas la terre entière !

```
$ docker image build --tag myjava:1.42 ./
Sending build context to Docker daemon  172.8MB
```

Répertoire contenant le Dockerfile.
NB : tous les fichiers présents dans ce répertoire seront envoyés au Docker daemon pour construire vos images !

📁 Embarquer des fichiers

Oui, mais pas la terre entière !

```
$ docker image build --tag myjava:1.42 ./
Sending build context to Docker daemon  172.8MB
```

Répertoire contenant le Dockerfile.
NB : tous les fichiers présents dans ce répertoire seront envoyés au Docker daemon pour construire vos images !

?!!!

📁 Embarquer des fichiers

Oui, mais pas la terre entière !

```
$ docker image build --tag myjava:1.42 ./
Sending build context to Docker daemon  172.8MB
```

Répertoire contenant le Dockerfile.
NB : tous les fichiers présents dans ce répertoire seront envoyés au Docker daemon pour construire vos images !

?!!!

Si un gros fichier traîne dans le dossier alors qu'il n'est même pas utilisé ni référencé dans le Dockerfile, il sera tout de même envoyé au Docker daemon.

📁 Embarquer des fichiers

Oui, mais pas la terre entière !

🙈 .dockerignore

```
# ignore les dossiers .git et .cache
.git
.cache
```

Commande	Usage
FROM	Pour spécifier l'image à partir de laquelle on construit la nouvelle image
LABEL	Pour décrire l'image à partir de clé/valeur
RUN	Pour exécuter une action au moment du build
ENV	Pour insérer une variable d'environnement dans tous les futurs containers de cette image
ENTRYPOINT	Pour définir une commande à lancer au démarrage du container
CMD	Pour définir une commande à lancer au démarrage du container ou pour compléter un ENTRYPOINT existant
COPY	Pour insérer des fichiers dans l'image

📝 Renommage des images

```
$ docker tag 0e5574283393 fedora/httpd:version-1.0
```

📝 Renommage des images

```
$ docker tag 0e5574283393 fedora/httpd:version-1.0
```

ID existant

📝 Renommage des images

```
$ docker tag 0e5574283393 fedora/httpd:version-1.0
```

ID existant

tag à ajouter

📝 Renommage des images

```
$ docker tag 0e5574283393 fedora/httpd:version-1.0
```

ID existant

tag à ajouter

```
$ docker tag httpd fedora/httpd:version-1.0
```

🖼️ LES IMAGES

Le bilan

Vous savez désormais:

- Rédiger un Dockerfile
- Nommer vos images
- Créer vos outils

🖼️ LES IMAGES

Le bilan

Vous savez désormais:

- Rédiger un Dockerfile
- Nommer vos images
- Créer vos outils

🗂️ Les registries

Golden Rule

La construction d’une image doit être automatisée.

🤖 = 🔒 ? Automatique == sécurisé

Le fait de déléguer la construction des images permet d'ajouter toute une chaîne de traitement, de contrôles des images pour s'assurer qu'elle respecte les règles RSSI.

Exemple de mise en place

Sources

TEST

PROD

Exemple de mise en place

Sources

TEST

PROD

1

Un automate récupère les sources de l'image, la construit et la pousse sur une registry de test

Exemple de mise en place

1

Un automate récupère les sources de l'image, la construit et la pousse sur une registry de test

Exemple de mise en place

1

Un automate récupère les sources de l'image, la construit et la pousse sur une registry de test

2

Un automate récupère l'image sur la registry de test, effectue les contrôles et si elle est valide, la pousse sur la registry de prod

Exemple de mise en place

1

Un automate récupère les sources de l'image, la construit et la pousse sur une registry de test

2

Un automate récupère l'image sur la registry de test, effectue les contrôles et si elle est valide, la pousse sur la registry de prod

➡️ Exemple : continuous build

🎓 Travaux pratiques #5

https://gitlab.univ-artois.fr/bruno.verachten/devops-docker-tp05

✅ Solution Travaux pratiques #5

```diff
diff --git a/src/Dockerfile b/src/Dockerfile
index c92c67cd48121705628f67a2af14b2a65e54cdfd..77919a581fffde7c57f43c8b7cbee2a8d84b628b 100644
--- a/src/Dockerfile
+++ b/src/Dockerfile
@@ -9,6 +9,10 @@ COPY start.sh /bin/start.sh
 EXPOSE 4321
 # This line opens port 4321 on the container. It's like installing a door in your house.

+RUN adduser --disabled-password --gecos "" --home /home/www www
+USER www
+WORKDIR /home/www
+
 ENTRYPOINT ["/bin/start.sh"]
 # This line sets the command that will be run when the container starts. It's like setting the alarm clock in your house
```

✅ Solution Travaux pratiques #5

```
diff --git a/src/start.sh b/src/start.sh
index 44006d6cd8414706a5fcf564dbfd1e9c38d4bc0b..a6ed60fdf88759d4962685e7ceb6cdbc21867448 100755
--- a/src/start.sh
+++ b/src/start.sh
@@ -4,7 +4,7 @@
 set -eux
 # These are shell options. 'e' exits on error, 'u' treats unset variables as an error, and 'x' prints each command to th

-touch /home/www/started.time
+touch /tmp/started.time
 # This creates a file named 'started.time' in the '/home/www' directory. It's like the script's way of marking its terri

 if [ $? -ne 0 ]; then
@@ -12,7 +12,7 @@ if [ $? -ne 0 ]; then
 fi
 # This checks the exit status of the last command. If it's not zero, which means there was an error, it exits the script

-date > /home/www/started.time
+date > /tmp/started.time
 # This writes the current date and time to the 'started.time' file. It's like the script's way of keeping a diary.

 exec "$@"
```

✅ Solution Travaux pratiques #5

```yaml
stages:
  - "validator.sh"
  - "docker scout"

variables:
  PROXY: "http://cache-etu.univ-artois.fr:3128"
  IMAGE: "devops-docker-tp05:latest"

validator-job:
  image: cache-ili.univ-artois.fr/proxy_cache/library/docker
  stage: "validator.sh"
  before_script:
    - export HTTP_PROXY="$PROXY"
    - export HTTPS_PROXY="$PROXY"
    - apk add -U bash
  script:
    - errors=$(./validator.sh)
    - echo "$errors"
    - exit $([ -z "$errors" ] && echo 0 || echo 1)
  tags:
    - docker2

# No login needed, using docker desktop with wsl
scout-job:
  stage: "docker scout"
  script:
    - cd ./src
    - docker build . -t "$IMAGE"
    - docker scout cves --format only-packages --only-vuln-packages "$IMAGE"
```

✅ Solution Travaux pratiques #5

```yaml
stages:
  - run_script
  - docker_scan

run_script:
  stage: run_script
  script:
    - chmod +x validator.sh src/start.sh && ./validator.sh # Makes the scripts executable and runs validator.sh
    - |
        if [ $? -eq 0 ]; then
          echo "Script exited successfully."
        else
          echo "Script exited with an error."
          exit 1  # Mark the job as a failure
        fi

docker_scan:
  stage: docker_scan
  script:
    - IMG=$(echo img$$)
    - docker image build --tag $IMG ./src > /dev/null
    - echo "Will scan $IMG"
    - echo $DOCKER_TOKEN | docker login -u $DOCKER_USERNAME --password-stdin
    - docker scout cves --format only-packages --only-vuln-packages $IMG # Runs docker scout to scan the image
    - docker scout recommendations $IMG # View base image update recommendations
```

🔍 Inspection et nommage des containers

Nommage des containers

```
$ docker container logs 47d6
Tue Oct 2400:39:52 UTC 2023
Tue Oct 2400:39:53 UTC 2023
...
```

🚫 Nommage des containers

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
793906a4c2d2        centos              "/bin/bash"         3 hours ago         Up 3 hours
festive_poitras
```

Nommage des containers

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
793906a4c2d2        centos              "/bin/bash"         3 hours ago         Up 3 hours
festive_poitras
```

Une première alternative !

Nommage des containers

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
793906a4c2d2        centos              "/bin/bash"         3 hours ago         Up 3 hours
festive poitras
```

Une première alternative !

Par défaut, Docker génère un nom en combinant une humeur et un inventeur.

Nommage des containers

Humour de g33k

```go
// GetRandomName generates a random name from the list of adjectives and surnames in this package
// formatted as "adjective_surname". For example 'focused_turing'. If retry is non-zero, a random
// integer between 0 and 10 will be added to the end of the name, e.g `focused_turing3`
func GetRandomName(retry int) string {
begin:
	name := fmt.Sprintf("%s_%s", left[rand.Intn(len(left))], right[rand.Intn(len(right))])
	if name == "boring_wozniak" /* Steve Wozniak is not boring */ {
		goto begin
	}

	if retry > 0 {
		name = fmt.Sprintf("%s%d", name, rand.Intn(10))
	}
	return name
}
```

source : https://github.com/docker/engine/blob/master/pkg/namesgenerator/names-generator.go

Nommage des containers

```
docker container run --detach --name my-web httpd
```

🦸 Renommage des containers

Attention, spoiler pour les n00bs de DC Comics

🦸 Renommage des containers

```
$ docker container rename clark_kent superman
```

attention, spoiler pour les n00bs de DC Comics

ancien nom

🦸 Renommage des containers

```
$ docker container rename clark_kent superman
```

attention, spoiler pour les n00bs de DC Comics

ancien nom

nouveau nom

🔍 Inspection des containers

```
docker container inspect my-web
[
    {
        "Id": "0ac8cdf8d447c6d316a04bd1a7f74cd2677eea3478f11f0be5241c1bb2d4c7da",
        "Created": "2023-10-24T19:40:11.84788911Z",
        "Path": "httpd-foreground",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3275,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-10-24T19:40:12.363841649Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
...
]
```

Comment exploiter le JSON ?

JQ est le meilleur outil pour parser du JSON dans le shell

🎓 Travaux pratiques #6

Étapes:

✅ Solution Travaux Pratiques #6

On démarre un container nginx avec la commande

```
# Here we're using the 'docker container run' command to start a new Docker container.
# The '--detach' option tells Docker to run the container in the background and print the container ID.
# The '--name' option allows us to give our container a custom name, in this case 'my-web'.
# Finally, 'nginx' is the name of the Docker image we want to use to create the container.
# So, to sum up, this command will start a new Docker container in the background, using the 'nginx' image, and name it '
docker container run --detach --name my-web nginx
```

✅ Solution Travaux Pratiques #6

```
[
  "nginx",
  "-g",
  "daemon off;"
]
```

Qu'on pourrait nettoyer pour avoir `nginx -g daemon off;` avec:

✅ Solution Travaux Pratiques #6

```
# This command is using 'docker container ls' to list our Docker containers. The '--filter' option allows us to only show
# We then pipe this output to 'awk', a powerful text processing tool. 'NR==1 {cmd=index($0, "COMMAND"); created=index($0,
# 'NR>1 {print substr($0, cmd, created-cmd)}' then tells 'awk' to print the substring from the start of the 'COMMAND' fie
# So, to sum up, this command will give us the command used to start the 'my-web' container, extracted from the output of
docker container ls --filter "name=my-web" --no-trunc | awk 'NR==1 {cmd=index($0, "COMMAND"); created=index($0, "CREATED"
```

Trouver son container en réseau

Le Container en réseau

```
docker container run --detach nginx
bd12c4d7110d17ce80...`
```

```
docker container ls
CONTAINERID     IMAGE   COMMAND      CREATED    STATUS    PORTS
bd12c4d71       nginx   "nginx …"    35 s. ago  Up 33 s.  80/tcp, 443/tcp
```

🔍 Que nous dit docker container inspect ?

```
"Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "c0fcec43e3e8…eecd6558ac0870a468a3",
                    "EndpointID": "0ec8fb237a10e9227359b4…db23edc32",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
```

🔍 Que nous dit docker container inspect ?

```
"Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "c0fcec43e3e8…eecd6558ac0870a468a3",
                    "EndpointID": "0ec8fb237a10e9227359b4…db23edc32",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
```

🔍 Que nous dit docker container inspect ?

```
"Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "c0fcec43e3e8…eecd6558ac0870a468a3",
                    "EndpointID": "0ec8fb237a10e9227359b4…db23edc32",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
```

Et voilà !

```
curl -I --noproxy '*' http://172.17.0.2:80
HTTP/1.1 200 OK
....
`Server: nginx/1.25 ...
`
```

Il est maintenant facile d'interagir avec notre serveur web !

🗺️ Mapping de Port

```
$ docker container run --detach   -p 8000:80
nginx
```

le port de la machine hôte

Mapping de Port

```
$ docker container run --detach   -p 8000:80 nginx
```

le port de la machine hôte

le port du container

🗺 Mapping de Port

```
$ docker container run --detach   -p 8000:80 nginx
```

le port de la machine hôte

le port du container

```
$ curl -I --noproxy '*' http://172.17.0.2:80
it works !
$ curl -I --noproxy '*' http://localhost:8000
it works !
```

🗺️ Mapping de Port

```
curl -I --noproxy '*' http://172.17.0.2:80
HTTP/1.1 200 OK
Server: nginx/1.25.2
Date: Wed, 18 Oct 2023 11:54:49 GMT
Content-Type: text/html
Content-Length: 284237
Last-Modified: Tue, 10 Oct 2023 12:12:13 GMT
Connection: keep-alive
ETag: "65253f9d-4564d"
Accept-Ranges: bytes
```

🎓 Travaux pratiques #7

Étapes:

✅ Solution travaux pratiques #7

```
<html><head></head><body><header>
<title>http://info.cern.ch</title>
</header>

<h1>http://info.cern.ch - home of the first website</h1>
<p>From here you can:</p>
<ul>
<li><a href="http://info.cern.ch/hypertext/WWW/TheProject.html">Browse the first website</a></li>
<li><a href="http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html">Browse the first website using the line-mode bro
<li><a href="http://home.web.cern.ch/topics/birth-web">Learn about the birth of the web</a></li>
<li><a href="http://home.web.cern.ch/about">Learn about CERN, the physics laboratory where the web was born</a></li>
</ul>
</body></html>
```

✅ Solution travaux pratiques #7

```
wget https://gitlab.univ-artois.fr/bruno.verachten/devops-docker-tp07/-/raw/master/sample.war
```

✅ Solution travaux pratiques #7

```
git clone https://github.com/spring-guides/gs-spring-boot.git
```

C’est un début…

✅ Solution travaux pratiques #7

Assemblons tout ça dans un `Dockerfile`.

```
# Use a Maven image for building the application
FROM maven:3.9.11-eclipse-temurin-25-alpine
WORKDIR /app

# Clone the Spring Boot application repository
RUN apk add git && git clone https://github.com/spring-guides/gs-spring-boot.git

# Change directory to the application's complete directory
WORKDIR /app/gs-spring-boot/complete

# Build the application using Maven
RUN mvn package

WORKDIR /app/gs-spring-boot/complete/target

# Command to run the Spring Boot application
CMD ["java", "-jar", "spring-boot-complete-0.0.1-SNAPSHOT.jar"]
```

⏳ Attends un peu…

Ça fonctionne, mais c'est comme si je laissais la grue dans la chambre une fois que j'avais fini de construire ma maison !

✅ Solution travaux pratiques #7

On essaye un `Dockerfile` moins BTP?

Une première solution

Avec un autre exemple…Une image pour construire l’appli:

Une première solution

Avec un autre exemple

```
#!/bin/sh
echo Building alexellis2/href-counter:build

docker image build --build-arg https_proxy=$https_proxy --build-arg http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build
docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract

echo Building alexellis2/href-counter:latest
docker image build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```

🧩 Une première solution

Avec un autre exemple

build.sh

```
#!/bin/sh
echo Building alexellis2/href-counter:build

docker image build --build-arg https_proxy=$https_proxy --build-arg
http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build

docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract

echo Building alexellis2/href-counter:latest

docker image build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```

on crée l'image de construction

🧩 Une première solution

Avec un autre exemple

build.sh

```
#!/bin/sh
echo Building alexellis2/href-counter:build
```

on crée l'image de construction

```
docker image build --build-arg https_proxy=$https_proxy --build-arg
http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build
```

on fait un container pour récupérer le build

```
docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract
```

```
echo Building alexellis2/href-counter:latest

docker image build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```

🧩 Une première solution

Avec un autre exemple

build.sh

```
#!/bin/sh
echo Building alexellis2/href-counter:build
```

on crée l'image de construction

```
docker image build --build-arg https_proxy=$https_proxy --build-arg
http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build
```

on fait un container pour récupérer le build

```
docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract
```

```
echo Building alexellis2/href-counter:latest
```

on crée l'image d'exécution

```
docker image build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```

Une première solution

Avec un autre exemple

build.sh

```
#!/bin/sh
echo Building alexellis2/href-counter:build
```

on crée l'image de construction

```
docker image build --build-arg https_proxy=$https_proxy --build-arg
http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build
```

on fait un container pour récupérer le build

```
docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract
```

```
echo Building alexellis2/href-counter:latest
```

on crée l'image d'exécution

```
docker image build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```

🪜 Multistage build

```
FROM golang:1.19

WORKDIR /go/src/github.com/alexellis/href-counter/

COPY go.mod .
COPY go.sum .
COPY app.go     .

RUN CGO_ENABLED=0 GOOS=linux go build  -ldflags "-s -w" -o app .

FROM alpine:3.17.1
RUN apk --no-cache add ca-certificates

WORKDIR /root/

COPY --from=0 /go/src/github.com/alexellis/href-counter/app    .

CMD ["./app"]
```

🪜 Multistage build

```
FROM golang:1.19

WORKDIR /go/src/github.com/alexellis/href-counter/

COPY go.mod .
COPY go.sum .
COPY app.go     .

RUN CGO_ENABLED=0 GOOS=linux go build  -ldflags "-s -w" -o app .

FROM alpine:3.17.1
RUN apk --no-cache add ca-certificates

WORKDIR /root/

COPY --from=0 /go/src/github.com/alexellis/href-counter/app    .

CMD ["./app"]
```

On copie les fichiers depuis le premier stage

📦 LES CONTAINERS

Le bilan : achievement unlocked

Vous savez désormais:

Les volumes

3 types de gestion de données

Mapping de FS

Volumes

FS en RAM

🔴🔵🟢 3 types de gestion de données

Mapping de FS

Partage d'un répertoire de votre système hôte avec un conteneur Docker.

🔴🔵🟢 3 types de gestion de données

Volumes

Stockage persistant ⇒ données en dehors du système de fichiers du conteneur.

🔴🔵🟢 3 types de gestion de données

FS en RAM

Système de fichiers temporaire stocké en RAM.

🔴🔵🟢 3 types de gestion de données

Mapping de FileSystem

Mapping de FileSystem

"Créer une image avec les sources de mon application web juste pour distribuer des ressources statiques via Apache ! Merci Docker !"

Un étudiant excédé

🗺️📂 Mapping de FileSystem

Hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Source : https://hub.docker.com/_/nginx

Mapping de FileSystem

Partage de dossier

Les volumes

Host
/
var
lib
docker
volumes
some-data
myvolume
data
content
dir
Container
/
content
dir

Les volumes

Il est possible de créer des "espaces mémoire" que l’on peut mettre à disposition des containers.

Les volumes

Lister tous les volumes

```
docker volume ls
```

📦 📁 🕶️ Les volumes "anonymes"

```
FROM alpine:3.23
WORKDIR /repertoire/travail
VOLUME ["/data"]
RUN echo hello > ./world.txt
```

💾🧠 FileSystem en RAM

```
docker container run
  -d
  --read-only
  --tmpfs /run/httpd
  --tmpfs /tmp
  httpd
```

💾🧠 FileSystem en RAM

```
$ docker container run
      -d
      --read-only
      --tmpfs /run/httpd
      --tmpfs /tmp
      httpd
```

Le container ne peut pas écrire sur le FS

🔥 Les données sont perdues à l'arrêt du container.

💾🧠 FileSystem en RAM

```
$ docker container run
      -d
      --read-only
      --tmpfs /run/httpd
      --tmpfs /tmp
      httpd
```

Le container ne peut pas écrire sur le FS

Sauf là et là

🔥 Les données sont perdues à l'arrêt du container.

📊📝 Bilan de compétences

- Création d'images
- Création des containers
  - cycle de vie
  - nommage
  - débug
  - réseau
  - volumes

📊📝 Bilan de compétences

- Création d'images ✅
- Création des containers ✅
  - cycle de vie ✅
  - nommage ✅
  - débug ✅
  - réseau ✅
  - volumes ✅

🎓 Travaux pratiques #8

Lancer un container `nginx `et lui faire distribuer une page web externe au container.

✅ Solution travaux pratiques #8

En utilisant un montage de répertoire (Bind Mount) :

✅ Solution travaux pratiques #8

En utilisant un volume nommé (Named Volume) :

🎓 Travaux pratiques #9

Étapes:

✅ Solution travaux pratiques #9

Créer un volume nommé "data" :

✅ Solution travaux pratiques #9

Démarrer un second conteneur attaché au même volume :

```
docker container run -it --name second-container -v data:/donnees busybox
```

Dans le second conteneur, lisez les données du volume (le fichier texte) :

🎓 Travaux pratiques #10

Écrire un Dockerfile qui:

✅ Solution travaux pratiques #10

Tout d'abord, créez un répertoire pour votre projet et placez-vous dedans.

✅ Solution travaux pratiques #10

```
docker image build -t mon-image .
```

Assurez-vous que "mon-image" est le nom que vous souhaitez donner à votre image Docker. Après avoir créé l’image, vous pouvez exécuter un conteneur basé sur cette image :

✅ Solution travaux pratiques #10

```
cat *
```

```
Contenu du fichier 1
Contenu du fichier 2
Nouveau contenu du fichier 3
```

Réseaux et Docker

Interagir avec le Docker ENGINE

🤝 Interagir avec le Docker Engine

🤝 Interagir avec le Docker Engine

🤝 Interagir avec le Docker Engine

🤝 Interagir avec le Docker Engine

🤝 Interagir avec le Docker Engine

Ex : intégration IntelliJ

cours-devops-docker · volumes

Project

- cours-devops-docker C:\Support\users\Talks\fac\2024\cours
  - .github
    - workflows
      - build-workflow.yml
    - dependabot.yml
  - .idea
  - assets
  - content
    - chapitres
      - sous-chapitres
      - bibliographie.adoc
      - bonus.adoc
      - cas-d-usage.adoc
      - cli.adoc
      - compose.adoc
      - dev-env.adoc
      - docker-base.adoc
      - fichiers-nommage-inspect.adoc
      - git-base.adoc
      - images.adoc
      - intro.adoc

index.adoc | volumes.adoc | fichiers-nommage-inspect.adoc | Dockerfile | welcome.adoc | images.adoc | .env | attributes.adoc | Docker_m2_ILI_2022_part1_cont

Toggle Title · PDF · DOCX · HTML

réseau — 2 results

```
[%auto-animate]
== Bilan de compétences

* Création d'images ✅
* Création des containers ✅
** cycle de vie ✅
** nommage ✅
** débug ✅
** réseau ✅
** volumes ✅

== Travaux pratiques #8

Lancer un container `nginx `et lui faire distribuer une page web externe au container.

image::1763880002-image10.png[]

[%auto-animate]
```

Les volumes › Bilan de compétences › (List) › Création des containers › (List) › débug

Services

- Docker
  - Docker-compose: cours-devops-docker
    - qrcode
    - serve
    - cours-devops-docker_default
  - Docker-compose: jenkins
    - adb healthy
    - adb-connector
    - android-agent
    - emulator healthy
    - jenkins
    - jenkins-agent
    - rethinkdb
    - stf
    - jenkins_default
  - Containers
    - buildx_buildkit_my_builder_10
    - unruffled_hodgkin
  - Images
  - Networks
    - bridge
    - host
    - none
  - Volumes
    - 0b5d51da6b75286fa460cea8bee33f77856ac2a1a83857917d0672fa0bdb56d6
    - 0c7f3f9421532b4c9797ad1b6e55d8b057b59764688c5a2b7ee24a9f6f8bb66f
    - 0c65e5bebaa2c241e08d9295c6e5367a1fe484d16d013b412642244465a47c0c
    - 0ca44cf218e769c4dd4e6bd39093b5a38ac4edff6dcf65a1f4137d017be4cc7b
    - 0d2f577314a4dcfb54d7550be3ee607a64336c63e5567d19ecc12f87e7e1f235

Log · Dashboard

oby/buildkit:buildx-stable-1

```
msg="auto snapshotter: using overlayfs"
ng msg="using host network as the default"
msg="found worker \"xqtbwe7o5czz1jxn80zs3en76\", labels=map[org.mobyp
time="2023-10-19T12:24:28Z" level=warning msg="skipping containerd worker, as \"/run/containerd/containerd.s
time="2023-10-19T12:24:28Z" level=info msg="found 1 workers, default=\"xqtbwe7o5czz1jxn80zs3en76\""
time="2023-10-19T12:24:28Z" level=warning msg="currently, only the default worker can be used."
time="2023-10-19T12:24:28Z" level=info msg="running server on /run/buildkit/buildkitd.sock"
time="2023-10-19T13:05:26Z" level=info msg="trying next host" error="error getting credentials - err: exit s
time="2023-10-19T13:05:26Z" level=error msg="/moby.buildkit.v1.frontend.LLBBridge/Solve returned error: rpc
time="2023-10-19T13:05:26Z" level=error msg="/moby.buildkit.v1.Control/Solve returned error: rpc error: code
time="2023-10-19T13:06:33Z" level=info msg="trying next host" error="error getting credentials - err: exit s
time="2023-10-19T13:06:33Z" level=error msg="/moby.buildkit.v1.frontend.LLBBridge/Solve returned error: rpc
time="2023-10-19T13:06:33Z" level=error msg="/moby.buildkit.v1.Control/Solve returned error: rpc error: code
time="2023-10-19T13:08:16Z" level=info msg="trying next host" error="error getting credentials - err: exit s
time="2023-10-19T13:08:16Z" level=error msg="/moby.buildkit.v1.frontend.LLBBridge/Solve returned error: rpc
time="2023-10-19T13:08:16Z" level=error msg="/moby.buildkit.v1.Control/Solve returned error: rpc error: code
time="2023-10-19T13:20:19Z" level=error msg="failed to kill process in container id ssp4i449gdfl1rsayknecczj
time="2023-10-19T13:20:19Z" level=error msg="/moby.buildkit.v1.Control/Solve returned error: rpc error: code
time="2023-10-19T13:20:19Z" level=error msg="/moby.buildkit.v1.Control/Solve returned error: rpc error: code
time="2023-10-20T12:36:49Z" level=info msg="auto snapshotter: using overlayfs"
time="2023-10-20T12:36:49Z" level=warning msg="using host network as the default"
time="2023-10-20T12:36:50Z" level=info msg="found worker \"xqtbwe7o5czz1jxn80zs3en76\", labels=map[org.mobyp
time="2023-10-20T12:36:50Z" level=warning msg="skipping containerd worker, as \"/run/containerd/containerd.s
time="2023-10-20T12:36:50Z" level=info msg="found 1 workers, default=\"xqtbwe7o5czz1jxn80zs3en76\""
time="2023-10-20T12:36:50Z" level=warning msg="currently, only the default worker can be used."
time="2023-10-20T12:36:50Z" level=info msg="running server on /run/buildkit/buildkitd.sock"
```

cours-devops-docker › content › chapitres › fichiers-nommage-inspect.adoc

14°C Ciel couvert · Search

🤝 Interagir avec le Docker Engine

Sauf que…

Les réseaux de containers

🗺️🔀 Mapping de Port : Rappel

```
docker container run -d -p 8000:80 nginx
```

🗺️🔀 Mapping de Port : Rappel

```
$ docker container run -d -p 8000:80 nginx
```

le port de la machine hôte

🗺️🔀 Mapping de Port : Rappel

```
$ docker container run -d -p 8000:80 nginx
```

le port de la machine hôte

le port du container

Les réseaux

```
docker network ls
```

Les réseaux

```
docker network ls
```

```
NETWORK ID     NAME                                                      DRIVER    SCOPE
11b7af7b16e4   bridge                                                    bridge    local
1112832d205b   cours-devops-docker_default                               bridge    local
7ab15cc28199   docker_volumes-backup-extension-desktop-extension_default bridge    local
34caf4674478   host                                                      host      local
2a179c7be3b3   none                                                      null      local
0c577a792771   portainer_portainer-docker-extension-desktop-extension_default bridge    local
```

Bridge	Host	Null
Mode par défaut. C'est un sous réseau dans lequel les containers viennent s'attacher.	Le container sera attaché au réseau de la machine hôte.	Le container ne sera attaché à aucun réseau.

🔍🌐 Les réseaux : inspection

```
docker network inspect bridge
```

🌐✨ Les réseaux : création

```
docker network create mon-reseau
```

🐳🔗 Attacher un container à un réseau particulier

```
docker run -d --net=mon-reseau --name=app img
```

🖥️🔍 MicroDNS

```
docker network create mynet
```

🖥️🔍 MicroDNS

On peut appeler le container voisin par son nom !

```
$ docker network create mynet

$ docker container run -d --name web --net mynet nginx

$ docker container run --net mynet alpine ping web
PING nginx (172.18.0.2) 56(84) bytes of data.
64 bytes from web.mynet (172.18.0.2): icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=2 ttl=64 time=0.136 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=3 ttl=64 time=0.137 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=4 ttl=64 time=0.124 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=5 ttl=64 time=0.108 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=6 ttl=64 time=0.056 ms
64 bytes from web.mynet (172.18.0.2): icmp_seq=7 ttl=64 time=0.052 ms
```

🚨 Piège NGINX : Résolution DNS dans upstream blocks

Le Problème

NGINX, contrairement à d'autres outils (curl, wget, applications), **résout les noms DNS au démarrage uniquement**.

Exemple de Configuration Défaillante

```
events {
    worker_connections 1024;
}

http {
    # ❌ AUCUNE directive resolver !

    upstream backend {
        server app-server:8080;  # ❌ ÉCHEC au démarrage !
    }
```

📶🔌 Se connecter à un réseau

```
docker network connect mynet myapp
```

🎓 Travaux pratiques #11

Étapes:

✅ Solution travaux pratiques #11

Étape 1 : Lister et inspecter les réseaux existants

```
# Liste tous les réseaux Docker existants
docker network ls

# Inspecte un réseau spécifique (par exemple, le réseau bridge, d'autres réseaux peuvent être inspectés)
docker network inspect bridge
```

✅ Solution travaux pratiques #11

Étape 3 : Vérifier le réseau auquel "web1" est attaché

```
# Récupère l'identifiant de réseau complet pour le container "web1"
network_id=$(docker container inspect -f '{{.NetworkSettings.Networks.bridge.NetworkID}}' web1)

# Raccourcit l'identifiant du réseau aux premiers 12 caractères
shortened_network_id=$(echo $network_id | cut -c 1-12)

# Liste tous les réseaux docker, en filtrant par l'identifiant du réseau
docker network ls --format "table {{.ID}}\t{{.Name}}" | awk -v network_id="$shortened_network_id" '$1 == network_id {prin
```

```
bridge
```

✅ Solution travaux pratiques #11

Étape 5 : Lancer un container "nginx" nommé "web2" et l'attacher au réseau créé

```
# Lance un conteneur "nginx" nommé "web2" et l'attache au réseau "mon-reseau"
docker container run --name web2 -d --network mon-reseau nginx
```

Étape 6 : Vérifier l'attachement du container "web2" au réseau

✅ Solution travaux pratiques #11

Étape 9 : Corriger pour permettre la communication entre "web1" et "web2"

🚨 Piège : localhost vs nom de service dans Docker

La Confusion Fatale

Terme	Hors Docker (machine locale)	Dans Docker (environnement conteneurisé)
`localhost`	Votre ordinateur	**Le conteneur actuel uniquement**
`127.0.0.1`	Votre ordinateur	**Le conteneur actuel uniquement**
`nom_service`	*(n'existe pas)*	**Un autre conteneur du même réseau**
`nom_service:port`	*(n'existe pas)*	**Communication inter-conteneurs**

⚠ **Dans Docker :** `localhost` / autre conteneur

🐳🐙 Docker compose

🌐🏡 Un cas de la vraie vie

Une application riche repose souvent sur plusieurs éléments techniques à coordonner ensemble.

(Apache, Tomcat, Node, MongoDB, ElasticSearch, Logstash, etc.)

Comment automatiser ces déploiements ?

📜💻 On pourrait tout scripter…

```
#!/bin/sh
docker network create my-net
docker container run -d --net=my-net -p 3306:3306 mysql
docker container run -d --net=my-net -v /docs:/docs --name col1 httpd:2.4.58-bookworm
docker container run -d --net=my-net -v /docs:/docs --name col2 httpd:2.4.58-bookworm
```

📜💻 Tout scripter

🐳🐙 docker compose.yml

```
services:
```

🐳🐙 docker compose.yml

```
services:
  apache:
  middle:
  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"



  middle:



  db:
```

🐳🐙 docker compose.yml

```
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"



  middle:



  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle


  middle:



  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle
    volumes:
      - ./etc/httpd/workers.properties:/etc/httpd/conf/workers.properties
  middle:



  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle
    volumes:
      - ./etc/httpd/workers.properties:/etc/httpd/conf/workers.properties
  middle:
    build: .


  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle
    volumes:
      - ./etc/httpd/workers.properties:/etc/httpd/conf/workers.properties
  middle:
    build: .
    environment:
      - DB_HOST=db
  db:
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle
    volumes:
      - ./etc/httpd/workers.properties:/etc/httpd/conf/workers.properties
  middle:
    build: .
    environment:
      - DB_HOST=db
  db:
    image: "mysql:5.6"
```

🐳🐙 docker compose.yml

```yaml
services:
  apache:
    image: "my-httpd:2.4"
    ports:
      - "80:80"
    environment:
      - MIDDLE_HOST_1=middle
    volumes:
      - ./etc/httpd/workers.properties:/etc/httpd/conf/workers.properties
  middle:
    build: .
    environment:
      - DB_HOST=db
  db:
    image: "mysql:5.6"
    ports:
      - "3306:3306"
```

🤔➡️ Et après ?

```
docker compose up -d
docker compose build
docker compose logs
docker compose stop
docker compose restart
docker compose down
```

🏁🔢 Ordre de démarrage

L'ordre de démarrage fait référence à la séquence dans laquelle les services définis dans un fichier Docker Compose sont lancés.

🏁🔢 Ordre de démarrage

Pourquoi est-ce important ?

🏁🔢 Ordre de démarrage

Comment Docker Compose gère-t-il l'ordre de démarrage ?

🏁🔢 Ordre de démarrage

Comment gérer les dépendances entre services ?

🏁🔢 Ordre de démarrage

Exemple Voici un exemple de fichier Docker Compose qui utilise `depends_on` et `healthcheck` pour gérer l'ordre de démarrage des services :

🏁🔢 Ordre de démarrage?🤔 Conditions.

⚠ Rappel: La directive `depends_on` dans un fichier Docker Compose est utilisée pour indiquer qu'un service dépend d'un autre service. Cela signifie que le service dépendant ne sera pas démarré tant que les services dont il dépend n'auront pas été démarrés.

🏁🔢 Ordre de démarrage?🤔 Autres conditions.

En plus de la condition service_healthy, il existe d'autres conditions que vous pouvez utiliser avec `depends_on`:

🏁🔢 Ordre de démarrage?🤔 Autre condition.

Voici un exemple de comment vous pourriez utiliser `service_started` dans un fichier Docker Compose :

🏁🔢 Ordre de démarrage & 🩺✅ healthcheck

⚠️ Rappel `healthcheck` est une instruction dans un `Dockerfile` qui permet de vérifier l'état de santé d'un service. Il peut être utilisé pour déterminer si un service est prêt à être utilisé ou non.

🏁🔢 Ordre de démarrage & 🩺✅ healthcheck

⚠️ Rappel `depends_on` est une directive dans un fichier Docker Compose qui indique qu'un service dépend d'un autre service. Il peut être utilisé pour contrôler l'ordre de démarrage des services.

🏁🔢 🩺✅ Comment healthcheck et depends_on travaillent ensemble ?

En combinant `healthcheck` et `depends_on`, vous pouvez contrôler l'ordre de démarrage des services en fonction de leur état de santé. Par exemple, vous pouvez vous assurer qu'un service de base de données est prêt à être utilisé avant de démarrer une application web qui en dépend.

🏁🔢 Ordre de démarrage 📊📝 Pour résumer

L'ordre de démarrage fait référence à la séquence dans laquelle les services définis dans un fichier Docker Compose sont lancés. Par défaut, Docker Compose démarre les services dans l'ordre dans lequel ils sont définis dans le fichier Docker Compose.

🏁🔢 Ordre de démarrage 📊📝 Pour résumer

- `depends_on` : Cette directive peut être utilisée pour indiquer qu'un service dépend d'un autre service. Elle peut être utilisée avec deux conditions : `service_started` et `service_healthy`.

- `service_started` : Cette condition signifie que le service dépendant ne sera pas démarré tant que le service dont il dépend n'aura pas été démarré. Cependant, cela ne garantit pas que le service dont il dépend est prêt à être utilisé.

- `service_healthy` : Cette condition est utilisée avec la directive healthcheck dans un Dockerfile pour vérifier l'état de santé d'un service. Si la commande healthcheck réussit, Docker considère le service comme sain. Sinon, il est considéré comme malsain.

🏁🔢 Ordre de démarrage 📊📝 Pour résumer

Exemple Voici un exemple de fichier Docker Compose qui utilise depends_on et healthcheck pour gérer l'ordre de démarrage des services :

🏁🔢 Étude de cas 📚🔍

Un exemple de fichier docker-compose.yml utilisé dans Jenkins:

https://raw.githubusercontent.com/jenkins-docs/quickstart-tutorials/refs/heads/main/docker-compose.yaml

```yaml
sidekick_service:
    # Configuration for the sidekick service
    image: ${DOCKERHUB_USERNAME}/jenkinsci-tutorials:sidekick_
    stdin_open: true
    tty: true
    entrypoint: sh -c "/usr/local/bin/keygen.sh /ssh-dir"  # Runs the keygen.sh script and specifies the output directory
    volumes:
      - agent-ssh-dir:/ssh-dir  # Mounts the agent-ssh-dir volume to the /ssh-dir path inside the container
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Checks if the conductor_ok file exists in the /s
      interval: 5s
      timeout: 10s
      retries: 5
```

🏁🔢 Étude de cas 📚🔍

```yaml
sidekick_service:
    # Configuration for the sidekick service
    image: ${DOCKERHUB_USERNAME}/jenkinsci-tutorials:sidekick_
    stdin_open: true
    tty: true
    entrypoint: sh -c "/usr/local/bin/keygen.sh /ssh-dir"  # Runs the keygen.sh script and specifies the output directory
    volumes:
      - agent-ssh-dir:/ssh-dir  # Mounts the agent-ssh-dir volume to the /ssh-dir path inside the container
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Checks if the conductor_ok file exists in the /s
      interval: 5s
      timeout: 10s
      retries: 5
```

- La directive `entrypoint` est utilisée pour spécifier la commande qui sera exécutée lorsque le conteneur démarre.
- Dans ce cas, elle exécute une commande shell qui exécute un script nommé `keygen.sh` situé à `/usr/local/bin/keygen.sh`.
- Le script reçoit un argument `/ssh-dir`, qui est le répertoire où le script effectuera ses opérations.

🏁🔢 Étude de cas 📚🔍

```yaml
sidekick_service:
    # Configuration for the sidekick service
    image: ${DOCKERHUB_USERNAME}/jenkinsci-tutorials:sidekick_
    stdin_open: true
    tty: true
    entrypoint: sh -c "/usr/local/bin/keygen.sh /ssh-dir"  # Runs the keygen.sh script and specifies the output directory
    volumes:
      - agent-ssh-dir:/ssh-dir  # Mounts the agent-ssh-dir volume to the /ssh-dir path inside the container
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Checks if the conductor_ok file exists in the /s
      interval: 5s
      timeout: 10s
      retries: 5
```

🏁🔢 Étude de cas 📚🔍

```yaml
sidekick_service:
    # Configuration for the sidekick service
    image: ${DOCKERHUB_USERNAME}/jenkinsci-tutorials:sidekick_
    stdin_open: true
    tty: true
    entrypoint: sh -c "/usr/local/bin/keygen.sh /ssh-dir"  # Runs the keygen.sh script and specifies the output directory
    volumes:
      - agent-ssh-dir:/ssh-dir  # Mounts the agent-ssh-dir volume to the /ssh-dir path inside the container
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Checks if the conductor_ok file exists in the /s
      interval: 5s
      timeout: 10s
      retries: 5
```

⚠️ Piège : Outils manquants dans les images de base

Le Problème

❗ Si la commande utilisée dans un health check **n'existe pas dans l'image**, le health check **échouera** !

Exemples d'Images et Outils Disponibles

Image de Base	Outils Inclus	Outils ABSENTS
`debian:12`	sh, bash, ps, grep	✖ wget, curl, netcat (installables via apt)
`debian:12-slim`	sh, bash, ps	✖ curl, wget, netcat

🏁🔢 Étude de cas 📚🔍

```yaml
services:
  sidekick_service: [...]

  jenkins_controller: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Check
      interval: 5s
      timeout: 10s
      retries: 5

  default_agent: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
      jenkins_controller:
        condition: service_started
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit
      interval: 5s
      timeout: 10s
      retries: 5
    volumes:
      - agent-ssh-dir:/home/jenkins/.ssh:ro  # Mounts the agent-ssh-dir volume
```

- Service jenkins_controller

🏁🔢 Étude de cas 📚🔍

```yaml
services:
  sidekick_service: [...]

  jenkins_controller: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Check
      interval: 5s
      timeout: 10s
      retries: 5

  default_agent: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
      jenkins_controller:
        condition: service_started
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit
      interval: 5s
      timeout: 10s
      retries: 5
    volumes:
      - agent-ssh-dir:/home/jenkins/.ssh:ro  # Mounts the agent-ssh-dir volume
```

Un `healthcheck` est également défini pour ce service.

🏁🔢 Étude de cas 📚🔍

```yaml
services:
  sidekick_service: [...]

  jenkins_controller: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Check
      interval: 5s
      timeout: 10s
      retries: 5

  default_agent: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
      jenkins_controller:
        condition: service_started
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit
      interval: 5s
      timeout: 10s
      retries: 5
    volumes:
      - agent-ssh-dir:/home/jenkins/.ssh:ro  # Mounts the agent-ssh-dir volume
```

Service `default_agent`

🏁🔢 Étude de cas 📚🔍

```yaml
services:
  sidekick_service: [...]

  jenkins_controller: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Check
      interval: 5s
      timeout: 10s
      retries: 5

  default_agent: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
      jenkins_controller:
        condition: service_started
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit
      interval: 5s
      timeout: 10s
      retries: 5
    volumes:
      - agent-ssh-dir:/home/jenkins/.ssh:ro  # Mounts the agent-ssh-dir volume
```

Un `healthcheck` est également défini pour ce service.

🏁🔢 Étude de cas 📚🔍

```yaml
services:
  sidekick_service: [...]

  jenkins_controller: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /ssh-dir/conductor_ok ] || exit 1" ]  # Check
      interval: 5s
      timeout: 10s
      retries: 5

  default_agent: [...]
    depends_on:
      sidekick_service:
        condition: service_completed_successfully  # Depends on the successful
      jenkins_controller:
        condition: service_started
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit
      interval: 5s
      timeout: 10s
      retries: 5
    volumes:
      - agent-ssh-dir:/home/jenkins/.ssh:ro  # Mounts the agent-ssh-dir volume
```

Enfin, un volume nommé `agent-ssh-dir` est monté sur le chemin `/home/jenkins/.ssh` à l'intérieur du conteneur en lecture seule.

🏁🔢 Étude de cas 📚🔍

À étudier au calme chez vous:

https://raw.githubusercontent.com/gounthar/MyFirstAndroidAppBuiltByJenkins/stf/jenkins/docker-compose.yml

Transformer son application en docker compose

Deux approches différentes et complémentaires :

🎓 Travaux pratiques #12

https://gitlab.univ-artois.fr/bruno.verachten/devops-docker-tp12.git

🎓 Travaux pratiques #12-BIS 🔨🔥

Ça vous dirait d'avoir votre propre forge Gitlab ?

🎓 Travaux pratiques #12-BIS

🎓 Travaux pratiques #12-BIS 🔨🔥

Pas assez détaillé? 😨

🎓 Travaux pratiques #12-BIS 🔨🔥

- Sous `gitlab:`, ajoutez les détails de votre service.
  - Par exemple, `image: 'gitlab/gitlab-ce:latest'` pour spécifier l'image Docker à utiliser pour ce service.
  - Continuez à ajouter des détails pour le service gitlab, comme `restart: always` pour toujours redémarrer le conteneur s'il s'arrête, et `hostname: 'localhost'` pour définir le nom d'hôte du serveur GitLab.
- Définissez l'URL externe de votre serveur GitLab en ajoutant `environment:` et `GITLAB_OMNIBUS_CONFIG: |` sur de nouvelles lignes,
- puis `external_url 'http://localhost'` sur la ligne suivante.
- Exposez les ports nécessaires en ajoutant `ports:` sur une nouvelle ligne,
- puis `- '80:80'`, `- '443:443'` et `- '22:22'` sur les lignes suivantes.

🎓 Travaux pratiques #12-BIS 🔨🔥

🎓 Travaux pratiques #12-BIS 🔨🔥

🎓 Travaux pratiques #12-BIS 🔨🔥

✅ Solution Travaux pratiques #12-BIS 🔨🔥

```yaml
# This is a Docker Compose file for setting up a GitLab server and two GitLab runners.

services:
  # The GitLab server service
  gitlab:
    # The Docker image to use for the GitLab server
    image: 'gitlab/gitlab-ce:16.6.0-ce.0'
    # Always restart the container if it stops
    restart: always
    # The hostname for the GitLab server
    hostname: 'localhost'
    # Environment variables for the GitLab server
    environment:
      # Configuration for the GitLab Omnibus package
      GITLAB_OMNIBUS_CONFIG: |
        # The external URL for the GitLab server
        external_url 'http://localhost'
    # The ports to expose from the GitLab server
    ports:
      - '80:80'   # HTTP
      - '443:443' # HTTPS
      - '22:22'   # SSH
    # The volumes to mount for the GitLab server
    volumes:
      - 'gitlab_config:/etc/gitlab'       # Configuration files
      - 'gitlab_logs:/var/log/gitlab'     # Log files
      - 'gitlab_data:/var/opt/gitlab'     # Data files

  # The first GitLab runner service
  gitlab-runner-1:
```

🔨🔥 J’ai ma forge! Bon, et maintenant? 😐

🔨🔥 J’ai ma forge! Bon, et maintenant? 😐

Il va falloir se logguer, lier les runners au serveur, créer un projet, etc…

🔨🔥 J'ai ma forge! Bon, et maintenant? 😐

Pour trouver le mot de passe, il va falloir le demander gentiment à Gitlab:

```
docker compose -f tp12-bis.yml exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
Password: qaPxXU+RqioolV3bAljvs2VYnyYa4jO/UYcis/UXLAk=
```

🔨🔥 J’ai ma forge! Bon, et maintenant? 😐

Pour utiliser le runner GitLab dans GitLab, vous devez le configurer.

🔨🔥 J’ai ma forge! Bon, et maintenant? 😐

```
docker compose -f tp12-bis.yml exec gitlab-runner-1 gitlab-runner register --url http://gitlab  --token glrt-PF5rLbUKzku5
Runtime platform                                    arch=amd64 os=linux pid=103 revision=853330f9 version=16.5.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
[http://gitlab]:
Verifying runner... is valid                        runner=PF5rLbUKz
Enter a name for the runner. This is stored only in the local config.toml file:
[3c2ee0d97d2b]: runner 1
Enter an executor: parallels, docker-autoscaler, docker+machine, custom, docker, docker-windows, shell, ssh, virtualbox,
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically rel

Configuration (with the authentication token) was saved in "/etc/gitlab-runner/config.toml"
```

Runners

New instanc

All 2 | Instance 2 | Group 0 | Project 0

Search or filter results...

Created

Online 2 | Offline 0 | Stale 0

Status	Runner	Owner
Online, Idle	#2 (DDPNEs6EU) Instance Version 16.5.0 Last contact: 1 minute ago · 172.21.0.4 · 0 · Created 2 minutes ago by	Administrator
Online, Idle	#1 (x5UFWdxNd) 2 Instance Version 16.5.0 Last contact: 1 minute ago · 172.21.0.4 (+1) · 0 · Created 3 minutes ago by	Administrator

👤 Profils dans Docker Compose 🐙

On a vu déjà l'instruction `depends_on` dans un fichier Docker Compose.

👤 Profils dans Docker Compose 🐙

Par exemple :

👤 Profils dans Docker Compose 🐙

👤 Profils dans Docker Compose 🐙

👤 Profils dans Docker Compose 🐙

```yaml
services:
  service_dev:
    image: mon_image_dev
    profiles:
      - dev
  service_prod:
    image: mon_image_prod
    profiles:
      - prod
```

🔨🔥 Une autre forge, ça vous dit? 😐

🔨🔥 Une autre forge, ça vous dit? 😐

Indigeste? 😰

🔨🔥 Une autre forge, ça vous dit? 😐

Maven

```
FROM jenkins/ssh-agent:7.11.0-jdk17 as ssh-agent

# ca-certificates because curl will need it later on for the maven installation
RUN apt-get update && apt-get install -y --no-install-recommends ca-certificates curl && apt-get clean && \
  rm -rf /var/lib/apt/lists/*

# Now time to install maven
ARG MAVEN_VERSION=3.9.2

# Set SHELL flags for RUN commands to allow -e and pipefail
# Rationale:https://github.com/hadolint/hadolint/wiki/DL4006
SHELL ["/bin/bash", "-eo", "pipefail", "-c"]

# Add a checksum for the maven binary
RUN curl -sS -L -O --output-dir /tmp/ --create-dirs  https://archive.apache.org/dist/maven/maven-3/${MAVEN_VERSION}/binar
    && printf "%s" "$(sha512sum /tmp/apache-maven-${MAVEN_VERSION}-bin.tar.gz)" | sha512sum -c - \
    && curl -sS -L -O --output-dir /tmp/ --create-dirs  https://downloads.apache.org/maven/maven-3/${MAVEN_VERSION}/binar
    && printf "%s /tmp/apache-maven-${MAVEN_VERSION}-bin.tar.gz" "$(cat /tmp/apache-maven-${MAVEN_VERSION}-bin.tar.gz.sha
    && tar xzf "/tmp/apache-maven-${MAVEN_VERSION}-bin.tar.gz" -C /opt/ \
    && rm "/tmp/apache-maven-${MAVEN_VERSION}-bin.tar.gz" \
    && ln -s /opt/apache-maven-${MAVEN_VERSION} /opt/maven \
    && ln -s /opt/maven/bin/mvn /usr/bin/mvn \
    && mkdir -p /etc/profile.d \
    && echo "export JAVA_HOME=$JAVA_HOME \n \
             export M2_HOME=/opt/maven \n \
             export PATH=${M2_HOME}/bin:${PATH}" > /etc/profile.d/maven.sh
ENV M2_HOME="/opt/maven"
ENV PATH="${M2_HOME}/bin/:${PATH}"
RUN echo "PATH=${PATH}" >> /etc/environment && chown -R jenkins:jenkins "${JENKINS_AGENT_HOME}"
```

🔨🔥 Une autre forge, ça vous dit? 😐

Python

```yaml
# Le service python est un agent Jenkins avec Python installé.
python:
  build: dockerfiles/python/.  # Le Dockerfile pour construire l'image du service Python.
  container_name: desktop-jenkins_agent-1  # Le nom du conteneur.
  profiles:
    - python  # Les profils à appliquer au service. Cela permet de personnaliser le comportement du service en fonction
  depends_on:  # Les services dont ce service dépend.
    sidekick_service:
      condition: service_completed_successfully  # Le sidekick_service doit se terminer avec succès avant que ce servic
    jenkins_controller:
      condition: service_started  # Le service jenkins_controller doit démarrer avant que ce service ne démarre.
  healthcheck:  # La commande de vérification de santé pour le service.
    test: [ "CMD-SHELL", "[ -f /home/jenkins/.ssh/authorized_keys ] || exit 1" ]  # Vérifie si le fichier authorized_ke
    interval: 5s  # Le temps entre les vérifications de santé.
    timeout: 10s  # Le temps à attendre avant de considérer que la vérification a échoué.
    retries: 5  # Le nombre d'échecs consécutifs nécessaires pour considérer un service comme malsain.
  volumes:
    - agent-ssh-dir:/home/jenkins/.ssh:ro  # Monte le volume agent-ssh-dir au chemin /home/jenkins/.ssh à l'intérieur d
```

🔨🔥 Une autre forge, ça vous dit? 😐

```
# Use the base image
FROM jenkins/ssh-agent:5.12.0-jdk17 as ssh-agent

# Install necessary Dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    binutils ca-certificates curl git python3 python3-pip python3-setuptools python3-wheel python3-dev wget

# Create an alias for python3 as python
RUN ln -s /usr/bin/python3 /usr/bin/python

# Install required python packages
RUN pip install docker-py feedparser nosexcover prometheus_client pycobertura pylint pytest pytest-cov requests setuptool

# Add the Jenkins agent user to the environment
RUN echo "PATH=${PATH}" >> /etc/environment

# Set ownership for Jenkins agent home directory
RUN chown -R jenkins:jenkins "${JENKINS_AGENT_HOME}"
```

🔨🔥 Une autre forge, ça vous dit? 😐

À vous maintenant!

```
git clone https://github.com/jenkins-docs/quickstart-tutorials.git
```

Search (CTRL+K)

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☼	(simple) demo job	13 sec #2	N/A	9.2 sec

Icon: S M L

Icon legend Atom feed for all Atom feed for failures A

Autre étude de cas 📚🔍

Et pourquoi pas s'attaquer au docker compose qui a généré ce cours ?

Autre étude de cas 📚🔍

Et pourquoi pas s'attaquer au docker compose qui a généré ce cours ?

```yaml
# Ceci est un fichier Docker Compose pour un projet qui comprend plusieurs services.

# 'x-slides-base' est une ancre YAML qui définit une configuration commune pour plusieurs services.
x-slides-base: &slides-base
  # La configuration de construction pour l'image Docker.
  build:
    # Le contexte de construction est le répertoire courant.
    context: ./
    args:
      # Active la fonction de mise en cache en ligne de Docker BuildKit.
      BUILDKIT_INLINE_CACHE: 1
  # Variables d'environnement pour le conteneur Docker.
  environment:
    - PRESENTATION_URL=${PRESENTATION_URL}
    - REPOSITORY_URL=${REPOSITORY_URL}
  # L'ID utilisateur qui exécute les commandes à l'intérieur du conteneur.
  user: ${CURRENT_UID}
  # Un montage tmpfs pour des opérations d'E/S plus rapides.
  tmpfs:
    - ${BUILD_DIR}
  # Volumes pour le conteneur Docker.
  volumes:
    - ./content:/app/content
    - ./assets:/app/assets
    - ${DIST_DIR}:/app/dist
    - ./gulp/gulpfile.js:/app/gulpfile.js
    - ./gulp/tasks:/app/tasks
    - ./npm-packages:/app/npm-packages

# Les services qui composent l'application
```

Autre étude de cas 📚🔍

Et pourquoi pas s'attaquer au docker compose qui a généré ce cours ?

➡️🔄 Les ancres dans `docker compose` ⚓

Docker Compose permet de définir et de gérer plusieurs services Docker dans un seul fichier YAML.

➡️🔄 Les ancres dans `docker compose` ⚓

Les ancres peuvent être référencées ailleurs dans le fichier YAML en utilisant l'opérateur `*`.

Autre étude de cas 📚🔍

Le fameux qrcode…

Autre étude de cas 📚🔍

Le service qui construit les slides à partir d'`asciidoc` avec `reveal.js`.

Autre étude de cas 📚🔍

Le service qui construit les slides à partir d'`asciidoc` avec `reveal.js`.

Autre étude de cas 📚🔍

Le service qui construit le pdf à partir des slides.

Autre étude de cas 📚🔍

Pour rappel, l’ancre contenait :

➡️🔄 Build quoi? 🛠️📦

➡️🔄 Environnement 🌍🌿

```
# Variables d'environnement pour le conteneur Docker.
environment:
  - PRESENTATION_URL=${PRESENTATION_URL}
  - REPOSITORY_URL=${REPOSITORY_URL}
```

Deux variables d'environnement sont définies pour le conteneur Docker : `PRESENTATION_URL` et `REPOSITORY_URL`.

➡️🔄 Environnement 🌍🌿

```
# Variables d'environnement pour le conteneur Docker.
environment:
  - PRESENTATION_URL=${PRESENTATION_URL}
  - REPOSITORY_URL=${REPOSITORY_URL}
```

Lorsque Docker Compose rencontre cette syntaxe, il cherche la valeur de la variable d'environnement dans plusieurs endroits, en suivant un ordre spécifique :

➡️🔄 Environnement 🌍🌿

```
# Variables d'environnement pour le conteneur Docker.
environment:
  - PRESENTATION_URL=${PRESENTATION_URL}
  - REPOSITORY_URL=${REPOSITORY_URL}
```

- Si la variable n'est définie ni dans le shell ni dans le fichier `.env`, Docker Compose utilise la valeur par défaut spécifiée dans le fichier `docker-compose.yml` (s'il y en a une).
- Dans notre cas, aucune valeur par défaut n'est spécifiée, donc si la variable n'est définie ni dans le shell ni dans le fichier `.env`, Docker Compose générera une erreur.

Autre étude de cas 📚🔍

```
# L'ID utilisateur qui exécute les commandes à l'intérieur du conteneur.
user: ${CURRENT_UID}
```

Autre étude de cas 📚🔍

```
# Ce Dockerfile met en place un environnement Node.js avec des outils et dépendances supplémentaires.
# Il est basé sur l'image Docker officielle de Node.js (version 22, variante Alpine).

FROM node:25-alpine

# Installer la dernière version des dépendances requises
# hadolint ignore=DL3018
RUN apk add --no-cache \
  curl \ # Outil pour transférer des données avec des URLs
  git \ # Système de contrôle de version distribué
  tini \ # Un init minuscule mais valide pour les conteneurs
  unzip # Outil pour décompresser les fichiers zip

# Installer les dépendances NPM globalement (dernières versions)
# hadolint ignore=DL3016,DL3059
RUN npm install --global npm npm-check-updates # Mettre à jour npm à la dernière version et installer npm-check-updates

# Copier les dépendances NPM de l'application dans l'image Docker
COPY ./npm-packages /app/npm-packages
# Créer des liens symboliques pour package.json et package-lock.json à la racine de /app
# Cela permet d'exécuter les opérations npm sans erreur ENOENT
RUN ln -s /app/npm-packages/package.json /app/package.json \
  && ln -s /app/npm-packages/package-lock.json /app/package-lock.json

# Définir le répertoire de travail dans l'image Docker à /app
WORKDIR /app

# Télécharger et installer une version spécifique de FontAwesome
ARG FONTAWESOME_VERSION=6.4.0
```

Autre étude de cas 📚🔍

```
RUN npm install --global npm npm-check-updates # Mettre à jour npm à la dernière version et installer npm-check-updates

# Copier les dépendances NPM de l'application dans l'image Docker
COPY ./npm-packages /app/npm-packages
# Créer des liens symboliques pour package.json et package-lock.json à la racine de /app
# Cela permet d'exécuter les opérations npm sans erreur ENOENT
RUN ln -s /app/npm-packages/package.json /app/package.json \
  && ln -s /app/npm-packages/package-lock.json /app/package-lock.json
```

Autre étude de cas 📚🔍

```
ARG FONTAWESOME_VERSION=6.4.0
RUN curl --silent --show-error --location --output /tmp/fontawesome.zip \
    "https://use.fontawesome.com/releases/v${FONTAWESOME_VERSION}/fontawesome-free-${FONTAWESOME_VERSION}-web.zip" \
  && unzip -q /tmp/fontawesome.zip -d /tmp \
  && mv /tmp/"fontawesome-free-${FONTAWESOME_VERSION}-web" /app/fontawesome \
  && rm -rf /tmp/font*
```

Autre étude de cas 📚🔍

```
# Installer les dépendances NPM en utilisant le package-lock.json
# Si l'installation échoue, revenir à une installation npm régulière
RUN { npm install-clean && npx update-browserslist-db@latest; } || npm install

# Lier la commande gulp pour qu'elle soit disponible dans le PATH
# hadolint ignore=DL3059
RUN npm link gulp
```

Autre étude de cas 📚🔍

```
# Copier les tâches gulp et la configuration dans l'image Docker
COPY ./gulp/tasks /app/tasks
COPY ./gulp/gulpfile.js /app/gulpfile.js

# Définir un volume pour le répertoire /app
VOLUME ["/app"]

# Exposer le port 8000 pour HTTP
EXPOSE 8000
```

Autre étude de cas 📚🔍

```
# Utiliser tini comme point d'entrée, et exécuter gulp par défaut
ENTRYPOINT ["/sbin/tini","-g","gulp"]
CMD ["default"]
```

🐳 Tini dans Docker

🤔 Qu’est-ce que Tini ?

Tini est un `init` minuscule mais valide pour les conteneurs. Il est conçu pour être le système init le plus simple possible.

🛠️ Que fait Tini ?

Tini fait deux choses :

1. Il génère votre processus en tant que son enfant (directement, pas en tant que petit-enfant, comme le ferait un shell).
2. Il attend ensuite les signaux et les transmet au processus enfant.

🎯 Pourquoi Tini est-il utile dans Docker ?

1. Docker exécute un seul processus dans un conteneur par défaut. Si ce processus génère des processus enfants et ne les récolte pas correctement, ils deviennent des processus zombies.
2. Tini assure que ces processus zombies sont correctement récoltés, améliorant ainsi le comportement du conteneur et réduisant la probabilité de cas limites.

🐳 Tini dans notre Dockerfile

Dans notre Dockerfile, nous utilisons Tini comme point d'entrée :

```
ENTRYPOINT ["/sbin/tini","-g","gulp"]
```

1. Cela signifie que Tini est le premier processus qui est lancé dans notre conteneur.
2. Il lancera ensuite `gulp` en tant que processus enfant.
3. Tout signal envoyé au conteneur sera transmis par Tini à `gulp`.
4. Si `gulp` génère des processus enfants et ne les récolte pas, Tini le fera.

🐳 Docker peut travailler main dans la main avec d'autres outils… 👫🤝

```
# Ce Makefile est utilisé pour gérer le projet Docker Compose.

# Définir les valeurs par défaut pour les variables DIST_DIR et REPOSITORY_URL.
DIST_DIR ?= $(CURDIR)/dist
REPOSITORY_URL ?= file://$(CURDIR)
export REPOSITORY_URL DIST_DIR

# Activer Docker BuildKit pour une construction plus rapide et la mise en cache des images.
DOCKER_BUILDKIT ?= 1
COMPOSE_DOCKER_CLI_BUILD ?= 1
export DOCKER_BUILDKIT COMPOSE_DOCKER_CLI_BUILD

# Définition des commandes shell réutilisables pour Docker Compose.

# compose_cmd est une fonction qui exécute la commande 'docker compose' avec le fichier docker-compose.yml du répertoire
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose'.
# $(CURDIR) est une variable d'environnement dans le Makefile qui représente le répertoire courant dans lequel
# le Makefile est exécuté. C'est une fonctionnalité intégrée de GNU Make. Elle est souvent utilisée pour référencer
# des fichiers ou des répertoires relatifs au répertoire courant.
compose_cmd = docker compose --file=$(CURDIR)/docker-compose.yml $(1)

# compose_up est une fonction qui utilise compose_cmd pour exécuter 'docker compose up'.
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose up'.
# L'option '--build' est toujours incluse, ce qui signifie que Docker construira les images avant de démarrer les contene
compose_up = $(call compose_cmd, up --build $(1))

# compose_run est une fonction qui utilise compose_cmd pour exécuter 'docker compose run'.
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose run'.
# L'option '--user=0' est toujours incluse, ce qui signifie que les commandes seront exécutées en tant que root à l'intér
```

🐳 Docker peut travailler main dans la main avec d'autres outils… 👫🤝

```
# Activer Docker BuildKit pour une construction plus rapide et la mise en cache des images.
DOCKER_BUILDKIT ?= 1
COMPOSE_DOCKER_CLI_BUILD ?= 1
export DOCKER_BUILDKIT COMPOSE_DOCKER_CLI_BUILD
```

Cette configuration de docker compose concerne Docker BuildKit, une fonctionnalité de Docker qui améliore les performances de construction des images Docker.

- La ligne `DOCKER_BUILDKIT ?= 1` vérifie si la variable d'environnement `DOCKER_BUILDKIT` est déjà définie.
  - Si ce n'est pas le cas, elle lui attribue la valeur `1`, ce qui active Docker BuildKit.
- De même, `COMPOSE_DOCKER_CLI_BUILD ?= 1` vérifie si la variable d'environnement `COMPOSE_DOCKER_CLI_BUILD` est déjà définie.
  - Si ce n'est pas le cas, elle lui attribue la valeur `1`.
  - Cette variable d'environnement est utilisée pour activer l'utilisation de Docker CLI lors de l'utilisation de Docker Compose.
  - C'est nécessaire car Docker Compose ne supporte pas BuildKit par défaut, donc cette variable

🐳 Docker peut travailler main dans la main avec d'autres outils… 👫🤝

```
# Définition des commandes shell réutilisables pour Docker Compose.

# compose_cmd est une fonction qui exécute la commande 'docker compose' avec le fichier docker-compose.yml du répertoire
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose'.
# $(CURDIR) est une variable d'environnement dans le Makefile qui représente le répertoire courant dans lequel
# le Makefile est exécuté. C'est une fonctionnalité intégrée de GNU Make. Elle est souvent utilisée pour référencer
# des fichiers ou des répertoires relatifs au répertoire courant.
compose_cmd = docker compose --file=$(CURDIR)/docker-compose.yml $(1)

# compose_up est une fonction qui utilise compose_cmd pour exécuter 'docker compose up'.
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose up'.
# L'option '--build' est toujours incluse, ce qui signifie que Docker construira les images avant de démarrer les contene
compose_up = $(call compose_cmd, up --build $(1))

# compose_run est une fonction qui utilise compose_cmd pour exécuter 'docker compose run'.
# Elle prend un argument $(1) qui représente les options supplémentaires à passer à la commande 'docker compose run'.
# L'option '--user=0' est toujours incluse, ce qui signifie que les commandes seront exécutées en tant que root à l'intér
compose_run = $(call compose_cmd, run --user=0 $(1))
```

🐳 Docker peut travailler main dans la main avec d'autres outils… 👫🤝

```
# Construire le projet à l'intérieur d'un conteneur Docker.
# Cette règle Makefile utilise la fonction compose_up définie précédemment pour exécuter 'docker compose up' avec l'optio
# L'option '--exit-code-from=build' signifie que la commande 'docker compose up' renverra le code de sortie du service 'b
# Si le service 'build' se termine avec un code de sortie non nul (ce qui signifie qu'une erreur s'est produite), alors '
# Cela permet à Make de savoir si la construction du projet a réussi ou non.
build:
 @$(call compose_up,--exit-code-from=build build)
```

📖🔚 Fin du chapitre docker compose 🐳🐙

Sécurité et Production Docker

Objectifs de la session

Projet Fil Rouge : SecureVote

SecureVote : Une application de vote en ligne moderne

Dépôt GitHub : https://github.com/gounthar/securevote

Architecture de l'application

```
services:
  frontend:     # React app
  backend:      # Flask API
  database:     # PostgreSQL
  cache:        # Redis
  proxy:        # Nginx
```

Architecture multi-tiers classique pour DevOps

Votre mission

Phase 1 : Découverte (30 min)

```
git clone https://github.com/gounthar/securevote.git
cd securevote/phase1
docker compose up -d

# Vérifier l'état des services
docker compose ps

# Suivre les logs (Ctrl+C pour quitter)
docker compose logs -f
```

Analyse des vulnérabilités

Explorez les fichiers et identifiez les problèmes :

Points d’attention

À vérifier :

Sécurité Docker : Les fondamentaux

Les 3 piliers de la sécurité Docker :

Principe du moindre privilège

Un conteneur ne devrait avoir QUE les permissions nécessaires à son fonctionnement

Scan de vulnérabilités

Les images Docker peuvent contenir des vulnérabilités connues (CVE)

Outils de scan :

Docker Scout en action

```
# Scanner une image locale
docker scout cves python:3.11

# Comparer deux images
docker scout compare python:3.11 --to python:3.11-slim
```

Démonstration live recommandée

Installation de Trivy

Méthode rapide (pour le TP) :

```
# Installation directe du binaire
curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sudo sh -s -- -b /usr/local/bin
trivy --version
```

⚠ `curl | sudo sh` est pratique MAIS dangereux en production !

Méthode sécurisée (recommandée) :

```
# 1. Télécharger le script
curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh -o /tmp/trivy-install.sh

# 2. Inspecter le contenu
less /tmp/trivy-install.sh

# 3. Exécuter après vérification
sudo sh /tmp/trivy-install.sh -b /usr/local/bin
```

Installation Trivy : Alternatives

Si vous n'avez pas les droits sudo :

```
# Via Docker (aucune installation nécessaire)
docker run --rm \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v $HOME/.cache/trivy:/root/.cache \
  aquasec/trivy:0.54.1 image python:3.11
```

Pin la version (`:0.54.1`) et persist le cache (`-v $HOME/.cache`) pour performances

Autres systèmes :

Exemple de résultat Trivy

```
trivy image python:3.11

Total: 145 vulnerabilities (52 HIGH, 93 MEDIUM)
┌──────────┬────────────────┬──────────┬──────────┐
│ Library  │ Vulnerability  │ Severity │ Version  │
│ openssl  │ CVE-2023-XXX   │ HIGH     │ 1.1.1n   │
│ curl     │ CVE-2023-YYY   │ MEDIUM   │ 7.68.0   │
└──────────┴────────────────┴──────────┴──────────┘
```

Utilisateurs non-root

Problème : Par défaut, les processus s'exécutent en tant que root

```
FROM python:3.11
COPY app.py /app/
CMD ["python", "/app/app.py"]  # Root !
```

⚠ Si compromis, l'attaquant a les privilèges root !

Solution : Utilisateur dédié

```
FROM python:3.11-slim

RUN groupadd -r appuser && useradd -r -g appuser appuser
WORKDIR /app
COPY app.py /app/
RUN chown -R appuser:appuser /app

USER appuser
CMD ["python", "/app/app.py"]
```

Bonnes pratiques utilisateur

Comprendre les UID Linux

Linux numérote les utilisateurs (User ID) :

UID : Exemple pratique

```
# RISQUÉ : UID fixe < 1000
RUN useradd -u 33 appuser
# Conflit possible si l'hôte a www-data (UID 33) !

# SÛR : UID > 1000
RUN useradd -u 1001 appuser

# OU laisser le système choisir (sera ≥ 1000)
RUN useradd appuser
```

Gestion des secrets

Mauvaise pratique :

```
environment:
  - POSTGRES_PASSWORD=super_secret_123  # NON !
```

⚠ Visible dans `docker inspect`, logs, historique !

Bonne pratique : Fichiers .env

Étape 1 : Créer le fichier .env

```
# .env (à ne JAMAIS commiter)
DB_PASSWORD=super_secret_123
```

Étape 2 : Protéger avec .gitignore

```
# .gitignore
.env
secrets/
```

Étape 3 : Utiliser dans docker-compose.yml

```
environment:
  POSTGRES_PASSWORD: ${DB_PASSWORD}
  API_KEY: ${API_KEY:-default_key}     # Valeur par défaut si absent
  DB_HOST: ${DB_HOST?required}          # Erreur si non défini
```

Docker Compose charge automatiquement le fichier .env

Syntaxes avancées :

Alternative 1 : Docker Secrets

Gestion native dans Docker Swarm

```
echo "secret123" | docker secret create db_password -
docker service create --secret db_password postgres
```

Alternative 2 : HashiCorp Vault

Solution d’entreprise pour secrets

```
# Application interroge Vault via API
vault kv get -field=password secret/database/prod
```

Alternative 3 : Cloud Providers

Services managés AWS / Azure / GCP

Alternative 4 : Variables shell

Passage direct via environnement

```
# En une ligne
DB_PASSWORD="secret123" docker compose up -d

# Ou via export
export DB_PASSWORD="secret123"
docker compose up -d
```

Comparaison des solutions

Solution	Complexité	Coût	Cas d'usage
Fichiers .env	○ Faible	💰 Gratuit	Dev, petits projets
Docker Secrets	🟡 Moyenne	💰 Gratuit	Docker Swarm prod
Vault	🔴 Élevée	💰💰 Variable	Grandes organisations
Cloud Secrets	🟡 Moyenne	💰💰 Payant	Déjà sur cloud
Variables shell	○ Faible	💰 Gratuit	Dev local, CI/CD

Phase 2 : Sécurisation (1h15)

Transformer SecureVote en application sécurisée

Exercice 1 : Images sécurisées (20 min)

Mission :

```
docker scout cves securevote-backend:latest
# Modifier Dockerfile → python:3.11-slim
docker compose build backend
docker scout cves securevote-backend:latest
```

Exercice 2 : Utilisateurs non-root (25 min)

Mission :

Modifier tous les Dockerfiles pour des utilisateurs non-root

```
FROM python:3.11-slim
RUN groupadd -r flaskuser && useradd -r -g flaskuser flaskuser
USER flaskuser
```

Vérification utilisateurs

```
docker compose exec backend ps aux
# Le processus principal (PID 1) doit tourner en tant que flaskuser

docker compose exec frontend ps aux
# Le processus principal (PID 1) doit tourner en tant que reactuser
```

Si vous voyez UID 0 ou `root` pour le processus principal, c'est à corriger !

Exercice 3 : Sécuriser les secrets (30 min)

Mission :

Checkpoint Phase 2

Vérification collective :

```
docker compose ps
docker compose exec backend ps aux
trivy image securevote-backend:latest --severity HIGH,CRITICAL
```

Configuration Production

Développement :

- Ressources illimitées
- Logs verbeux
- Redémarrages manuels

Production :

- Ressources limitées
- Logs structurés
- Auto-healing

Limites de ressources

Pourquoi limiter ?

⚠ Sans limites → 100% CPU/RAM !

Définir les limites

```yaml
services:
  backend:
    deploy:
      resources:
        limits:
          cpus: '0.5'       # Max 50% CPU
          memory: 512M      # Max 512 Mo RAM
        reservations:
          cpus: '0.25'      # Min garanti
          memory: 256M
```

Choisir les bonnes valeurs

Politiques de redémarrage

```yaml
services:
  backend:
    restart: no               # Jamais
    restart: always           # Toujours
    restart: on-failure       # Si erreur
    restart: unless-stopped   # Sauf arrêt manuel
```

Restart avec limite

```
services:
  backend:
    restart: on-failure
```

Pour limiter le nombre de tentatives en Docker Swarm, utilisez `deploy.restart_policy.max_attempts: 5`

Évite les boucles infinies de redémarrage

Health checks

```yaml
services:
  backend:
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

Endpoint de santé

```python
@app.route('/health')
def health():
    try:
        db.ping()
        return jsonify({"status": "healthy"}), 200
    except:
        return jsonify({"status": "unhealthy"}), 503
```

Dépendances entre services

```yaml
services:
  backend:
    depends_on:
      database:
        condition: service_healthy
      cache:
        condition: service_started
```

Monitoring et logs

Observabilité :

TP13 a déjà couvert ELK

Driver de logs

```yaml
services:
  backend:
    logging:
      driver: "json-file"
      options:
        max-size: "10m"       # Max 10 Mo
        max-file: "3"         # Garder 3 fichiers
```

⚠ Sans limites, les logs remplissent le disque !

Logs structurés

Mauvais : `Server started on port 5000`

Bon :

```
{"timestamp":"2025-01-15T10:30:00Z","level":"INFO","service":"backend","message":"Server started","port":5000}
```

Phase 3 : Production (1h)

Optimiser SecureVote pour la production

Exercice 4 : Limites de ressources (20 min)

Mission :

```
docker stats --no-stream
./load_test.sh
docker stats
```

Exercice 5 : Restart et Health (25 min)

Mission :

```
docker compose kill backend
docker compose ps
docker compose logs backend
```

Exercice 6 : Dépendances (15 min)

Mission :

Checkpoint Phase 3

Application production-ready :

```
docker compose config --services
docker compose ps
docker stats --no-stream
```

Registres et Distribution

Docker Hub

```
docker pull nginx:latest
# Équivalent à :
docker pull docker.io/library/nginx:latest
```

Registres privés

Pourquoi ?

Solutions de registres

Démarrer un registre local

```
docker run -d -p 5000:5000 --name registry registry:2
docker tag securevote-backend:latest localhost:5000/securevote-backend:latest
docker push localhost:5000/securevote-backend:latest
```

Tags et versions

```
# Bonne pratique : versionner
docker tag myapp:latest myregistry.com/myapp:1.2.3
docker tag myapp:latest myregistry.com/myapp:1.2
docker tag myapp:latest myregistry.com/myapp:latest
```

Plusieurs tags → même image

Exercice 7 : Registre local (Bonus)

Si le temps le permet :

Récapitulatif et bonnes pratiques

Checklist sécurité

Checklist production

Ce qu’on a appris

SecureVote : Avant/Après

Avant :

- Images non scannées
- Root partout
- Secrets en clair
- Pas de limites

Après :

- Images slim, scannées
- Utilisateurs dédiés
- Secrets protégés
- Production-ready ✅

🔧 Troubleshooting Production

Conteneurs qui ne démarrent pas

Étape 1 : Vérifier les logs

```
docker logs nom-conteneur

# OU pour suivre en temps réel
docker logs -f nom-conteneur
```

Étape 2 : Inspecter le conteneur

```
docker inspect nom-conteneur

# Voir seulement l'état
docker inspect --format='{{.State.Status}}' nom-conteneur

# Voir le code de sortie
docker inspect --format='{{.State.ExitCode}}' nom-conteneur
```

Problèmes courants :

Symptôme	Cause probable	Solution
`host not found in upstream`	NGINX : résolution DNS statique	Utiliser variables : `set $backend "app:8080"` +

Aller plus loin

Ressources utiles

- Docker Security: https://docs.docker.com/engine/security/
- CIS Docker Benchmark
- Trivy: https://github.com/aquasecurity/trivy
- OWASP Docker Security Cheat Sheet

Fin de la session

Bravo, vous avez transformé SecureVote en application sécurisée et production-ready !

Questions ?

questions docker

gounthar@gmail.com

Bonus

Les petits plus

docker

docker

"Il en remet une couche"

```
docker history nginx
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
f35646e83998        4 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:1d0a4127e78a26c1…   1.96kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>           4 weeks ago         /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.3     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:0dc53e7886c35bc21…   69.2MB
```

"Il en remet une couche"

```
docker history nginx
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
f35646e83998        4 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:1d0a4127e78a26c1…   1.96kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>           4 weeks ago         /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.3     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:0dc53e7886c35bc21…   69.2MB
```

Il est préférable de limiter le nombre de couches pour limiter la bande passante.

"Il en remet une couche"

```
RUN apk add \
     patch \
     tar \
     mercurial \
     git \
     ruby \
     ruby-devel \
     rubygem-bundler \
     make \
     gcc-c++ \
     zlib-devel \
     libxml2-devel \
     docker \
     nodejs \
     npm \
     sssd \
&& mkdir -p /var/lib/docker-builder \
&& mkdir -p /etc/docker-builder
```

Ne pas charger inutilement !

```
&& apk remove make gcc maven ... \
&& rm -f previously-downloaded.tar.gz \
&& apk clean all \
&& rm -rf /tmp/*
```

Documentez !

Les mots clés `EXPOSE` et `LABEL` peuvent fournir de précieuses informations aux utilisateurs de vos images !

(les ports exposés par défaut, les auteurs de l'image, la version d'un middleware embarqué, …)

Préparez-vous au read-only

Il est possible de lister toutes les modifications qui ont été apportées au filesystem d'un container.

```
docker diff 29f1c4
C /run
A /run/nginx.pid
C /var
C /var/lib
C /var/lib/nginx
C /var/lib/nginx/tmp
A /var/lib/nginx/tmp/client_body
A /var/lib/nginx/tmp/fastcgi
A /var/lib/nginx/tmp/proxy
A /var/lib/nginx/tmp/scgi
A /var/lib/nginx/tmp/uwsgi
C /var/log
C /var/log/nginx
A /var/log/nginx/access.log
A /var/log/nginx/error.log
```

HealthCheck

Savoir que mon PID 1 est toujours en cours d'exécution n'est peut-être pas la meilleure piste pour savoir si mon conteneur est en bonne santé !

```
FROM ghost:3
RUN apt update && apt install curl -y \
        && rm -rf /var/lib/apt/lists/*

HEALTHCHECK --interval=1m --timeout=30s --retries=3 CMD curl --fail http://localhost:2368 || exit 1
```

source : https://www.grottedubarbu.fr/docker-healthcheck/

Debugging

Une tonne d'outils !

Les logs

- des containers
- du démon Docker

```
docker container exec
docker inspect
docker cp
docker history
docker stats
```

Debugging : Les logs

rappel : `docker container logs` permet de lister les logs des conteneurs.

Concentrateur de logs

Par défaut, les logs des conteneurs sont stockés dans des fichiers json. Mais comment faire pour les envoyer vers un concentrateur ?

Travaux pratiques #13

Étapes

Lancer une stack ELK grâce aux fichiers fournis dans le repo https://gitlab.univ-artois.fr/bruno.verachten/devops-docker-tp13

Alimenter en logs avec la commande `docker container run --log-driver=gelf --log-opt gelf-address=udp://localhost:12201 alpine bash -c 'seq 1 100'`

Créer un index "timestamp" sur Kibana puis aller à l'écran "discover"

Lancer un conteneur nginx et concentrez ses logs dans ELK

GitLab et Dependabot

🤔 Qu’est-ce que Dependabot ?

Dependabot est un outil qui vous aide à maintenir vos dépendances à jour.

🛠️ Comment Dependabot fonctionne-t-il avec Docker ?

Dependabot peut analyser vos `Dockerfiles` et vos fichiers `docker-compose` pour trouver les dépendances qui peuvent être mises à jour.

🎯 Pourquoi utiliser Dependabot avec Docker ?

1. Garder vos dépendances à jour est crucial pour la sécurité et la stabilité de vos applications.
2. Dependabot automatise ce processus, vous faisant gagner du temps et réduisant le risque d'oublier une mise à jour importante.
3. Dependabot n'est pas magique, il faut que votre CI soit capable de construire votre application avec les nouvelles dépendances.
4. Ce qui n'est pas testé ne fonctionne pas.

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

- Malheureusement, il n'y a pas encore de support officiel pour GitLab dans Dependabot (et vice versa).
- Notre forge Gitlab est définie dans un fichier docker-compose.yml, et Dependabot aussi.
- Dependabot doit avoir accès à notre forge GitLab pour pouvoir ouvrir des merge requests.
- Suivons donc la documentation officielle.

Search page

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate again
They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Active personal access tokens 0

Token name	Scopes	Created	Last Used	Expires	Actio
This user has no active personal access tokens.					

Feed token

Your feed token authenticates you when your RSS reader loads a personalized RSS feed or when your calendar application loads a personalized calendar. It is visible in thos
cannot be used to access any other data.

Feed token

••••••••••••••••••••••••

Keep this token secret. Anyone who has it can read activity and issue RSS feeds or your calendar feed as if they were you. If that happens, reset this token.

Personal Access Tokens

Add a personal access token

Token name

dependabot-token

For example, the application using the token or the purpose of the token.

Expiration date

2023-12-21

Select scopes

Scopes set the permission levels granted to the token. Learn more.

- [x] api
  Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- [x] read_api
  Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- [ ] read_user
  Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access
  API endpoints under /users.
- [ ] create_runner
  Grants create access to the runners.
- [ ] k8s_proxy
  Grants permission to perform Kubernetes API calls using the agent for Kubernetes.
- [x] read_repository
  Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- [x] write_repository
  Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
- [x] sudo
  Grants permission to perform API actions as any user in the system, when authenticated as an admin user.
- [x] admin_mode
  Grants permission to perform API actions as an administrator, when Admin Mode is enabled.

Create personal access token | Cancel

Token name	Scopes	Created	Last Used	Expires	Actio

This user has no active personal access tokens.

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

> ⚠️ Notez bien le token, vous ne le reverrez plus jamais.

L'étape suivante, c'est de positionner les valeurs de certaines variables d'environnement. Ça peut être dans un `.env`, ou dans les variables d'environnement de votre machine.

```
export SETTINGS__GITLAB_URL=http://localhost
export SETTINGS__GITLAB_ACCESS_TOKEN=glpat-LXUfrxeFXuRJXNTNNifD
```

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

Démarrez l'application avec `docker compose`:

```
curl -s https://gitlab.com/dependabot-gitlab/dependabot/-/raw/v3.8.0-alpha.1/docker-compose.yml | docker compose -f - up
```

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

Vous vous souvenez du chapitre sur les réseaux Docker ?

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

- Lorsque vous utilisez Docker Compose, il crée automatiquement un réseau par défaut pour votre application et tout service défini dans le `docker-compose.yml` peut atteindre les autres en utilisant le nom du service comme nom d'hôte.

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

- Voici un exemple :

```
version: '3'
services:
  web:
    image: web
    networks:
      - mynetwork
  gitlab-instance-gitlab-1:
    image: gitlab
    networks:
      - mynetwork
networks:
  mynetwork:
```

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

```
  web:
    image: *base_image
    networks:
[...]
networks:
  mynetwork:
```

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

Sauf que…

🚀 Comment configurer Dependabot pour GitLab ? 🦊🛠️

Victoire?

Debugging : docker exec

rappel : `docker container exec `permet de lancer une commande dans un container

```
docker exec <containerID> echo "hello"
```

```
docker exec -it <containerID> bash
```

Debugging : docker inspect

rappel : `docker inspect `permet de lister toutes les caractéristiques d'une image ou d'un container.

On peut filtrer le retour de la commande avec `jq` ou l'option `--format`.

Debugging : docker cp

Cette commande permet d'échanger des fichiers entre un conteneur et la machine hôte.

```
docker cp --help
Usage:  docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
        docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
Copy files/folders between a container and the local filesystem
Options:
  -a, --archive       Archive mode (copy all uid/gid information)
  -L, --follow-link   Always follow symbol link in SRC_PATH
```

Debugging : docker history

Cette commande permet d'afficher la concaténation de tous les Dockerfiles qui ont abouti à cette image.

```
docker history nginx
IMAGE               CREATED             CREATED BY                                      SIZE                f35646e83998
<missing>           4 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:1d0a4127e78a26c1…   1.96kB
<missing>           4 weeks ago         /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>           4 weeks ago         /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.3     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:0dc53e7886c35bc21…   69.2MB
```

Debugging : docker stats

Cette commande permet d'avoir les stats en temps réel d'un conteneur.

```
docker stats 42f128
CONTAINER           CPU %     MEM USAGE/LIMIT     MEM %     NET I/O
42f128              0.00%     1.454 MB/4.145 GB   0.04%     648 B/648 B
```

🎓 Travaux pratiques #14

Notre valeureux collègue Jean-Michel s'initiait à l'art mystérieux de Docker, et généreusement partageait ses créations sous forme d'images Docker pour l'entreprise.

Michel a soudainement décroché le jackpot du Loto, laissant derrière lui son bureau du jour au lendemain, alors qu'il était sur le point de nous offrir une image Nginx.

Désormais, nous n'avons que le binaire de son chef-d'œuvre, accessible à cette adresse :

https://owncloud.univ-artois.fr/index.php/s/9s8XBLXvRwBJsrm

Rassurez-vous, on nous a dit que `docker load` est la première étape de la formule magique pour percer les secrets de son œuvre.

À vos claviers ! 🧙‍♂️✨

✅ Solution travaux pratiques #14🕵️🔍

`docker load` est la première étape de la formule magique

✅ Solution travaux pratiques #14🕵️🔍

```
docker container ls --format '{{.Names}}' | grep "bad"
```

✅ Solution travaux pratiques #14🕵️🔍

Je ne sais pas ce qu’a fait Jean-Michel, mais il a cassé le binaire de Nginx.

✅ Solution travaux pratiques #14🕵️🔍

```
[...]
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
],
```

✅ Solution travaux pratiques #14🕵️🔍

```
[...]
"Entrypoint": [
    "/bin/sh",
    "-c",
    "/tmp/entrypoint.sh"
],
```

✅ Solution travaux pratiques #14🕵️🔍

```
        [...]
        "Labels": {
            [...]
            "org.label-schema.name": "CentOS Base Image",
            [...]
        }
    },
    "DockerVersion": "18.09.0",
    [...]
    "Architecture": "amd64",
    "Os": "linux",
    [...]
    }
]
```

✅ Solution travaux pratiques #14🕵️🔍

Tournons-nous
vers l'histoire !

✅ Solution travaux pratiques #14🕵️🔍

```
docker history bad-nginx:1
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
a469e4d1cb0c   3 years ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "ngin…   0B
<missing>      3 years ago   /bin/sh -c #(nop)  ENTRYPOINT ["/bin/sh" "-c…   0B
<missing>      3 years ago   /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>      3 years ago   /bin/sh -c #(nop) COPY file:b0fc89bcaf962602…   98B
<missing>      3 years ago   /bin/sh -c yum install -y nginx && yum clean…   56.7MB
<missing>      3 years ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      3 years ago   /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>      3 years ago   /bin/sh -c #(nop) ADD file:538afc0c5c964ce0d…   215MB
```

✅ Solution travaux pratiques #14🕵️🔍

Résumons-nous… On a:

✅ Solution travaux pratiques #14🕵️🔍💀

```dockerfile
# Utiliser Alpine ou Debian comme image de base
FROM alpine

# Mettre à jour le système et installer nginx
RUN apk update && apk add nginx

# Exposer le port 80
EXPOSE 80

# Ajouter un fichier HTML (remplacer /chemin/vers/votre/fichier.html par le chemin réel vers votre fichier HTML)
ADD /chemin/vers/votre/fichier.html /usr/share/nginx/html

# Définir le point d'entrée pour lancer nginx
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

✅ Solution travaux pratiques #14🕵️🔍

Essayons de lancer un conteneur basé sur notre image, et de récupérer le fichier HTML.

✅ Solution travaux pratiques #14🕵️🔍

Le conteneur qui tourne, on oublie, donc… Jean-Michel a laissé un terrain miné derrière lui. 💣

✅ Solution travaux pratiques #14🕵️🔍

```
head html/index.html
```

✅ Solution travaux pratiques #14🕵️🔍

```
# Ce Dockerfile met en place un serveur web simple en utilisant Nginx sur Alpine Linux.

# Utilise Alpine Linux comme image de base. Alpine Linux est une distribution Linux légère et orientée sécurité.
FROM alpine

# Met à jour la liste des paquets du système et installe Nginx.
# Nginx est un serveur web populaire qui peut également être utilisé comme proxy inverse, équilibreur de charge et cache
RUN apk update && apk add nginx && rm -rf /var/cache/apk/*

# Expose le port 80 au monde extérieur. C'est le port standard pour le trafic HTTP.
EXPOSE 80

# Ajoute un fichier HTML à la racine du document Nginx.
# Remplacez /chemin/vers/votre/fichier.html par le chemin réel vers votre fichier HTML.
ADD /chemin/vers/votre/fichier.html /usr/share/nginx/html

# Définit le point d'entrée pour le conteneur. Cette commande sera exécutée lorsque le conteneur démarre.
# La commande "nginx" démarre le serveur Nginx.
# Le flag "-g" nous permet de définir des directives globales. Dans ce cas, nous disons à Nginx de fonctionner au premier
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

🐳😴 Lazy Docker

https://blog.stephane-robert.info/docs/conteneurs/moteurs-conteneurs/lazydocker/

Conseils pour l’éval'

- https://moodle.univ-artois.fr/mod/resource/view.php?id=39414
- https://moodle.univ-artois.fr/mod/choicegroup/view.php?id=39415
- https://moodle.univ-artois.fr/mod/assign/view.php?id=39416

Conseils pour l’eval'

Vous devez avoir les compétences suivantes:

Savoir lancer un container avec toutes les options

- `-v, -p, -w, -e, --rm, etc.`

Écrire un Dockerfile optimisé pour "dockeriser" une application

- pas trop gourmand, facile à modifier, paramétrable.

Étudier une image ou un conteneur pour tout débug éventuel.

Savoir s'outiller pour avoir des images qui servent d'environnement d'exécution à votre CI.

Monter un écosystème applicatif complexe via docker-compose.

Que revoir ?

Les TPs sont vos meilleurs amis.

La documentation officielle de Docker

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

https://docs.docker.com/engine/reference/builder/

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

🤔👉 One more thing

Bibliographie

Ligne de commande

- https://tldp.org
- https://en.wikipedia.org/wiki/POSIX
- https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop
- https://linuxhandbook.com/linux-directory-structure/
- https://blog.stephane-robert.info/docs/admin-serveurs/linux/script-shell/
- https://linuxopsys.com/topics/bash-script-examples

Git / VCS

- https://docs.github.com
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- http://martinfowler.com/bliki/VersionControlTools.html
- http://martinfowler.com/bliki/FeatureBranch.html
- https://about.gitlab.com/2014/09/29/gitlab-flow/
- https://www.atlassian.com/git/tutorials/comparing-workflows
- http://nvie.com/posts/a-successful-git-branching-model/
- https://blog.stephane-robert.info/docs/developper/version/git/introduction/

Intégration Continue

- http://martinfowler.com/articles/continuousIntegration.html
- http://martinfowler.com/bliki/ContinuousDelivery.html
- https://jaxenter.com/implementing-continuous-delivery-117916.html
- https://technologyconversations.com/2014/04/29/continuous-delivery-introduction-to-concepts-and-tools/
- http://blog.arungupta.me/continuous-integration-delivery-deployment-maturity-model
- http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

Docker

- https://labs.play-with-docker.com/
- https://dduportal.github.io/cours/cnam-docker-2018
- https://kodekloud.com/blog/docker-for-beginners/
- https://www.slideshare.net/dotCloud/why-docker
- https://docs.docker.com/engine/reference/builder/
- https://www.r-bloggers.com/2021/05/best-practices-for-r-with-docker/
- https://github.com/wagoodman/dive
- https://docs.docker.com/engine/tutorials/networkingcontainers/
- https://towardsdatascience.com/docker-networking-919461b7f498
- https://blog.stephane-robert.info/post/docker-buildkit-partie-1/
- https://dev.to/montells/docker-args-1ael

DevOps

- https://blog.stephane-robert.info/docs/glossaire/introduction/

Merci !

gounthar@gmail.com

Slides: https://gounthar.github.io/cours-devops-docker/main

Source on : https://github.com/gounthar/cours-devops-docker