Docker

Patrick Hainaut

Introduction

- A l'heure où tout doit aller très vite, perdre du temps à mettre en place une infrastructure de A à Z pour utiliser une application, c'est révolu
- On va plutôt utiliser des packages OS/application prêts à l'emploi qu'il suffit de déployer
- C'est l'objet de cette présentation non-exhaustive centrée sur Docker
- Je prends le parti pris d'utiliser des termes franglais parce que communément employés par la communauté et que, à mon sens, la traduction à outrance dessert l'efficacité du discours

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

But

- Docker est un système de gestion de conteneurs
- Comme pour les conteneurs de transport, le but est de séparer l'infrastructure de l'applicatif
- Si vous transportez des objets directement, ces objets subissent de la manutention et peuvent s'en suivre dégâts et pertes
- Si vous transportez un conteneur scellé (avec les objets dedans), vous devez juste vous assurez du transport correct du conteneur

©Hainaut P. 2021 - www.coursonline.be

But

- Il en va de même avec les applications
- Si elles sont utilisées telles quelles, il faut les adapter au système d'exploitation sur lequel on les exécute, peuvent être incompatibles entre-elles et ne plus fonctionner après une mise à jour de l'OS
- Si chaque application s'exécute dans un conteneur différent, il faut juste s'assurer que les conteneurs soient correctement pris en charge

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

But dans les réseaux étendus

- Prenez Amazon, Netflix, Google, … qu'ont-ils en commun ?
- Un réseau étendu avec des montées en charge à certaines occasions (période de Noël, nouveau film sur la plateforme, évènement particulier, …)
- Il faut pouvoir augmenter rapidement le nombre de machines pour soutenir la charge
- Et dès que la tempête est passée, il faut pouvoir rediminuer le nombre de machines pour éviter de consommer des ressources pour rien

©Hainaut P. 2021 - www.coursonline.be

But dans les réseaux étendus

- Un système de conteneurs est idéal pour cela
- On peut déployer facilement et rapidement plusieurs instances d'un même conteneur sur plusieurs nœuds réseaux
- Que ce soit Titus pour Netflix, AWS pour Amazon, GKE pour Google, …
  tous les grands consommateurs de bande passante utilisent des systèmes de conteneurs

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

But dans les équipes

- Les développeurs veulent toujours avoir les dernières mises à jour des OS et des environnements d'applications
- Les administrateurs système veulent un système le plus stable possible
- Avec les conteneurs, ces deux mondes peuvent coexister ;-)
- Et les développeurs peuvent plus facilement se mettre à l'infra …
- Les "devops", le meilleur des deux mondes ?

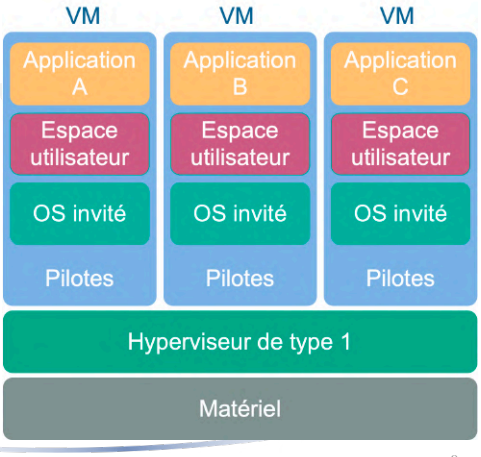
©Hainaut P. 2021 - www.coursonline.be

©Hainaut Patrick

Principes

Rappel: hyperviseur de type 1

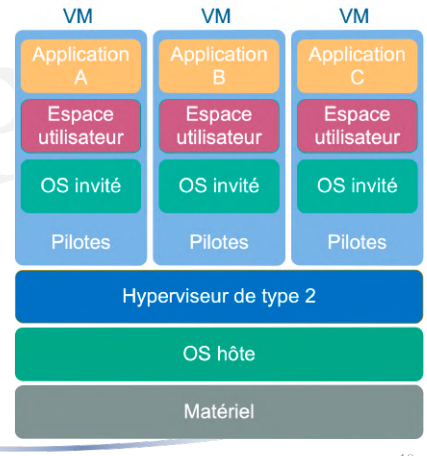
- Noyau système très léger et optimisé pour faciliter les accès au matériel par les OS invité
- La machine doit être dédiée à cela, avec un seul hyperviseur
- Souvent utilisé dans un environnement serveur
- Le matériel pris en charge peut être limité
- Exemples: KVM, Oracle VM,Citrix XEN Server, VMWare Sphere, Microsoft Hyper-V



©Hainaut P. 2021 - www.coursonline.be

Rappel: hyperviseur de type 2

- Système lourd (3 couches au lieu de 2) et gourmand; peu performant
- L'hyperviseur n'est qu'une application, qui peut être arrêtée, cohabiter avec un autre hyperviseur de même type
- Permet de tester rapidement un OS ou une application
- Exemples: Oracle VirtualBox,VMWare player, Microsoft VirtualPC

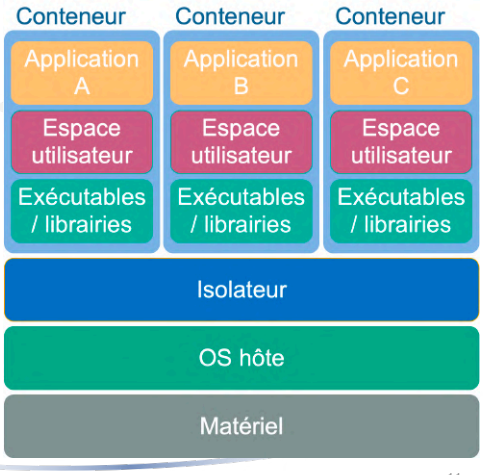


©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Rappel: isolateur

- Permet d'isoler l'exécution d'applications dans des zones différentes d'exécution appelées contextes
- On peut ainsi exécuter plusieurs instances d'une même application
- Exemples: OpenVZ, Docker



©Hainaut P. 2021 - www.coursonline.be

Hyperviseur vs Isolateur

- L'hyperviseur de type 2 offre le degré d'isolation nécessaire entre les applications et la bonne prise en charge par l'OS hôte
- Mais avec un hyperviseur de type 2:
  - Chaque VM prend beaucoup de ressources (mémoire vive, processeur, espace disque, réseau)
  - La machine hôte devra par conséquent être très puissante
- Le conteneur offre l'isolation nécessaire mais reste léger vu qu'il s'appuie sur le noyau de l'OS hôte

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Hyperviseur vs Isolateur

- On peut donc faire tourner plus de conteneurs que de machines virtuelles pour un environnement donné
- Par rapport à l'hyperviseur de type 1, Docker est plus facile à mettre en œuvre car un nombre important de conteneurs existent déjà et il suffit de les utiliser
- Avec Docker, la construction des conteneurs et des images est standardisé et leur mise à disposition et utilisation en est facilité

©Hainaut P. 2021 - www.coursonline.be

Conteneurs dans une VM

- On peut exécuter des conteneurs dans une VM
- Ca prend plus de ressources, bien sur, mais ça fonctionne et ajoute un degré d'isolation
- Cela est mis en œuvre dans les conteneurs HyperV et dans les projets Kata Containers et HyperKit

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Quelques exemples d'utilisation

- Tests de commandes d'un langage de programmation dont l'interpréteur s'exécute dans un conteneur Docker
- Tests de commandes sur une base de données s'exécutant dans un conteneur Docker et accessible via un port mappé localement

©Hainaut Patrick

Docker = Linux

- Docker est une technologie liée à Linux et à son noyau
- Pour faire fonctionner Docker sous Windows ou Mac OS X, jusqu'il y a peu, Docker Desktop (anciennement Docker Toolbox) comprenait VirtualBox et une image Linux minimale qui fournissait les services nécessaires à Docker

©Hainaut Patrick

Docker = Linux + Windows + Mac

- Depuis 2016, Docker a été porté sous Windows et sur MacOS
- Sous Windows, on peut exécuter des conteneurs Windows nativement et des conteneurs Linux ou Windows s'exécutant dans une VM HyperV très légère
- Sous Mac, c'est le même principe avec une VM Hyperkit très légère également
- Linux reste quand même l'OS privilégié pour utiliser Docker

©Hainaut P. 2021 - www.coursonline.be

Principe de fonctionnement de Docker

- libcontainer est une bibliothèque de bas niveau (driver) qui encapsule les fonctions essentielles à la virtualisation
- Elle s'appuie sur deux extensions du kernel Linux
  - Cgroups (Control Groups)
  - Namespaces

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

CGroups

- Cgroups (Control Groups) permet de fragmenter les ressources d'un hôte
  - Temps processeur
  - Mémoire allouée
  - Accès au réseau
  - …
- Le but est de contrôler la consomation des ressources par processus afin de garantir une bonne répartition de la charge

©Hainaut P. 2021 - www.coursonline.be

Namespaces

- Ils permettent de faire en sorte que des processus ne voient pas les ressources utilisées par d'autres processus
- Cela apporte l'isolation nécessaire à la création des conteneurs
- Les Namespaces dérivent de la commande chroot qui permet de changer, pour un processus donné, la racine du système
- Cela évite qu'un utilisateur ne se "promène" dans le système de fichiers (très utilisé avec un serveur ftp)

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Namespaces

- Voici les différents Namespaces Linux:

Namespace:	Isole:
IPC	Isole les communications interprocessus
Network	Donne une pile (stack) réseau privé
Mount	Donne un système de fichiers privé
PID	Identification du processus et de ses processus fils
User	Mise en correspondance (mapping) entre les utilisateurs de la machine hôte et du conteneur (container)
UTS	Gestion du nom de l'hôte (host)

©Hainaut P. 2021 - www.coursonline.be

Conteneur

- Un conteneur est donc un système de fichiers sur lequel s'exécute des processus (de préférence; un par conteneur) de manière:
  - contrainte: grâce à Cgroups qui limite les ressources
  - isolée: grâce à Namespaces qui isole les conteneurs

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

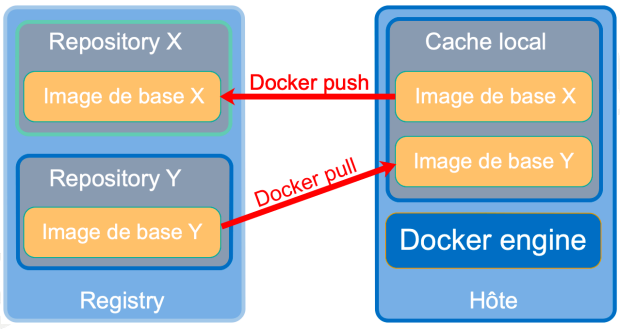
Apports de Docker

- Docker n'est pas le seul système de gestion de conteneurs
- Mais il apporte certains avantages qui le rendent très populaire:
  - Les images: des archives en lecture seule qui peuvent être échangées et réutilisées
  - Union file system:
    - un système de fichiers qui permet une organisation en couches du conteneur
    - un conteneur est un empilement ordonné d'images, chaque couche surchargeant la précédente en y apportant éventuellement des ajouts et modifications
  - Le registry:
    - un annuaire et un centre de stockage des images réutilisables
    - le principal est le Docker Hub officiel accessible via https://hub.docker.com
      On peut y créer un compte gratuit et y télécharger et publier gratuitement des images publiques

©Hainaut P. 2021 - www.coursonline.be

Apports de Docker: le registry

- Voici une illustration du fonctionnement du registry



- Au niveau du Docker Hub, on trouvera des images (contenant des applications; ex.: serveur Web) exploitables immédiatement

©Hainaut P. 2021 - www.coursonline.be

Apports de Docker

- Copy on write:
  - un conteneur va exécuter des processus qui vont vouloir écrire des données dans le système de fichiers
  - Les images étant en lecture seule, les données seront placées dans une couche conteneur (container layer)
  - Il en va de même lorsqu'on modifie un fichier de configuration, le fichier modifié est placé dans la couche conteneur et surcharge le fichier présent dans l'image, non modifié, c'est le principe du copy on write (cow)
  - Attention, que si on arrête un conteneur, la couche conteneur disparaît et les données qu'elle contient aussi
  - Pour sauvegarder les données et fichiers modifiés entre deux lancement du conteneur, il faudra utiliser un volume (notion abordée plus tard)

©Hainaut P. 2021 - www.coursonline.be

L'écosystème Docker: le moteur

- Docker Engine
- Il fournit:
  - une ligne de commande
  - des API
  - la capacité de créer des conteneurs à partir d'images ou de modèles
  - la capacité de gérer le cycle de vie des conteneurs
- Quelques alternatives compatibles avec les images Docker:
  - LXC
  - runC
  - Rkt
  - OpenVZ

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

L'écosystème Docker: l'OS hôte

- N'importe quelle distribution Linux récente
- Quelques OS spécialisés:
  - Boot2Docker : distribution Linux très compacte, utilisé dans kit Docker pour Windows et Mac quand il n'y avait pas de solution native
  - Container Linux (anciennement CoreOS)
  - Atomic (distribution de base de RedHat et CentOS)
  - Windows Server Core

©Hainaut P. 2021 - www.coursonline.be

L'écosystème Docker: composition

- Une bonne pratique est d'exécuter un processus par conteneur
- Une application est donc répartie sur plusieurs conteneurs
- Ces conteneurs doivent être associés, c'est la composition
- Ca permet de définir l'ordre de démarrage, de créer les volumes et de définir les liens entre conteneurs (et processus)
- Pour cela, on utilise (par exemple) Docker Compose avec des fichiers YAML

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

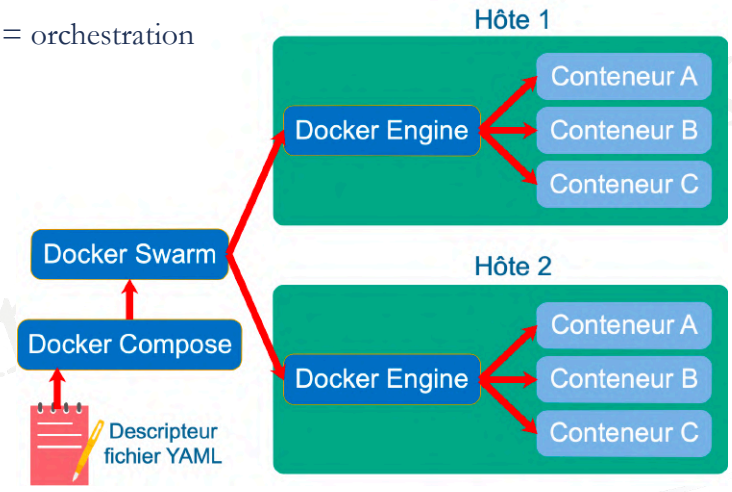
L'écosystème Docker: clustering

- Ces conteneurs peuvent être répartis sur plusieurs hôtes
- Il faut donc tenir compte de la distribution de la charge sur plusieurs machines et/ou plusieurs nœuds réseaux
- C'est le clustering et c'est géré, par exemple, par Docker Swarm ou Kubernetes (initié par google)

©Hainaut P. 2021 - www.coursonline.be

L'écosystème Docker: l'orchestration

- Composition + clustering = orchestration



©Hainaut P. 2021 - www.coursonline.be

L'écosystème Docker: l'orchestration

- Dans un premier temps, on est passé de multiples serveurs physiques exécutant chacun une application à des serveurs virtualisés sur une infrastructure physique
  -> modèle IaaS (Infrastructure as a Service)
- Avec les conteneurs, on passe au modèle CaaS (Container as a Service)
  dont l'orchestration est un des piliers
- Traditionnellement pour la configuration et le déploiement de l'infrastructures, on recours à plusieurs scripts de langages parfois différents
- Des outils, visant à uniformiser ces aspects ont vu le jour
  On trouve, par exemple: ansible, chef ou puppet

©Hainaut P. 2021 - www.coursonline.be

Un conteneur, qu'est ce que c'est ?

- C'est un processus lancé par **docker run** (exemple parmi d'autres)
- Partageant le noyau système de la machine hôte avec les autres conteneurs
- Isolé des autres processus, dans un environnement contrôlé par les **NameSpaces**
- Avec des ressources limitées par les **CGroups**
- Pouvant être reliés aux autres conteneurs par **docker compose** (exemple)
- Et pouvant être répartis sur plusieurs machines avec **docker swarm** (exemple)
- Les conteneurs sont particulièrement adaptés à une architecture micro-services

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

En pratique

PwD

Play with Docker

- Pour tester Docker sans l'installer, directement en ligne, gratuitement
- Accessible depuis https://labs.play-with-docker.com
- Ca permet de tester l'ensemble des fonctionnalités de Docker
- Les mises à jour sont faites régulièrement et on a l'assurance d'utiliser la dernière version de la plateforme

©Hainaut Patrick

Play with Docker

- Chaque session dure 4 heures
- Il est possible d'avoir jusqu'à 5 instances docker par session, basées sur une distribution alpine Linux
- Il est possible d'utiliser Docker Compose, Docker Machine, SWARM, …
- L'utilisation nécessite la création d'un compte gratuit sur le docker Hub accessible à l'adresse https://hub.docker.com

©Hainaut P. 2021 - www.coursonline.be

Play with Docker

- Lorsqu'on démarre une instance, on a accès à un shell avec les commandes docker disponibles
- C'est en fait un conteneur Docker dans lequel on exécute Docker, un DiD (Docker in Docker)
- C'est un playground, une sandbox, ce qui veut dire, qu'après la session, tout est détruit
- C'est surtout utilisé pour du training ou des démos

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

En pratique

Installation

Choix de l'édition

- Docker CE (Community Edition)
  - Open Source
  - Release stable tous les 6 mois
- Docker EE (Enterprise Edition)
  - 3 abonnements disponibles (basic, standard, avancé)
  - Release stable tous les 3 mois
  - 1 an de support
  - Technologie certifiée: infrastructure, conteneurs, plugins, …
- Nous installerons ici Docker CE

©Hainaut Patrick

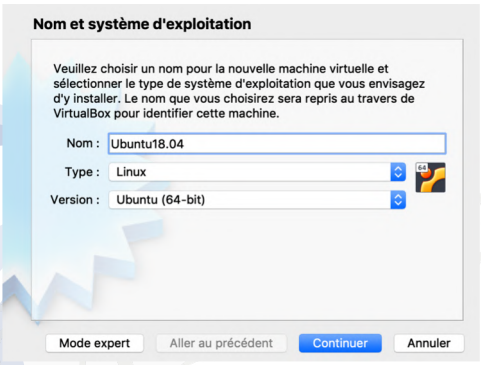
Choix de la plateforme

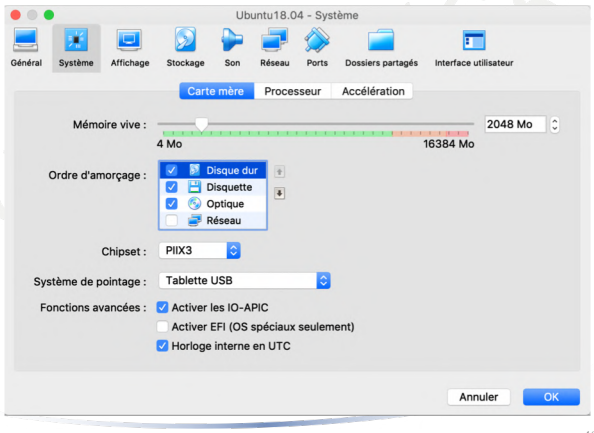
- Docker sera exploité au niveau d'un serveur, idéalement sous Linux
- Pour vos expérimentations, le mieux est d'installer une machine virtuelle Linux et d'installer Docker au niveau de cette machine virtuelle
- Des problèmes étant apparus pour faire fonctionner facilement Docker avec les distributions Red Hat (Google a fait du lobbying pour privilégier Kubernetes …), nous nous tournerons vers une distribution Ubuntu 20.04 LTS Server

©Hainaut P. 2021 - www.coursonline.be

Installation de Docker sous Ubuntu 20.04

- Sandbox: 2Gb de RAM, 10Gb de HDD (VDI dynamique)



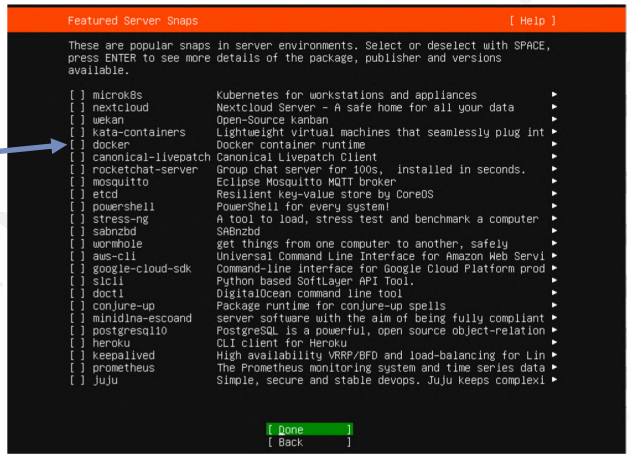


©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Installation de Docker sous Ubuntu 20.04

- Lors de l'installation d'Ubuntu, vous avez la possibilité de sélectionner le paquet Docker



©Hainaut P. 2021 - www.coursonline.be

Installation de Docker sous Ubuntu 20.04

- Mais il vaut mieux installer Docker manuellement (on verra pourquoi par la suite)
- Avec les privilèges root (sudo -s, une fois loggué):
  - apt update
  - apt install docker.io
- Pour que Docker puisse se lancer à chaque redémarrage, on tape la commande: systemctl enable --now docker

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Installation de Docker sous Ubuntu 20.04

- Docker est installé, mais seul un utilisateur avec les privilèges root peut l'exécuter
- Si vous voulez qu'un autre utilisateur puisse l'exécuter, il faut rajouter cet utilisateur au groupe docker
  usermod -aG docker *nom_de_l'utilisateur*
- Remarque: Lors de l'installation automatique avec l'OS, les fichiers se trouvent à des endroits différents, le groupe docker n'existe pas ainsi que certains fichiers, on préférera donc une installation manuelle

©Hainaut P. 2021 - www.coursonline.be

Fichier de configuration

- Quelle que soit la distribution de Linux employée (y compris sous PwD), le fichier de configuration qui spécifie les options de lancement du deamon dockerd est:
  /etc/docker/daemon.json
- Sous Windows, ce sera **%programdata%\docker\config\daemon.json**
- Les différentes options de dockerd peuvent se retrouver ici:
  https://docs.docker.com/engine/reference/commandline/dockerd/
- Suivant les environnements, ce fichier peut être absent et devra être créé si on désire un lancement particulier de Docker

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Socket de communication

- Si on examine le fichier docker.service, on remarque qu'il dépend de docker.socket

```
root@srv1804:~# cat /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the deleg
# exists and systemd currently does not support the cgroup featur
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

```
root@srv1804:~# cat /lib/systemd/system/docker.socket
[Unit]
Description=Docker Socket for the API
PartOf=docker.service

[Socket]
ListenStream=/var/run/docker.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker

[Install]
WantedBy=sockets.target
root@srv1804:~# _
```

- Le deamon dockerd va donc écouter par défaut sur un socket: /var/run/docker.sock
- Le client local docker communique sur ce socket par défaut

©Hainaut P. 2021 - www.coursonline.be

Socket de communication

- Un client distant pourra aussi communiquer avec le deamon via un socket TCP
- On pourrait par exemple communiquer de n'importe quelle adresse IP sur un port bien précis mais il faudrait sécuriser l'échange avec TLS
- Si on utilise la commande **dockerd -H unix:// -H tcp://*0.0.0.0:2375***
  on expose le deamon docker via TCP sans sécurité
- Le **–H unix://** permet de communiquer avec le deamon en local via le socket
- Le **–H tcp://*0.0.0.0:2375*** permet la communication via réseau
  2375 est choisi par convention lorsque l'échange n'est pas sécurisé, 2376 si sécurisé

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Socket de communication

- Un client distant pourra aussi se connecter à distance en ssh avec la commande **dockerd -H ssh://*utilisateur@IP commande***

```
MacBook-Pro-2:~ hainautpatrick$ docker -H ssh://hainautp@10.51.13.119 info
hainautp@10.51.13.119's password:
Containers: 2
 Running: 0
 Paused: 0
 Stopped: 2
Images: 2
Server Version: 19.03.8
Storage Driver: overlay2
 Backing Filesystem: <unknown>
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
```

- Remarque: la commande info donne plus de renseignements que ceux affichés ici

©Hainaut P. 2021 - www.coursonline.be

Exercices

- Expérimentez Play With Docker
- Installer une machine virtuelle Linux sur laquelle vous installer le Docker deamon
- Expérimentez les différentes façon de communiquer avec le Docker deamon

©Hainaut P. 2021 - www.coursonline.be

En pratique

Création et gestion d'un container

Création et gestion de containers Docker

- On crée un conteneur à partir d'une image:
  - qui contient le système de fichiers et l'application
  - qui est disponible en local ou dans un registry comme Docker Hub
- On utilise la commande:
  docker container run *[options] nom_image [commandes] [arg.]*
- La liste des options disponibles peut se trouver ici:
  https://docs.docker.com/engine/reference/commandline/run/
- Le containeur est stocké en local, sur l'hôte, dans un sous-répertoire de:
  /var/lib/docker/containers

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 1:
  #docker container run hello-world

```
root@srv1804:~# docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86d7e4e
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/

root@srv1804:~# _
```

Pas disponible localement, l'image est donc téléchargée depuis le docker hub

Le container est exécuté dans la foulée au premier plan (foreground)

Son exécution se limite à afficher un message, puis il est arrêté mais le container existe toujours

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 2:
  #docker container run centos echo hello world

```
root@srv1804:~# docker container run centos echo hello world
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Status: Downloaded newer image for centos:latest
hello world
root@srv1804:~# _
```

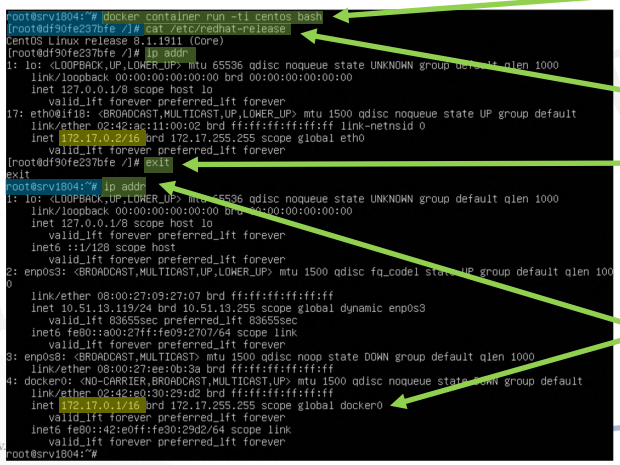
Ici, on télécharge une image de CentOS (la dernière version stable) et on exécute une commande à partir du container CentOS (echo hello world) et puis le container est arrêté
C'est également une exécution en foreground

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 3:
  #docker container run -ti centos bash



L'image est ici disponible en local (on vient de l'utiliser précédemment), elle s'exécute donc directement

On peut vérifier la version de l'OS

On arrête le container en tapant la commande **exit**

Si on regarde aux niveaux des adresses IP, on voit que le container est dans le même range que la machine hôte

©Hainaut P. 2021 - www.

Création et gestion de containers Docker

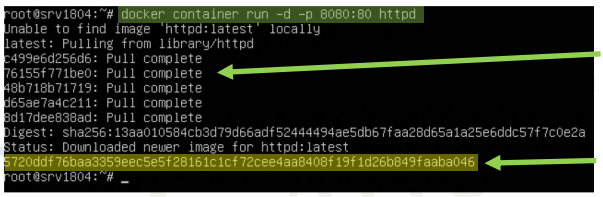
- Exemple 3:
  #docker container run -ti centos bash
- centos est une image présente dans le docker hub, on aurait pu choisir une image plus spécifique avec des applications incluses, une autre version de l'OS, …
- L'option **-t** permet d'allouer un pseudo terminal (TTY) au container
- L'option **-i** permet de garder l'entrée standard (STDIN) ouverte
- L'argument **bash** est facultatif, c'est le shell choisi par défaut
- Par défaut, ce container s'exécute en foreground

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 4:
  #docker container run -d -p 8080:80 httpd



L'image est plus volumineuse et se télécharge en plusieurs layers

C'est l'id long du container

- Ici, on télécharge un container Apache qu'on exécute en background (tâche de fond, option **-d**) et auquel on associe un mapping de port (option **-p *port hôte* : *port container***)
- La main est redonnée automatiquement à la machine hôte

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 4:
  #docker container run -d -p 8080:80 httpd
- Remarque: Si plusieurs containers utilise le même port sur la machine hôte (ici **8080**), il y aura conflit

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 4 bis: #**docker container run -d -P httpd**

- Avec l'option **–P**, Docker choisit un port hôte aléatoire dans la plage donné par /proc/sys/net/ipv4/ip_local_port_range, par défaut de 32768 à 60999
- Le port container est le port serveur par défaut (ici: 80)
- Pour récupérer le port assigné par Docker, on utilise la commande: **docker ps** et on regarde la colonne PORTS (ici: 32768)
- 0.0.0.0 signifie que le container accepte des requêtes de n'importe quelle adresse IP

```
root@srv1804:~# docker container run -d -P httpd
2ccd52d42a1237cd00a0ff5ec66a5186e23cc1a49fd73d52af1e6d57c68a8a73
root@srv1804:~# docker ps
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS
PORTS                     NAMES
2ccd52d42a12        httpd               "httpd-foreground"   8 seconds ago       Up 7 seconds
0.0.0.0:32768->80/tcp   admiring_kowalevski
```

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Pour tester l'exemple 4 sur le client, on peut télécharger elinks, un navigateur en ligne de commande via la commande: #**apt install elinks**

- On l'exécute via la commande: #**elinks http://localhost:8080** et la page d'index d'Apache qui tourne dans le container apparaît puisque le port 8080 de la machine hôte est associé au port 80 sur lequel Apache écoute par défaut dans le container

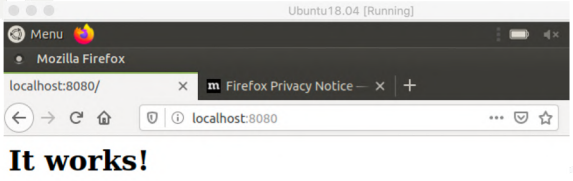
- Remarque: dans le cas de l'exemple 4 bis, on utiliserait: #**elinks http://localhost:32768**

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- On peut aussi installer sur notre Ubuntu Server une GUI
- On va éviter les poids lourd KDE et GNOME et choisir l'environnement MATE plus léger
- On l'installe par: #**apt install ubuntu-mate-core** et on choisit lightdm
- On le démarre par: #**service lightdm start**
- On peut jongler entre la GUI et la CLI avec les touches CTRL-ALT-F1 et CTRL-ALT-F7
- On peut vérifier l'exécution d'Apache dans le container

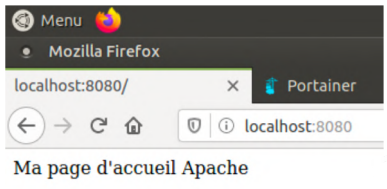


Création et gestion de containers Docker

- Ce serait bien de pouvoir changer la page d'accueil du serveur Apache de l'exemple 4
- On utilise pour ça la commande:
  #**docker exec -t -i *identifiant*** ou ***nom du container /bin/bash***

```
root@srv1804:~# docker exec -ti 5720ddf76baa /bin/bash
root@5720ddf76baa:/usr/local/apache2# cd htdocs
root@5720ddf76baa:/usr/local/apache2/htdocs# echo "Ma page d'accueil Apache" > index.html
root@5720ddf76baa:/usr/local/apache2/htdocs# exit
exit
root@srv1804:~# _
```

- La commande **exec** permet d'exécuter un processus dans container actif, dans cet exemple, un shell
- On change ici le contenu du fichier /usr/local/apache2/htdocs/index.html qui contient par défaut le fameux "It works!"



Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker images** ou **docker image ls**

```
root@srv1804:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
httpd               latest              8326be82abe6        2 weeks ago         166MB
centos              latest              470671670cac        2 months ago        237MB
hello-world         latest              fce289e99eb9        15 months ago       1.84kB
root@srv1804:~#
```

permet de voir les images présentes en local, l'option **-a** permet de voir aussi les images intermédiaires cachées, si il y en a

  - #**docker ps** ou **docker container ls**

```
root@srv1804:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
 PORTS                    NAMES
5720ddf76baa        httpd               "httpd-foreground"  2 hours ago         Up 2 hours
 0.0.0.0:8080->80/tcp   cranky_kare
```

permet de voir les containers (un container = un processus) actifs

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker ps -l**
    ou **docker container ls -l**

```
root@srv1804:~# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
 PORTS                    NAMES
5720ddf76baa        httpd               "httpd-foreground"  2 hours ago         Up 2 hours
 0.0.0.0:8080->80/tcp   cranky_kare
```

permet de voir le dernier container actif
(dans cet exemple, ça ne change rien par rapport à la commande précédente, puisque c'est le seul actif)

  - **#docker ps -a**
    ou **docker container ls -a**

```
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS                    NAMES
5720ddf76baa        httpd               "httpd-foreground"  2 hours ago         Up 2 hours
      0.0.0.0:8080->80/tcp   cranky_kare
df90fe237bfe        centos              "bash"              3 hours ago         Exited (0) 3 hours
ago                            strange_haibt
49c26434fd7d        centos              "bash"              3 hours ago         Exited (0) 3 hours
ago                            reverent_lewin
daed029d5cbc        centos              "bash"              3 hours ago         Exited (0) 3 hours
ago                            nostalgic_goldstine
8e6ca5d124b6        centos              "bash"              3 hours ago         Exited (0) 3 hours
ago                            bold_edison
5f66a2c965a4        centos              "echo hello world"  3 hours ago         Exited (0) 3 hours
ago                            modest_stonebraker
a016d1656440        hello-world         "/hello"            3 hours ago         Exited (0) 3 hours
ago                            trusting_gates
root@srv1804:~# _
```

permet de voir les tous containers présents, actifs ou pas

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker ps -a -q**
    ou **docker container ls -a -q**

```
root@srv1804:~# docker ps -a -q
5720ddf76baa
df90fe237bfe
49c26434fd7d
daed029d5cbc
8e6ca5d124b6
6f66a2c965a4
a016d1656440
root@srv1804:~#
```

permet de voir les tous les identifiants cours des containers , actifs ou pas

©Hainaut P. 2021 - www.coursonline.be

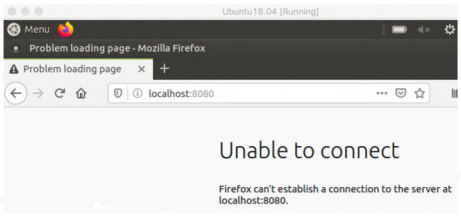
Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker stop**

```
root@srv1804:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                    NAMES
5720ddf76baa        httpd               "httpd-foreground"       2 hours ago         Up 2 hours
0.0.0.0:8080->80/tcp   cranky_kare
root@srv1804:~# docker stop 5720ddf76baa
5720ddf76baa
root@srv1804:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
root@srv1804:~#
```

permet d'arrêter un container en background en spécifiant son identifiant court, donné par la commande **docker ps**

Le serveur Apache n'est plus disponible, le container ayant été arrêté



©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker kill**

    Similaire à la commande docker stop, mais elle envoie un signal SIGKILL au processus principal s'exécutant dans le container, ce qui est assez brutal

    La commande docker stop envoie d'abord un signal SIGTERM avant d'envoyer, après un délai, le signal SIGKILL si le processus n'est pas arrêté

©Hainaut Patrick

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker start**

```
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
                    PORTS               NAMES
550bf0d18258        httpd               "httpd-foreground"       58 seconds ago      Exited (0) 15 seco
nds ago                                 competent_bassi
bd23484e08f9        hello-world         "/hello"                 About a minute ago  Exited (0) About a
 minute ago                             charming_spence
root@srv1804:~# docker start 550bf0d18258
550bf0d18258
root@srv1804:~# _
```

permet de démarrer un container présent localement en spécifiant son identifiant court, donné par la commande **docker ps -a**

©Hainaut Patrick

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker create**

```
root@srv1804:~# docker container create -p 8080:80 httpd
b49f8030ff58191a41ed48a2264f2a726cf05f9a1c5a4a20860b59b6fb5d3b27
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS
 PORTS               NAMES
b49f8030ff58        httpd               "httpd-foreground"   6 seconds ago       Created
                    competent_pike
root@srv1804:~# docker container start b49f8030ff58
b49f8030ff58
root@srv1804:~# _
```

permet créer un container mais sans le démarrer, ce qui pourra être fait plus tard avec **docker start**

Convient aux containers s'exécutant en tâche de fond, l'option **-d** ne devant pas être précisée

Si on utilise cette méthode avec le container hello-world, le container va s'exécuter automatiquement en tâche de fond et on ne verra pas le message, ce qui n'a aucun intérêt

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker pull**

```
root@srv1804:~# docker pull debian
Using default tag: latest
latest: Pulling from library/debian
7e2b2a5af8f6: Pull complete
Digest: sha256:7790bb087ef265cd56265767a8b689d52c7bc73b30b3cbb37525a9ea66fa9740
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
root@srv1804:~# _
```

permet de télécharger une image sans l'instancier dans un container

L'image est en local et ne devra plus être téléchargée dans le docker hub

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker rm**
    ou **docker container rm**

```
root@srvdocker:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
    PORTS               NAMES
563a46ce0f1d        hello-world         "/hello"            7 hours ago         Exited (0) 7 hours a
go                      recursing_euclid
3a08de19ac33        hello-world         "/hello"            3 weeks ago         Exited (0) 3 weeks a
go                      happy_jepsen
root@srvdocker:~# docker rm 3a08de19ac33
3a08de19ac33
root@srvdocker:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
    PORTS               NAMES
563a46ce0f1d        hello-world         "/hello"            7 hours ago         Exited (0) 7 hours a
go                      recursing_euclid
root@srvdocker:~# _
```

permet de supprimer un container arrêté en spécifiant son identifiant court, donné par la commande **docker ps -a**

Pour supprimer un container en cours d'exécution sans le stopper, on doit utiliser la commande **docker rm -f *identifiant*** (ou ***nom***, voir plus loin)

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Exemple 5:
  #docker container run -ti --rm centos bash

```
root@srv1804:~# docker container run -ti centos bash
[root@833967e4881a /]# exit
exit
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
        PORTS               NAMES
833967e4881a        centos              "bash"              19 seconds ago      Exited (0) 16 second
s ago                       pensive_poitras
root@srv1804:~# docker container run -ti --rm centos bash
[root@32e2f8c7b9ad /]# exit
exit
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
        PORTS               NAMES
833967e4881a        centos              "bash"              41 seconds ago      Exited (0) 38 second
s ago                       pensive_poitras
root@srv1804:~# _
```

L'option **--rm** permet de supprimer un container une fois son exécution terminée
On voit dans la photo écran que le container n'apparait pas dans la liste des containers

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker rmi**
    ou **docker image rm**

```
root@srv1804:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
httpd               latest              8326be82abe6        2 weeks ago         166MB
centos              latest              470671670cac        2 months ago        237MB
hello-world         latest              fce289e99eb9        15 months ago       1.84kB
root@srv1804:~# docker rmi hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must forc
e) - container a016d1656440 is using its referenced image fce289e99eb9
root@srv1804:~#
```

permet de supprimer une image en spécifiant son nom ou son identifiant court, donnés par la commande **docker images**

Il faut pour cela que tous les containers utilisant cette image soient supprimés, à moins d'utiliser le paramètre **-f** dans la commande précédente: **docker –f rmi *nom_image***

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker rm $(docker ps -aq)** permet de supprimer tous les containers arrêtés et **docker rm –f $(docker ps -aq)** permet de les supprimer tous

```
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS               NAMES
5720ddf76baa        httpd               "httpd-foreground"       6 hours ago         Exited (0) 3 hours
ago                 cranky_kare
df90fe237bfe        centos              "bash"                   6 hours ago         Exited (0) 6 hours
ago                 strange_haibt
49c26434fd7d        centos              "bash"                   6 hours ago         Exited (0) 6 hours
ago                 reverent_lewin
daed029d5cbc        centos              "bash"                   6 hours ago         Exited (0) 6 hours
ago                 nostalgic_goldstine
8e6ca5d124b6        centos              "bash"                   6 hours ago         Exited (0) 6 hours
ago                 bold_edison
6f66a2c965a4        centos              "echo hello world"       7 hours ago         Exited (0) 7 hours
ago                 modest_stonebraker
a016d1656440        hello-world         "/hello"                 7 hours ago         Exited (0) 7 hours
ago                 trusting_gates
root@srv1804:~# docker rm $(docker ps -a -q)
5720ddf76baa
df90fe237bfe
49c26434fd7d
daed029d5cbc
8e6ca5d124b6
6f66a2c965a4
a016d1656440
root@srv1804:~#
```

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker rmi**
    ou **docker image rm**

```
root@srv1804:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
httpd               latest              8326be82abe6        2 weeks ago         166MB
centos              latest              470671670cac        2 months ago        237MB
hello-world         latest              fce289e99eb9        15 months ago       1.84kB
root@srv1804:~# docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86d7e4e
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
Deleted: sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3
root@srv1804:~# docker rmi 470671670cac
Untagged: centos:latest
Untagged: centos@sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Deleted: sha256:470671670cac686c7cf0081e0b37da2e9f4f768ddc5f6a26102ccd1c6954c1ee
Deleted: sha256:0683de2821778aa9546bf3d3e6944df779daba1582631b7ea3517bb36f9e4007
root@srv1804:~#
```

Une fois les containers supprimés, on peut supprimer les images

Pour toutes les supprimer, on utilise la commande: **docker rmi $(docker images -aq)**
On peut forcer la suppression des images sans supprimer les containers avec l'option **–f**, mais les containers doivent être arrêtés et ils ne seront pas supprimés pour autant

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker container run -ti --name centos centos bash**

```
root@srv1804:~# docker container run -ti --name centos centos
[root@380e85d5a733 /]# cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@380e85d5a733 /]# exit
exit
root@srv1804:~# docker rm centos
centos
root@srv1804:~#
```

On reprend l'exemple 3 mais on rajoute une option **--name** ***nom_container***
qui permet de spécifier un nom au container, ce qui permettra d'utiliser ce nom à la place de l'identifiant court

Ce nom remplace le nom donné par défaut par Docker, qui peut aussi s'utiliser à la place de l'identifiant mais qu'on retiendra moins facilement

Création et gestion de containers Docker

- Quelques commandes utiles:

```
root@srv1804:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
          PORTS               NAMES
b145ccca3454        centos              "/bin/bash"              36 seconds ago      Exited (127) 24 sec
onds ago                                centos
4a2c5129adf1        hello-world         "/hello"                 5 minutes ago       Exited (0) 5 minute
s ago                                   distracted_perlman
2436787ae73c        hello-world         "/hello"                 5 minutes ago       Exited (0) 5 minute
s ago                                   jolly_noyce
2ccd52d42a12        httpd               "httpd-foreground"       37 minutes ago      Up 37 minutes
          0.0.0.0:32768->80/tcp   admiring_kowalevski
root@srv1804:~# _
```

La commande docker **ps -a** permet de voir les noms des containers

On remarque que si on lance deux instances d'une même image, on aura deux containers différents avec des ID et des noms différents

On ne peut donc pas employer deux fois le même nom

```
root@srv1804:~# docker container run -ti --name centos centos
docker: Error response from daemon: Conflict. The container name "/centos" is already in use by cont
ainer "b145ccca345458fe1bffb796f978ee6ca6dbb693e39b9a8b0818ca3cb529f88f". You have to remove (or ren
```

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:

Remarques:

le paramètre récursif **$(docker ps –aq)** peut s'appliquer à toute commande dont on veut donner une portée sur tous les containers (**start**, **stop**, …)

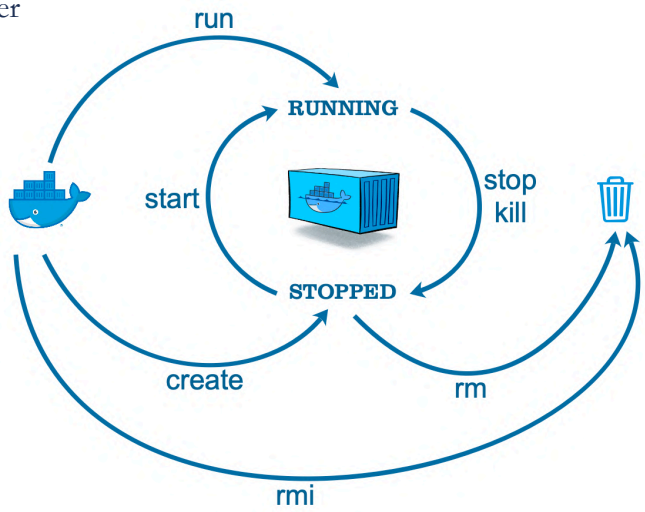
Les commandes Docker ont été revues et scindées en fonction de leur portée: container, image, … C'est la raison pour laquelle certaines commandes anciennes et nouvelles coexistent et effectuent la même tâche comme **docker ps** qui devient **docker container ls** ou **docker run** qui devient **docker container run**

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Cycle de vie d'un container Docker



©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - **#docker system prune**

```
root@srv1804:~# docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
23975358b4dc8794854820578605 4d842731994e91dbd4289c41872d7996ecb8
e75cf0981e25e0e6b7ad8a1087477d9e3ba4bfdf64ff93add390895ee8ed52b7
5964c93ea42fbc3d07b23fa129b93202cfdc0343c50047a4f77f1f645f21a646
d33301b3466a3202b7224f3905b95da7721cce70cb0de3d3f4ceefd9e0dfb7ff

Total reclaimed space: 0B
root@srv1804:~# _
```

permet de supprimer les containers arrêtés, et tous les objets non rattachés (parties d'images orphelines, volumes orphelins, …), inutiles et qui encombrent le système

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker inspect**

    permet de vérifier tous les paramètres du container, image, volume(s), port(s), réseau(x), …, et les paramètres de l'hôte également

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker inspect --format '{{** ***clé*** **}}'** ***identifiant ou nom du container***

    La commande **inspect** donne un résultat volumineux et parfois on a besoin de récupérer une donnée spécifique pour envoyer à un script par exemple

```
root@srv1804:~# docker inspect --format '{{ .NetworkSettings.IPAddress}}' web
172.17.0.4
root@srv1804:~#
```

    On utilise la notation **go template** qui permet de spécifier la ou les clés désirées

    Pour avoir plus de renseignements: https://docs.docker.com/engine/reference/commandline/inspect/

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Création et gestion de containers Docker

- Quelques commandes utiles:
  - #**docker container logs**

```
root@srv1804:~# docker container run -d --name centos centos ping 8.8.8.8
f7357bf689e17e2ff48b487a6caa7be04baa0efc236f93ff06701e7184c98c01
root@srv1804:~# docker container logs -f centos
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=52 time=15.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=52 time=15.10 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=52 time=16.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=52 time=18.7 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=52 time=23.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=52 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=52 time=20.1 ms
^C
root@srv1804:~# _
```

permet d'afficher les logs d'un container, l'option **-f** donnant les logs en temps réel

On écrira les logs sur la sortie standard et pas dans le container

©Hainaut P. 2021 - www.coursonline.be

Exercices

- Utilisez l'image Alpine Linux qui est une distribution Linux très légère et sécurisée
- Démarrer un container Alpine sans spécifier de commande. Que se passe t-il ? Avec la commande hello. Que se passe t-il ?
- Même chose mais en mode interactif sans spécifier de commande. Quelle commande a été lancée par défaut ?
- Démarrer un container Alpine en spécifiant la commande ping 8.8.8.8. Sortez du container. Quid de la commande ping ?
- Même chose en mode interactif. Sortez du container. Quid de la commande ping ?
- Même chose en background. Quid de la commande ping ?

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Exercices

- Démarrez un container en background basé sur nginx et publiez le port 80 du container sur le port 8080 de l'hôte. Vérifiez sur l'hôte la page par défaut de nginx
- Lancez un second container en publiant le même port. Que se passe t-il ?
- Lancez un autre container en publiant un port différent. Que se passe t-il ?
- Mêmes questions au niveau du port de destination

©Hainaut P. 2021 - www.coursonline.be

Exercices

- Listez les containers: tous, ceux en exécution, ceux qui sont stoppés, uniquement les ID
- Inspectez un container en exécution et un container arrêté
- Testez la notation go template pour filtrer les résultats de la commande inspect
- Sur un container Alpine démarré en background, lancez une commande ping 8.8.8.8
- Sur le même container, lancez un shell sh, en mode interactif
- Listez les processus du container. Qu'observez vous par rapport aux identifiants des processus ?

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Exercices

- Listez tous les containers (actifs et inactifs)
- Stoppez tous les containers encore actifs
- Vérifiez qu'il n'y a plus de containers actifs
- Listez les containers
- Supprimez tous les containers
- Vérifiez qu'il n'y a plus de containers

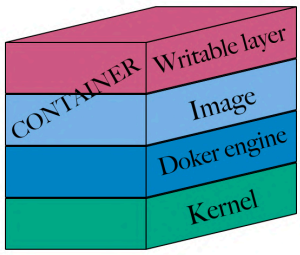
©Hainaut Patrick

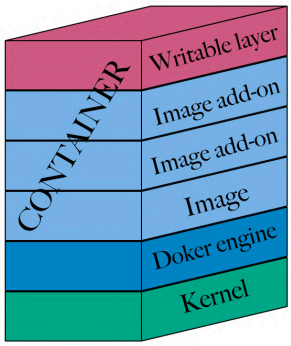
En pratique

Persistance des données

Docker: persistances des données

- Lorsqu'on lance un container, c'est une image en lecture seule qui est instanciée
- Pour pouvoir tenir compte des changements, une layer en lecture/écriture est superposée aux layers de l'image





©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Cette layer se trouve dans un des sous-répertoire de: /var/lib/docker/overlays2 où sont stockées les images
- Problème: elle est créée et détruite avec le container, donc pas de persistance de données en dehors de la vie du container
- Reprenons l'exemple 3 pour illustrer ce principe en créant un fichier à l'intérieur du container et en vérifiant son emplacement et sa disponibilité avant et après destruction du container

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 3 bis:

  #docker container run -ti --name shell centos bash

```
root@srv1804:~# docker container run -ti --name shell centos bash
[root@432c0d895c21 /]# touch test.txt
[root@432c0d895c21 /]# exit
exit
root@srv1804:~# cd /var/lib/docker/overlay2/
root@srv1804:/var/lib/docker/overlay2# find -name test.txt
./4b3219a8caa5c76b3dd5b554a2aef6a6716d7c5df67b1464743141c35d0ad644/diff/test.txt
root@srv1804:/var/lib/docker/overlay2# docker rm shell
shell
root@srv1804:/var/lib/docker/overlay2# find -name test.txt
root@srv1804:/var/lib/docker/overlay2#
```

On voit que le fichier **test.txt** est détruit en même temps que le container

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Pour avoir une persistance des données après destruction du container, il faut stocker les données en dehors de l'union filesystem

- Pour cela, on va créer un volume
  - À la création de l'image, dans le Dockerfile (voir plus loin)
  - Avec l'instruction **docker volume create**
  - A la création/lancement du container, avec l'option **-v** ou **-mount**

- Illustrons les 3 cas par des exemples

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 6:

#docker container run -d --name db mongo:4.0

```
root@srv1804:~# docker container run -d --name db mongo:4.0
bb03096887593853de9327687c30084c4d3613368cbffea551667dd045bf677f
root@srv1804:~# docker container inspect db|more
```

On instancie une image MongoDB 4.0 et si on inspecte sa structure, on voit que deux volumes sont créés en local qui pointent respectivement vers /data/configdb et /data/db, dans le container

©Hainaut P. 2021 - www.coursonline.be


```
        "Mounts": [
            {
                "Type": "volume",
                "Name": "e14899835979070facbba7783a3ab4466227697d19f70678b227987b1d5a1088",
                "Source": "/var/lib/docker/volumes/e14899835979070facbba7783a3ab4466227697d19f70678b
227987b1d5a1088/_data",
                "Destination": "/data/configdb",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            },
            {
                "Type": "volume",
                "Name": "4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5ac84a278",
                "Source": "/var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a93673957968 1e8c7545
06ad4c5ac84a278/_data",
                "Destination": "/data/db",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
```


Docker: persistances des données

- Exemple 6:

#docker container run -d --name db mongo:4.0

Ces volumes ont été prévus lors de la construction de l'image MongoDB 4.0 et créés lors de la création du container

On peut aller à l'emplacement source voir les données qui ont été stockées en local, sur l'hôte, par le container, et on voit que après sa destruction, les données sont toujours présentes

©Hainaut P. 2021 - www.coursonline.be

```
root@srv1804:~# cd /var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5a
c84a278/_data/
root@srv1804:/var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5ac84a27
8/_data# ls
WiredTiger          collection-0-4631813715409728246.wt  index-5-4631813715409728246.wt
WiredTiger.lock     collection-2-4631813715409728246.wt  index-6-4631813715409728246.wt
WiredTiger.turtle   collection-4-4631813715409728246.wt  journal
WiredTiger.wt       diagnostic.data                      mongod.lock
WiredTigerLAS.wt    index-1-4631813715409728246.wt       sizeStorer.wt
_mdb_catalog.wt     index-3-4631813715409728246.wt       storage.bson
root@srv1804:/var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5ac84a27
8/_data# docker rm -f db
db
root@srv1804:/var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5ac84a27
8/_data# ls
WiredTiger          collection-0-4631813715409728246.wt  index-5-4631813715409728246.wt
WiredTiger.lock     collection-2-4631813715409728246.wt  index-6-4631813715409728246.wt
WiredTiger.turtle   collection-4-4631813715409728246.wt  journal
WiredTiger.wt       diagnostic.data                      mongod.lock
WiredTigerLAS.wt    index-1-4631813715409728246.wt       sizeStorer.wt
_mdb_catalog.wt     index-3-4631813715409728246.wt       storage.bson
root@srv1804:/var/lib/docker/volumes/4db2ea841f10f68b20fd1fc8083a0a936739579681e8c754506ad4c5ac84a27
8/_data#
```

Docker: persistances des données

- Il serait plus facile de définir soi-même l'emplacement et le nom du volume, cela faciliterait la récupération des données
- On peut créer un volume avec la commande **docker volume create**

  Exemple: #**docker volume create *mongo***
- Cela va créer un volume appelé mongo dans: /var/lib/docker/volumes/
- Les données seront stockées dans: /var/lib/docker/volumes/mongo/_data/

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 7:

  #docker container run -d --name db –v mongo:/data/db mongo:4.0

On utilise le volume créé pour le mapper au répertoire correspondant dans le container

On constate la persistance de données et l'on voit comment supprimer un volume

```
root@srv1804:~# docker volume create mongo
mongo
root@srv1804:~# docker container run -d --name db -v mongo:/data/db mongo:4.0
90c455379ed816037b9b65dd1da15ab654bf792691feb428819c098a2e3241bd
root@srv1804:~# ls /var/lib/docker/volumes/mongo/_data/
WiredTiger           collection-0--8406426223734682251.wt  index-5--8406426223734682251.wt
WiredTiger.lock      collection-2--8406426223734682251.wt  index-6--8406426223734682251.wt
WiredTiger.turtle    collection-4--8406426223734682251.wt  journal
WiredTiger.wt        diagnostic.data                       mongod.lock
WiredTigerLAS.wt     index-1--8406426223734682251.wt       sizeStorer.wt
_mdb_catalog.wt      index-3--8406426223734682251.wt       storage.bson
root@srv1804:~# docker container rm -f db
db
root@srv1804:~# ls /var/lib/docker/volumes/mongo/_data/
WiredTiger           collection-0--8406426223734682251.wt  index-5--8406426223734682251.wt
WiredTiger.lock      collection-2--8406426223734682251.wt  index-6--8406426223734682251.wt
WiredTiger.turtle    collection-4--8406426223734682251.wt  journal
WiredTiger.wt        diagnostic.data                       mongod.lock
WiredTigerLAS.wt     index-1--8406426223734682251.wt       sizeStorer.wt
_mdb_catalog.wt      index-3--8406426223734682251.wt       storage.bson
root@srv1804:~# docker volume rm mongo
mongo
root@srv1804:~# _
```

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 8:
  #docker container run -d -p 8081:80 --name web –v /home/hainautp/html:/usr/local/apache2/htdocs httpd
- Pour l'exemple 4 (slide 60), on avait modifié la page d'accueil d'Apache avec la commande **exec**
- Cependant, si on détruit le container, on perd toutes les modifications et les données éventuellement générées, il n'y a pas de persistance des données
- Pour gérer cette persistance, on peut utiliser ici un bind-mount qui va nous permettre de monter un répertoire de l'hôte dans le container (équivalent de la commande **mount**)
- Le répertoire local est mappé dans le container, et son contenu persiste si le container est détruit
- Si le répertoire hôte n'est pas créé (web dans cet ex.), il le sera par la commande Docker

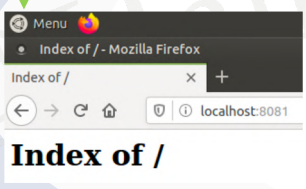
©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

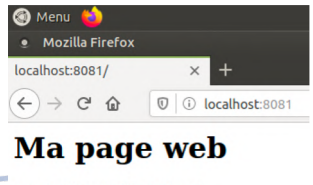
- Exemple 8:

  Si on ne met pas de fichiers html dans le répertoire /home/hainautp/web,
  on ne verra rien lorsqu'on voudra afficher le contenu de http://localhost:8081,
  le répertoire local prenant la place du répertoire correspondant du container

```
root@srv1804:~# docker container run -d -p 8081:80 --name web -v /home/hainautp/html:/usr/local/apac
he2/htdocs httpd
f0b8ba61e2d3aa81b8e6a87f788462d12f95b8d400140b03d3cdd31a1b9436
root@srv1804:~# cd html
root@srv1804:~/html# vi index.html
```




```
<html>
        <H1>Ma page web</H1>
        construite sur l'hote local
</html>
~
```



©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 9:
  #docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
- Là, on va déployer une image portainer qui est un outil de gestion de containers qui répond sur le port 9000, d'où le mapping de ports
- Cette image s'appelle **portainer/portainer**
- Comme elle gère les containers à partir de l'hôte, elle a besoin d'accéder au socket de communication Docker à partir du container dans lequel elle s'exécute
- Portainer va pouvoir gérer les images, les containers, les services, les volumes, le réseau, … bref, tout le cycle de vie des containers localisés sur l'hôte Docker

©Hainaut P. 2021 - www.coursonline.be

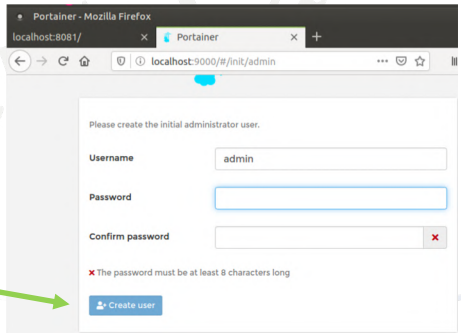
Docker: persistances des données

- Exemple 9:

```
root@srv1804:~# docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
Unable to find image 'portainer/portainer:latest' locally
latest: Pulling from portainer/portainer
d1e017099d17: Pull complete
a7dca5b5a9e8: Pull complete
Digest: sha256:4ae7f14330b56ffc8728e63d355bc4bc7381417fa45ba0597e5dd32682901080
Status: Downloaded newer image for portainer/portainer:latest
d58351c1d105bced22167bbc2ca3df25c6790a596827a1f06b255c7de3d13fa5
```

Après avoir lancé le container,
on peut se connecter à l'application,
à l'adresse http://localhost:9000

Il faut créer un utilisateur



©Hainaut P. 2021 - www.coursonline.be

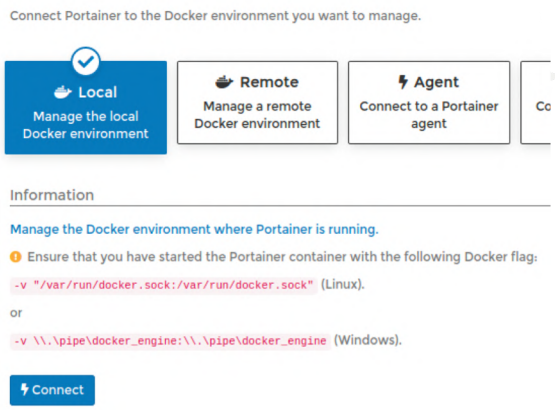
©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 9:

On choisira dans le cadre de cet exemple, l'administration locale des containers

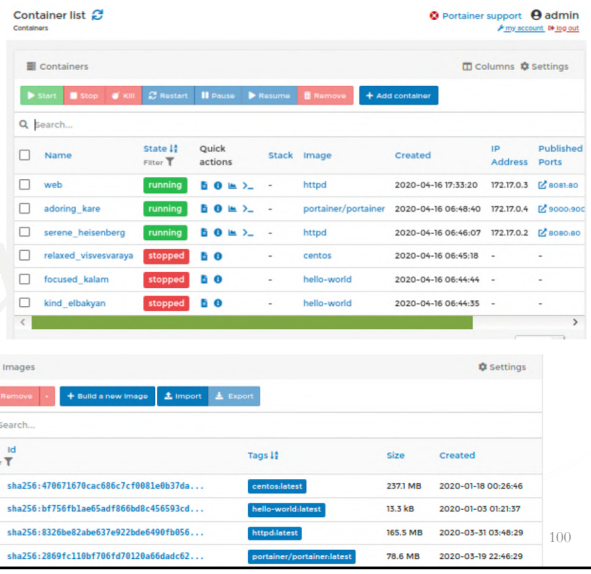
L'application nous engage à vérifier que le socket Docker est bien mappé



©Hainaut P. 2021 - www.coursonline.be

Docker: persistances des données

- Exemple 9:
  L'utilisation de Portainer est assez intuitive, expérimentez !



©Hainaut P. 2021 - www.coursonline.be

Exercices

- Démarrer un container Alpine en mode interactif sans spécifier de commande.
- Créez un fichier texte, sauvez-le. Stoppez le container et inspectez-le pour retrouver votre fichier
- Supprimer le container. Quid de votre fichier ?
- Spécifiez un volume au démarrage du container. Mêmes questions
- Créez un volume et renseignez-le lors du démarrage du container. Mêmes questions

©Hainaut Patrick

En pratique

Les applications multi-containers: Docker Compose

Docker Compose

- Cet outil va permettre d'utiliser des applications multi-container
- Utilisé notamment dans des architectures micro-services
- Toute la configuration de l'application se trouvera dans le fichier **docker-compose.yml**
- L'application pourra être lancée sur un hôte unique par le binaire, écrit en Python, **docker-compose**

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: le fichier docker-compose.yml

- On va y retrouver la définition:
  - Des services
  - Des réseaux (networks)
  - Des volumes
  - Des données sensibles (secrets) si l'application est déployée sur un cluster avec Docker Swarm
  - Des données de configuration (configs) si l'appli. est déployée sur un cluster avec Docker Swarm
- Lien vers la documentation officielle: https://docs.docker.com/compose/compose-file

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: le fichier docker-compose.yml

- Exemple de fichier docker-compose.yml
- Il est à noté que c'est le nom par défaut et que ce fichier peut être appelé autrement tant que l'extension reste .yml
- Il peut même y en avoir plusieurs pour une application

```yaml
version: "3"

services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"
    networks:
      - front-tier
      - back-tier

  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
    networks:
      - front-tier
      - back-tier

  worker:
    build:
      context: ./worker
    depends_on:
      - "redis"
      - "db"
    networks:
      - back-tier

  redis:
    image: redis:alpine
    container_name: redis
    ports: ["6379"]
    networks:
      - back-tier

  db:
    image: postgres:9.4
    container_name: db
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - "db-data:/var/lib/postgresql/data"
    networks:
      - back-tier

volumes:
  db-data:

networks:
  front-tier:
  back-tier:
```

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: le fichier docker-compose.yml

- Composition d'un service du fichier docker-compose.yml
  - L'image (obligatoire)
  - Le nombre d'instances de l'image
  - Les ports publiés à l'extérieur
  - La stratégie de redémarrage en cas de problème
  - Le Health check
  - Les contraintes de déploiement (Swarm)
  - Les configurations de mises à jour (Swarm)
  - Les secrets utilisés (Swarm)
  - Les configs utilisés (Swarm)
  - …

```yaml
db:
  image: postgres:9.4
  container_name: db
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
  volumes:
    - "db-data:/var/lib/postgresql/data"
  networks:
    - back-tier
```

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: le fichier docker-compose

- Gère une application définie par un ou plusieurs fichiers .yml (docker-compose.yml par défaut)
- On peut par exemple avoir un fichier .yml par type d'environnement (Windows, Linux, …)
- Indépendant du Docker deamon
- Le format de commande est: **docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]**
- Lien vers la documentation officielle: https://docs.docker.com/compose/reference/overview

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: le fichier docker-compose

- Commandes principale du fichier docker-compose:

up / down	création / suppression d'une application
start / stop	démarrage / arrêt d'une application
scale	spécifie le nombre de containers pour un service
build	création des images des services définis dans l'application
pull	Récupère les images nécessaires à partir d'un registry
ps	liste les containers de l'application
logs	Montre les logs de l'application ou d'un ou plusieurs services

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: service discovery

- Le DNS qui est intégré au Docker deamon permet de résoudre l'IP d'un service à partir de son nom
- Dans le code applicatif d'un service, on va pouvoir utliser le nom d'un service en référence
- Il faudra juste insérer, dans le code, un mécanisme qui permet d'attendre la disponibilité du service en question pour éviter d'avoir des erreurs d'exécution
- En effet, Docker Compose permet de gérer les dépendances entre services mais pas de définir quand un service est disponible par rapport à un autre

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: bonnes pratiques

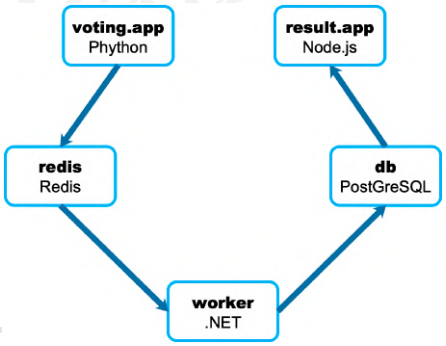
- Il y a certaines choses qui changeront entre l'environnement de développement et de production au niveau configuration et sécurité
  - Pas de bind-mounts
  - Pas de bindings de ports
  - Modification des variables d'environnement
  - Règles de redémarrage
  - Optimisation des ressources utilisées
  - Services supplémentaires comme du monitoring
  - …

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: exemple: Voting-app

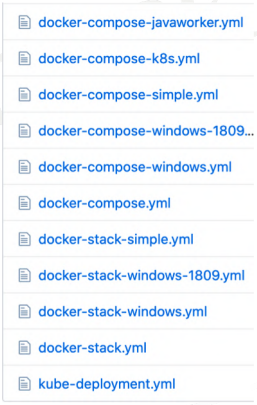
- Docker a publié un exemple didactique utilisant Docker Compose, la voting-app
- Il est disponible là: https://github.com/dockersamples/example-voting-app
- Elle se compose de 5 services:
  - voting.app est un frontend écrit en Phyton qui permet à l'utilisateur de voter
  - redis est une base de données intermédiaire Redis, qui permet de stocker le vote
  - worker récupère le vote dans redis pour le stocker dans PostGre
  - result.app est le frontend qui permet d'afficher les votes, écrit en Node.js



©Hainaut P. 2021 - www.coursonline.be

Docker Compose: exemple: Voting-app

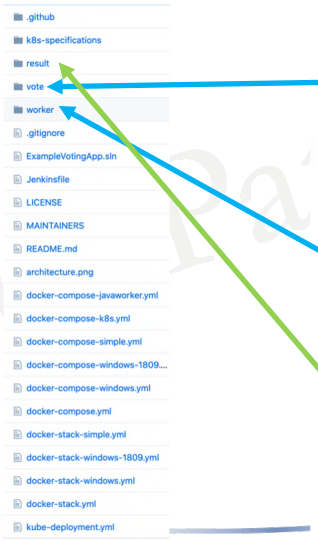
- Si on regarde la dossier GitHub de l'application, on voit qu'il y a plusieurs fichiers .yml
- Les -compose sont réservés au développement et les -stack sont réservés à la production
- Certains fichiers dépendent de la plateforme (Linux, Windows)
- Certains présentent des variantes au niveau d'un service (java au lieu de .NET pour le service worker)
- D'autres, des différences au niveau de l'orchestrateur (kubernetes au lieu de swarm)



©Hainaut P. 2021 - www.coursonline.be

Docker Compose: exemple: Voting-app

- L'instruction **build** précise où se trouvent les fichiers du service,le dockerfile et les fichiers nécessaires pour construire l'image
- L'instruction **volumes** précise le bind-mount du code applicatif,ce qui permet d'avoir le code source accessible à partir du container
- L'instruction **command** permet de changer la commande d'exécution du container, ce qui peut faciliter le débug ou la modification de code
- Le service de vote sera accessible sur le port 5000 et celui de result sur le port 5001




```yaml
version: "3"

services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"

  redis:
    image: redis:alpine
    ports: ["6379"]

  worker:
    build: ./worker

  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"

  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
```

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: exemple: Voting-app

- Si on compare la configuration d'un service entre docker-compose-simple.yml et docker-stack-simple.yml, on constate que:
  - **build** est remplacé par image car en production, on va déployer l'image et pas le code applicatif
  - **deploy** sera employé lors du déploiement sur un cluster swarm et ignoré avec docker-compose
  - Cette instruction permet de préciser le nombre d'instances du service, les conditions de mise à jour, les conditions de redémarrage du service, …
  - …

```yaml
vote:
  build: ./vote
  command: python app.py
  volumes:
   - ./vote:/app
  ports:
    - "5000:80"
```

```yaml
vote:
  image: dockersamples/examplevotingapp_vote:before
  ports:
    - 5000:80
  networks:
    - frontend
  depends_on:
    - redis
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure
```

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker Compose: exemple: Voting-app

- Si on regarde au niveau des networks, on en définit 2; frontend et backend
- vote, redis et worker qui communiquent ensemble utilisent frontend
- Worker, db et result utilisent backend

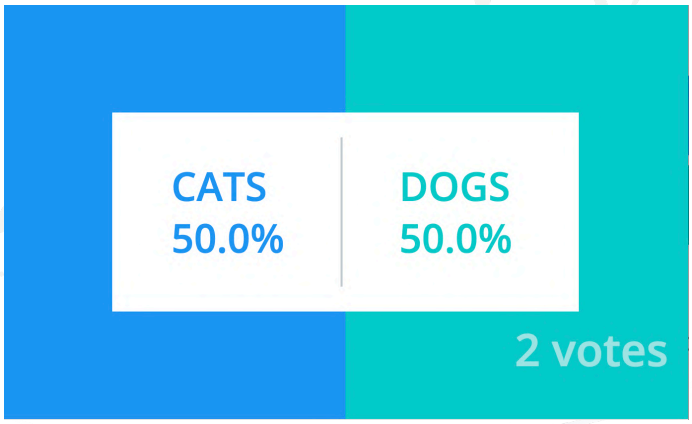
```yaml
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
```

```yaml
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    depends_on:
      - db
      - redis
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
  db-data:
```

©Hainaut P. 2021 - www.coursonline.be

Exercice

- Etudiez dans le détail le fonctionnement de l'application Voting-app



©Hainaut P. 2021 - www.coursonline.be

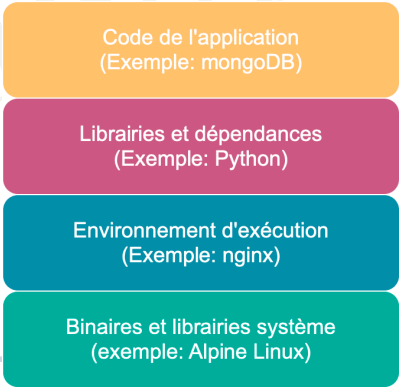
©Hainaut P. 2021 - www.coursonline.be

En pratique

Création et gestion d'une image

Docker: les images

- Une image est constituée d'une application et de toutes ses dépendances
- Elle est portable sur n'importe quel environnement où tourne Docker
- Elle peut être constituée de plusieurs layers
- Elle est distribuée via un registry
- Par défaut, les layers d'image(s) se trouvent dans: /var/lib/docker/overlay2/

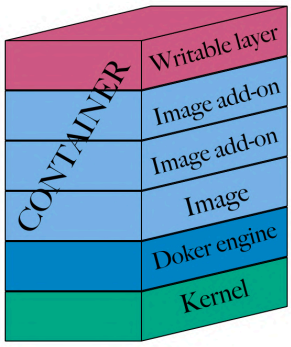


©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: les images

- Union file system:
  - Les layers sont en lecture seule
  - Un driver spécifique va réunir ces layers en un seul filesystem
  - Une layer en lecture/écriture sera ajoutée à l'image
  - Le mécanisme de copy-on-write permet alors de copier un fichier à modifier d'une layer en lecture seule vers la layer en lecture/écriture et d'apporter les modifications nécessaires qui persisteront dans cette layer



©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

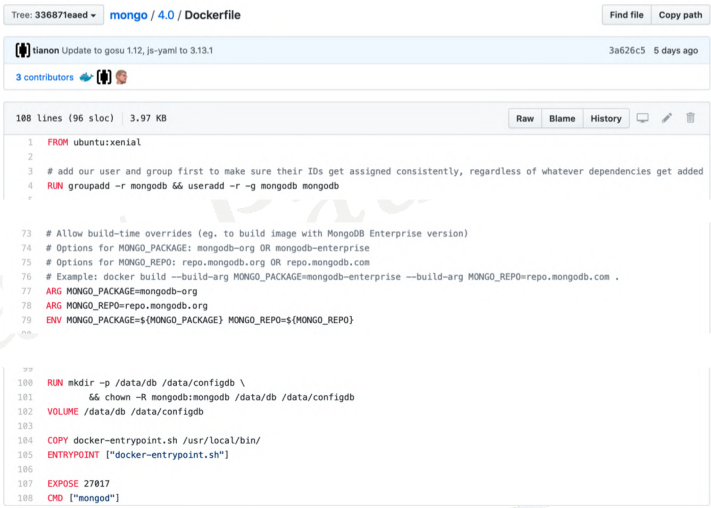
- C'est un fichier texte qui va contenir les instructions nécessaires pour construire le système de fichiers d'une image avec le code de l'application, ses dépendances, son environnement d'exécution, …
- On part généralement d'une image de base (exemple: Alpine Linux) qu'on va étoffer
- On génère l'image par la commande **docker image build**

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- Voici des extraits du Dockerfile de mongoDB 4.0



Tree: 336871eaed ▾ mongo / 4.0 / Dockerfile — Find file — Copy path

tianon Update to gosu 1.12, js-yaml to 3.13.1 — 3a626c5 — 5 days ago

3 contributors

108 lines (96 sloc) | 3.97 KB — Raw | Blame | History

```
FROM ubuntu:xenial

# add our user and group first to make sure their IDs get assigned consistently, regardless of whatever dependencies get added
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb

# Allow build-time overrides (eg. to build image with MongoDB Enterprise version)
# Options for MONGO_PACKAGE: mongodb-org OR mongodb-enterprise
# Options for MONGO_REPO: repo.mongodb.org OR repo.mongodb.com
# Example: docker build --build-arg MONGO_PACKAGE=mongodb-enterprise --build-arg MONGO_REPO=repo.mongodb.com .
ARG MONGO_PACKAGE=mongodb-org
ARG MONGO_REPO=repo.mongodb.org
ENV MONGO_PACKAGE=${MONGO_PACKAGE} MONGO_REPO=${MONGO_REPO}

RUN mkdir -p /data/db /data/configdb \
        && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 27017
CMD ["mongod"]
```

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- Voici les principales instructions qu'on peut rencontrer dans un Dockerfile

FROM	Image de base
COPY / ADD	copie des ressources locales dans le filesystem de l'image
RUN	exécute une commande dans le filesystem de l'image
ENV	spécifie une ou des variables d'environnement
USER	spécifie l'utilisateur qui a les droits pour lancer l'application
WORKDIR	spécifie le répertoire de travail de l'application
VOLUME	crée un volume localement pour la persistance des données
EXPOSE	expose un ou des ports bien déterminés de l'application
CMD / ENTRYPOINT	spécifie la commande à exécuter au lancement du container
HEALTHCHECK	vérifie la bonne santé de l'application

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- FROM permet de sélectionner une image de base
  - OS
  - OS + serveur applicatif
  - OS + environnement d'exécution
  - scratch -> une image vide reconnue par Docker qui permet de construire un image de base
- COPY/ADD permettent de copier des fichiers et répertoire de l'hôte dans l'image
  - Ça entraine la création d'une nouvelle layer de l'image
  - On peut spécifier des droits avec --chown (uniquement sous Linux)
  - COPY est préférable à ADD car l'instruction est mieux cadrée
  - On utilise ADD quand on a besoin de ressources à partir d'une URL ou de décompacter une archive

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- RUN permet d'exécuter une commande dans le système de fichiers de l'image
  - Engendre une nouvelle layer de l'image
  - 2 formats:
    - shell

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
```

ou

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

    - exec

```
RUN ["/bin/bash", "-c", "echo hello"]
```

- Le format shell exécute la commande dans le contexte d'un shell
- Avec la commande RUN, il faut faire attention aux problèmes de cache et dans le doute utiliser l'option --no-cache qui force le téléchargement des fichiers nécessaires

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- ENV permet la définition de variables d'environnement
  - qui servent pendant la construction de l'image et dans les containers lancés à partir de l'image

```
FROM busybox
ENV foo /bar
WORKDIR ${foo}   # WORKDIR /bar
ADD . $foo       # ADD . /bar
COPY \$foo /quux # COPY $foo /quux
```

- USER permet de définir l'utilisateur qui lancera l'application
  - Si il n'est pas précisé, c'est root qui lancera l'application
  - root dans le container = root sur la machine hôte
  - Influence les commandes RUN, CMD, ENTRYPOINT qui suivent dans le dockerfile
  - L'utilisateur peut être changé dynamiquement au démarrage du container (c'est le cas pour l'image mongo)

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- WORKDIR permet de définir un répertoire de travail qui par défaut sera un sous-répertoire de /var/lib/docker

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

- VOLUME permet de gérer les données en dehors du container (notion déjà vue)

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- EXPOSE spécifie les ports ouverts par l'application (notion déjà vue)
  - Ils peuvent être modifiés au lancement du container
    - par l'option -p port_du_container
    - par l'option -p port_de_l'hôte : port_du_container
    - par l'option –P où un port est choisi aléatoirement par Docker

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

- CMD/ENTRYPOINT spécifie la commande exécutée au lancement du container
  - ENTRYPOINT précise souvent le fichier de l'application à exécuter
  - CMD précise souvent les arguments
  - Comme pour RUN, on trouve les modes shell et exec

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile

- HEALTHCHECK permet de vérifier l'état de santé du processus
- Dans cet exemple, on vérifie, toutes les 5 minutes, que le serveur Web répond bien en local, et on définit un timeout de 3 secondes

```
HEALTHCHECK --interval=5m --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

- Cela permettra, par exemple à Swarm de redémarrer un container qui présente des défaillances

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: dockerfile

- Une fois le dockerfile constitué, on créera l'image via la commande:
  docker build [OPTIONS] PATH | URL | -

- Quelques options:
  - -f: précise le fichier à utiliser pour la construction de l'image (Dockerfile par défaut)
  - -t ou --tag: précise le nom et la version de l'image (user/repository:tag)
  - --label: permet d'ajouter des métadonnées à l'image

©Hainaut Patrick

Docker: le Dockerfile: exemple complet

- Voyons un exemple complet: un petit serveur Web basé sur node.js
  disponible ici: https://nodejs.org/fr/docs/guides/nodejs-docker-webapp/

- Ci-contre, le fichier principal server.js, qui définit
  le serveur Web basé sur la librairie express.js

- On voit que le serveur:
  - à besoin de la librairie express.js
  - écoute sur le port 8080
  - Renvoie comme message « Hello World »

```
'use strict';

const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

©Hainaut Patrick

Docker: le Dockerfile: exemple complet

- Ci-contre, le fichier package.json qui décrit l'application et ses dépendances

- On voit que:
  - le fichier utilisé est server.js
  - qu'il sera utilisé dans le script node server.js
  - qu'il dépend de la librairie express.js 4.16.1

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile: exemple complet

- Finalement, le fichier dockerfile:

- Au niveau des étapes:
  - on définit l'image de base
  - on définit le répertoire de travail
  - on copie le fichier .json dans le répertoire de travail
  - on installe les dépendances
  - on copie le code de l'application
  - on expose le port 8080
  - au lancement du container, on exécutera la commande node server.js

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile: exemple complet

- On peut créer l'image par la commande: docker image build -t app:1.0 .
  Le . final indique qu'on va chercher les fichiers locaux dans le répertoire courant

```
root@srv1804:~/webapp# docker image build -t app:1.0 .
Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM node:10
10: Pulling from library/node
99760bc62448: Pull complete
e3fa264a7a88: Pull complete
a222a2af289f: Pull complete
c1f89293f045: Pull complete
115b6fc5ace1: Pull complete
9eb516295c24: Pull complete
18f3759b33f3: Pull complete
627418129f22: Pull complete
1764e003d63c: Pull complete
Digest: sha256:e0c512c52c3ca8797fc60ada3698004f5cba13af4a8a5968041edac1b5d98577
Status: Downloaded newer image for node:10
 ---> a02c9f46f94a
Step 2/7 : WORKDIR /usr/src/app
 ---> Running in 44d178821393
Removing intermediate container 44d178821393
 ---> 8e8384ba9232
Step 3/7 : COPY package*.json ./
 ---> cf0325b477c3
Step 4/7 : RUN npm install
 ---> Running in cd78dba5fa86
```

```
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN docker_web_app@1.0.0 No repository field.
npm WARN docker_web_app@1.0.0 No license field.

added 50 packages from 37 contributors and audited 126 packages in 1.72s
found 0 vulnerabilities

Removing intermediate container cd78dba5fa86
 ---> 5a7c33bb1ae0
Step 5/7 : COPY . .
 ---> 9dc8ed7d54d5
Step 6/7 : EXPOSE 8080
 ---> Running in 3ef8bfe95038
Removing intermediate container 3ef8bfe95038
 ---> 2d171b07e7e9
Step 7/7 : CMD [ "node", "server.js" ]
 ---> Running in c773c5049215
Removing intermediate container c773c5049215
 ---> 9d07d9681dc1
Successfully built 9d07d9681dc1
Successfully tagged app:1.0
root@srv1804:~/webapp#
```

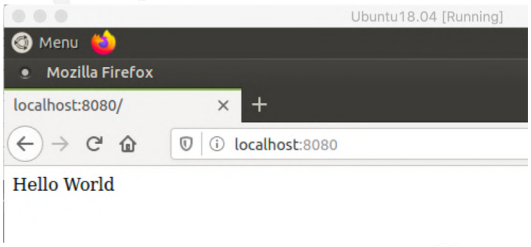
©Hainaut P. 2021 - www.coursonline.be

Docker: le Dockerfile: exemple complet

- On peut lancer un container basé sur cette image et vérifier l'exécution de l'application

```
root@srv1804:~/webapp# docker container run -p 8080:8080 -d app:1.0
96679307e75200fa8fe814d3225cf1a8799429e616c35dc5ee844fc8cead9c6e
root@srv1804:~/webapp#
```

- Comme on n'a pas utilisé l'option -v, le répertoire de travail /usr/src/app se trouve dans un sous-répertoire de /var/lib/docker/overlay2



©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Exercice 1

- Lancez un container basé sur Alpine Linux, en mode interactif, et en le nommant shell
- Lancez la commande curl google.be. Que constatez-vous ?
- Installez l'utilitaire curl et quittez le container avec CTRL-P CTRL-Q (pour ne pas killer le processus de PID 1)
- Créez une image, nommée alpine-curl, à partir du container shell
- Utilisez pour cela la commande commit (docker commit –help)
- Lancez un shell interactif dans un container basé sur l'image alpine-curl et vérifiez que curl est présent

©Hainaut P. 2021 - www.coursonline.be

Exercice 2

- Développez un serveur Web (réutilisez l'exemple complet et modifiez-le) qui a un sous-répertoire ping (accessible donc sur http://localhost/ping) accessible sur le port 80 et qui répond PONG
- Créez l'image
- Lancez un container basé sur cet image et vérifiez les résultats

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

En pratique

Les registries

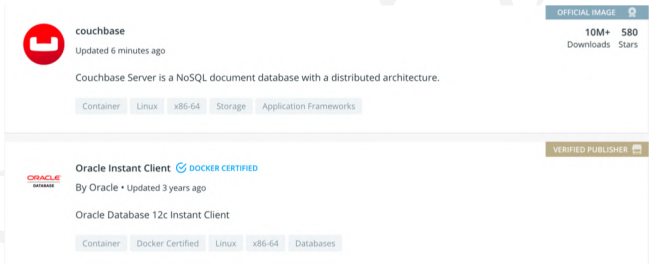
Docker: les registries

- Un registry, c'est une librairie d'images Docker
- On peut y downloader des images ou en uploader
- On connaît le Docker Hub, le registry par défaut Docker
- D'autres registry existent comme ceux d'Amazon, Google, GitLab, …

©Hainaut Patrick

Docker: le Docker Hub

- Accessible depuis: https://hub.docker.com
- On y trouve des images bien sur, mais aussi les versions de l'application et différents plugins qui augmentent les fonctionnalités de Docker



- Au niveau des images, il y a:
  - les images officielles publiées et certifiées par Docker
  - Les images fiables publiées par tiers de confiance, comme Oracle, certifiées également par Docker
  - Des images publiques dont on n'est pas sur de la fiabilité (vérifier les commentaires et la popularité)
  - Des images privées qui ne sont accessibles que si on a une autorisation

©Hainaut P. 2021 - www.coursonline.be

Docker: le Docker Hub

- Les images officielles sont scannées régulièrement à la recherche de failles de sécurité
- Ce n'est pas le cas des images non-certifiées
- Mais des outils fiables existent pour faire ce scan comme anchore ou harbor
- Il y a intégration avec des outils comme Github ou Bitbucket qui permettent une mise à jour automatique de l'image chaque fois le code est changé

©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Docker Hub

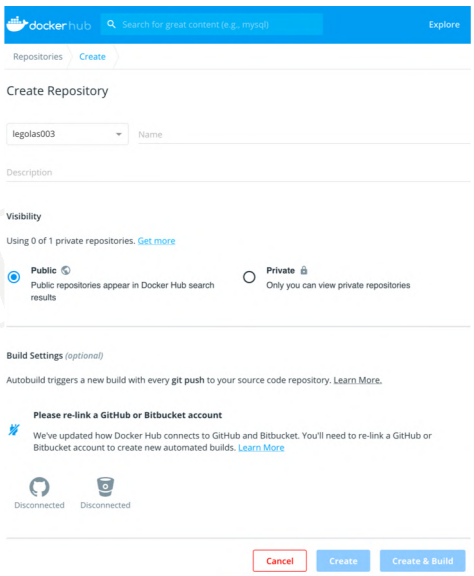
- Lorsqu'on clique sur une image dans le Docker Hub, on a accès à ses différentes versions, que l'on peut choisir suivant le tag
- Si on ne précise pas de tag lorsqu'on récupère l'image, il prend le dernière version par défaut (latest)
- On retrouve dans cette liste la version 4.0 que l'on avait utilisé dans l'exemple 7 en l'appelant mongo:4.0
- Ces différentes versions sont stockées dans un repository

son nom : *le tag*

©Hainaut P. 2021 - www.coursonline.be

Docker: le Docker Hub

- Quand on créée un compte sur le Docker Hub, on a accès à un repository personnel qu'on peut rendre public ou garder privé
- Si on en veut d'autres, il faut choisir un plan de payement
- Le repository va stocker toutes les versions de l'image avec comme nom:
  user/repository:version
  Exemple: hainautp/cloneos:1.0



©Hainaut P. 2021 - www.coursonline.be

©Hainaut P. 2021 - www.coursonline.be

Docker: le Docker Hub

- Pour donwloader une image, on utilise la commande **docker image pull** qu'on a déjà utilisé
- Pour uploader une image, on utilisera la commande **docker image push**

©Hainaut Patrick

Exercices

- Créez un compte sur le Docker Hub
- Créez le repository webserv avec la visibilité "public"
- Effectuez un login depuis la ligne commande en spécifiant vos identifiants Docker Hub
- Créez une image contenant un petit serveur web et taggez la avec username/webserv:1.0
- Uploadez l'image username/webserv:1.0 sur le Docker Hub
- Vérifiez que l'image est bien présente sur le docker hub et que vous pouvez l'utiliser

©Hainaut Patrick

Conclusion

- Docker est un système tentaculaire qui ne prend tout son intérêt que dans des process de développement qui sortent du cadre de ce cours
- Vous avez donc ici une introduction qui vous mettra l'eau à la bouche et vous facilitera le passage aux systèmes de container
- Dans la seconde partie, nous aborderons:
  - La gestion des ressources
  - La sécurité
  - Docker Swarm
  - Docker Machine

©Hainaut Patrick