Formation F2i

Prise en main de Docker

Intervenant: Hervé Meftah

17 et 18 décembre 2018

Historique de Docker

- Docker a été développé au début des années 2009 dans une maison de Montrouge par Solomon Hykes et 2 autres personnes passionnées par Linux.
- Première release en mars 2013.
- Initialement créé avec une base historique de librairies LXC.
- Docker est aujourd'hui développé en langage Go (Goland) de Google.

Qu’est ce que Docker?

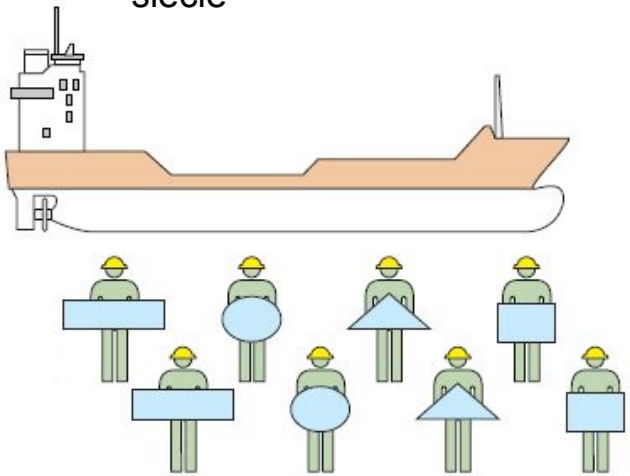
- Docker n’est pas un langage de programmation, c’est un ensemble d’outils pour construire des environnements d’exécution.
- C’est donc un ensemble d’outils pour vous aider à résoudre les problèmes d’installation, de retour-arrière, de distribution et de mise à jour de vos applications.
- Il est open source, c’est à dire que tout le monde peut contribuer à son développement.

Docker est une plate-forme qui permet de “*construire, transporter, et exécuter toutes applications, partout*”

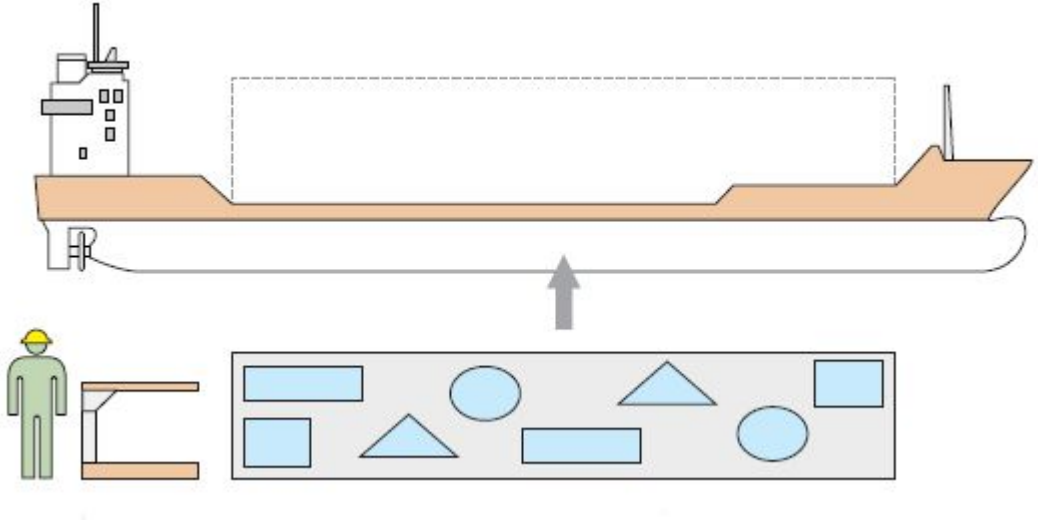
Il est utilisé pour pallier le problème le plus coûteux en informatique : le déploiement

Pourquoi le nom de Docker?

Les dockers dans un port au XIX siècle

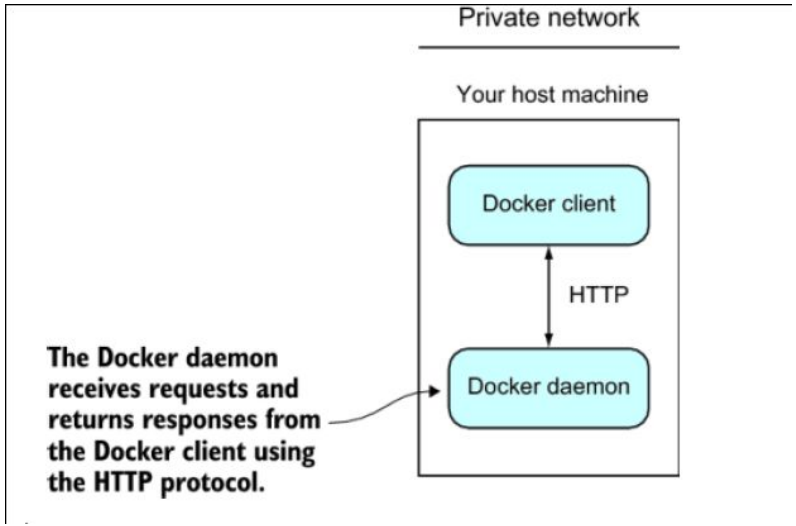


un docker dans un port aujourd'hui



Que contient Docker?

- Docker contient des applications qui fonctionnent en ligne de commande, un processus en tâche de fond (background daemon) et un ensemble de services distants.



Les avantages de Docker

- Replace les machines virtuels (VM).
- Permet de prototyper les applications.
- Permet le packaging d'applications.
- Ouverture vers les microservices.
- Modélisation d'un réseau informatique avec un budget réduit.
- Permet une certaine productivité mais avec des machines déconnectées.
- Réduire le temps de recherche des bugs.
- Renforce la documentation dès le début du cycle de vie d'une mise en production.
- Permet la mise en place du Continuous Delivery (CD).

Les inconvénients de Docker

- Fonctionne que sur des noyaux Linux récents, supérieur à la version 3.10. Faire un `uname -r` de votre système pour vérifier.
- Docker est rapide, mais pas aussi rapide que si vous utilisez directement votre machine physique.
- Pas encore complètement sécurisé. Donc pas encore prêt pour passer en production mais certaines sociétés le font déjà. (Red Hat, Google …).
- Pas de portabilité entre un container créé sous Windows ou sous Linux.
- Supporte difficilement des containers contenant une application avec une interface graphique complexe.
- Nécessite une remise en cause des équipes de sysadmin et nécessite également un certain temps d'apprentissage.

Les alternatives à Docker

- Le jail chroot de Linux

Jail Chroot est un moyen d'isoler un processus et ses children du reste du système. Il ne doit être utilisé que pour les processus qui ne s'exécutent pas en tant que root, car les utilisateurs root peuvent sortir du jail très facilement.

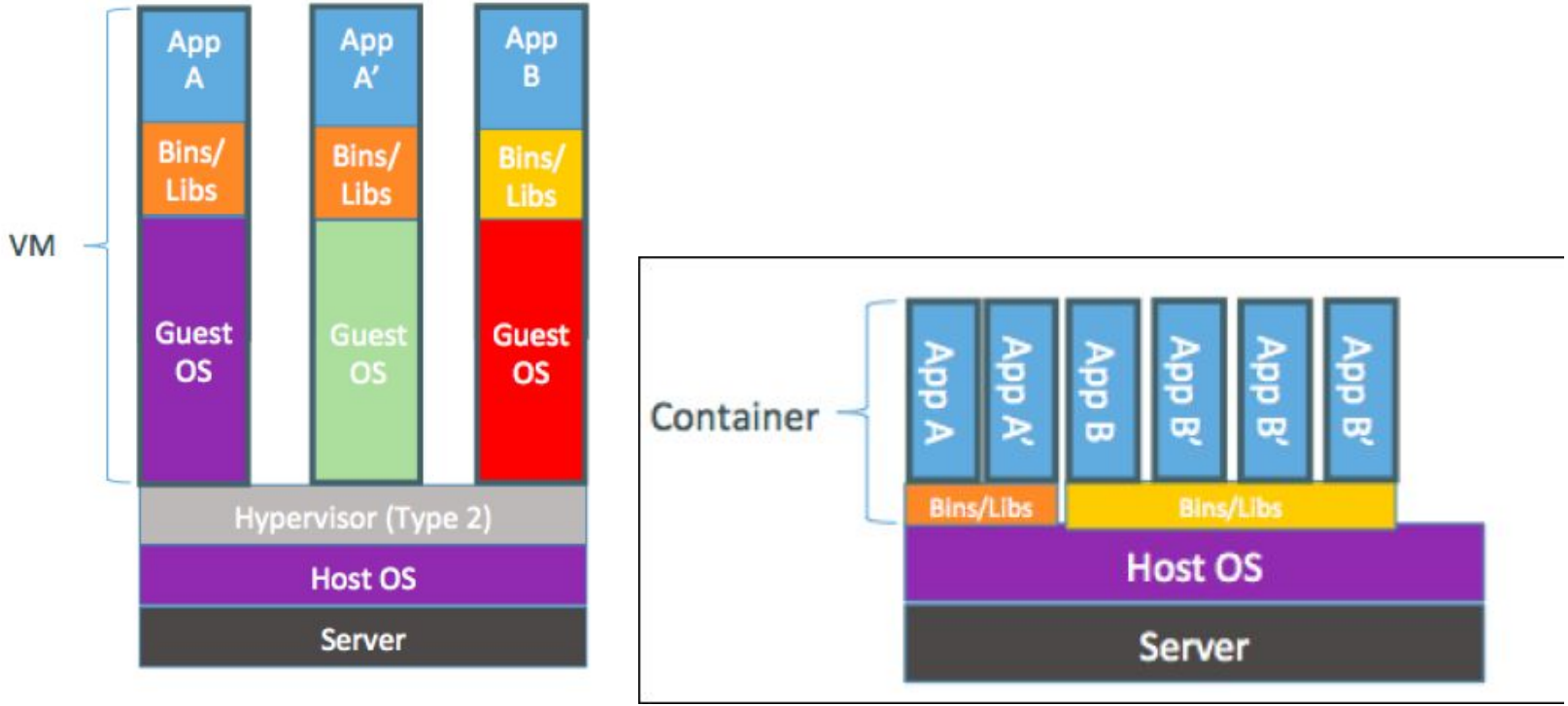
- LXC
  https://linuxcontainers.org/fr/

- OpenVZ
  https://openvz.org/

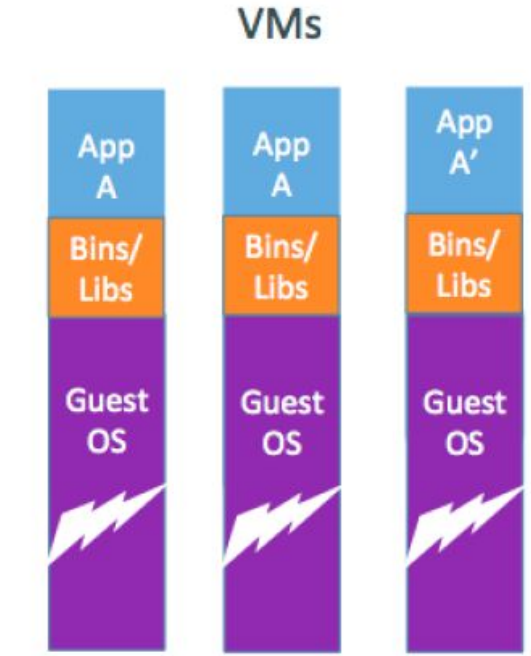
- Les Zones de solaris
  https://docs.oracle.com/cd/E19253-01/820-2318/zones.intro-1/index.html

- rkt (rocket) CoreOS
  https://coreos.com/rkt/

Comparaison entre VMs et Containers

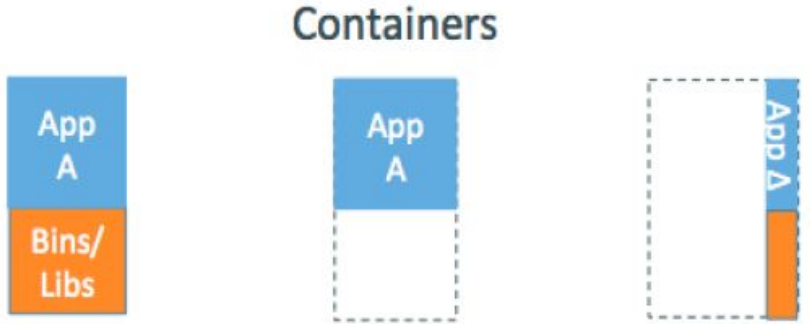


Copie des VM et copie de containers



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server



Original App
(No OS to take up space, resources, or require restart)

Copy of App
No OS. Can Share bins/libs

Modified App

Copy on write allows us to only save the diffs Between container A and container A'

Architecture de Docker

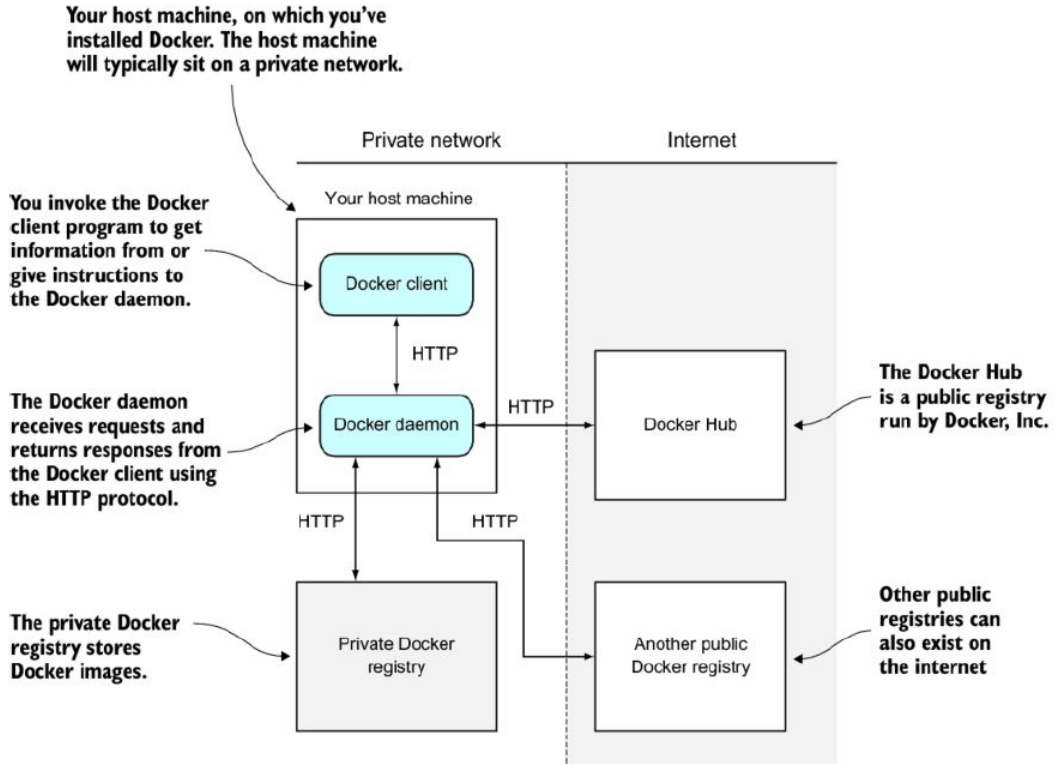
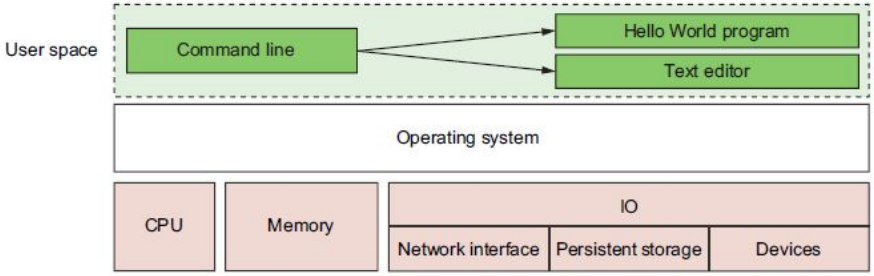
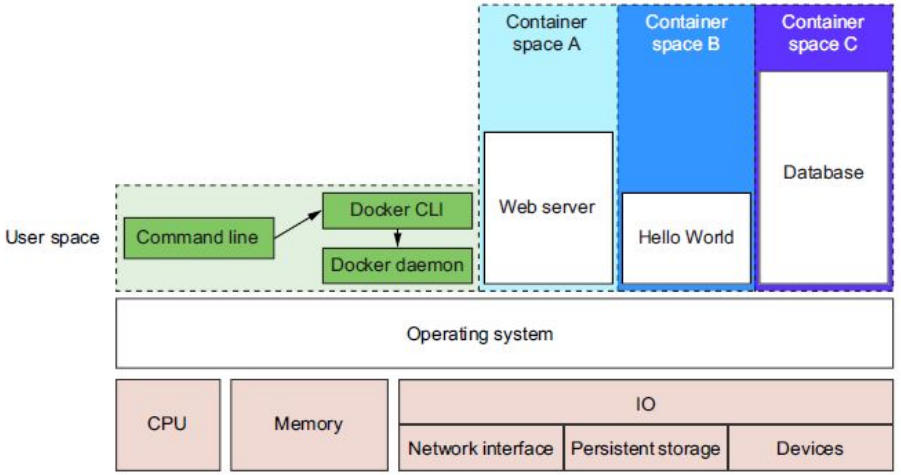


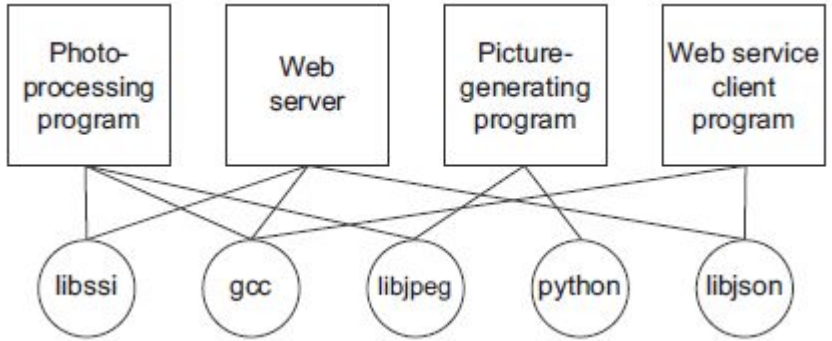
Schéma de la stack applicative



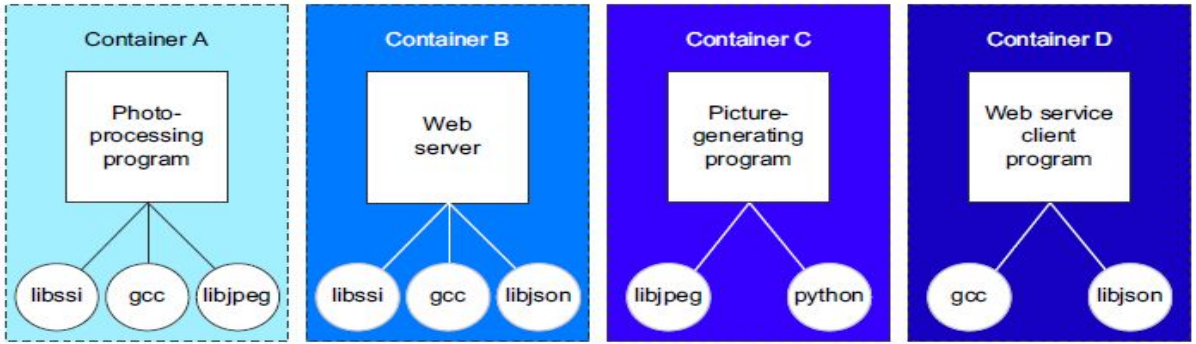


Sans ou Avec Docker (1)

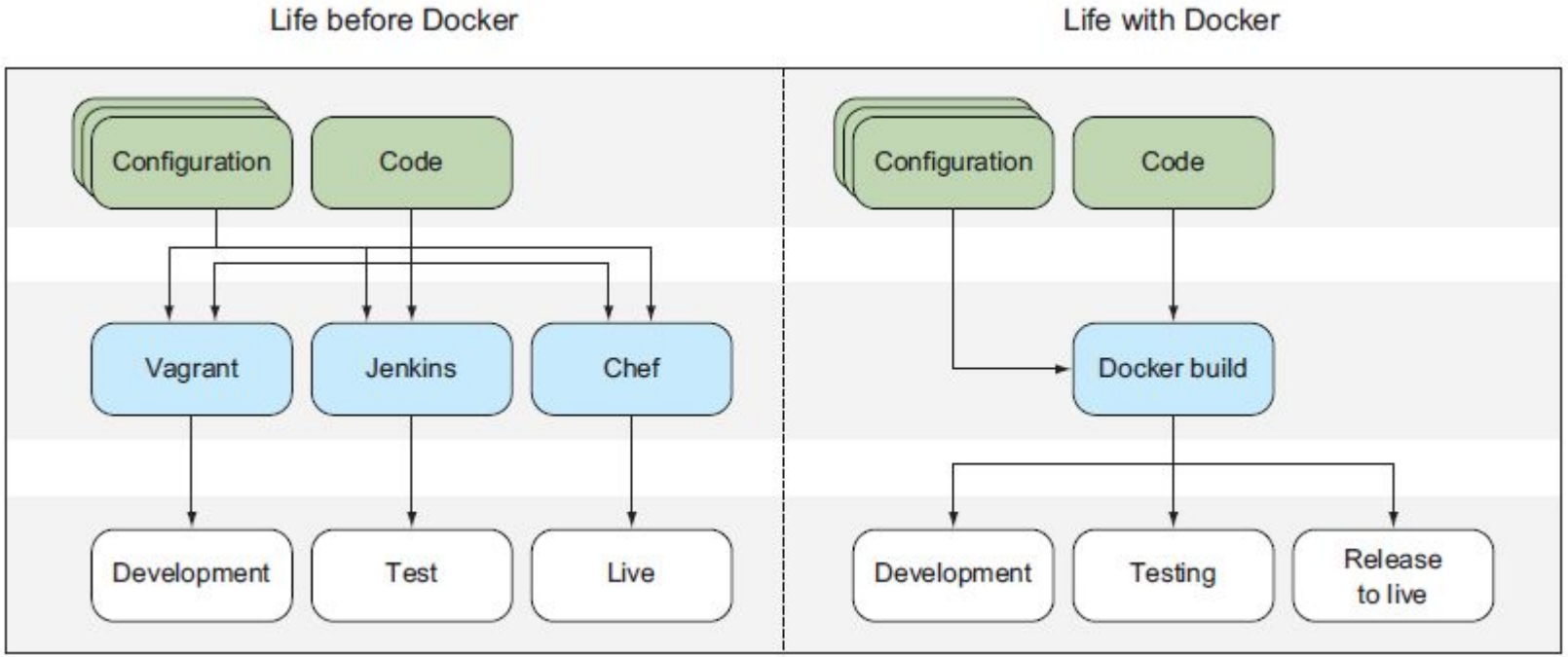
Sans Docker



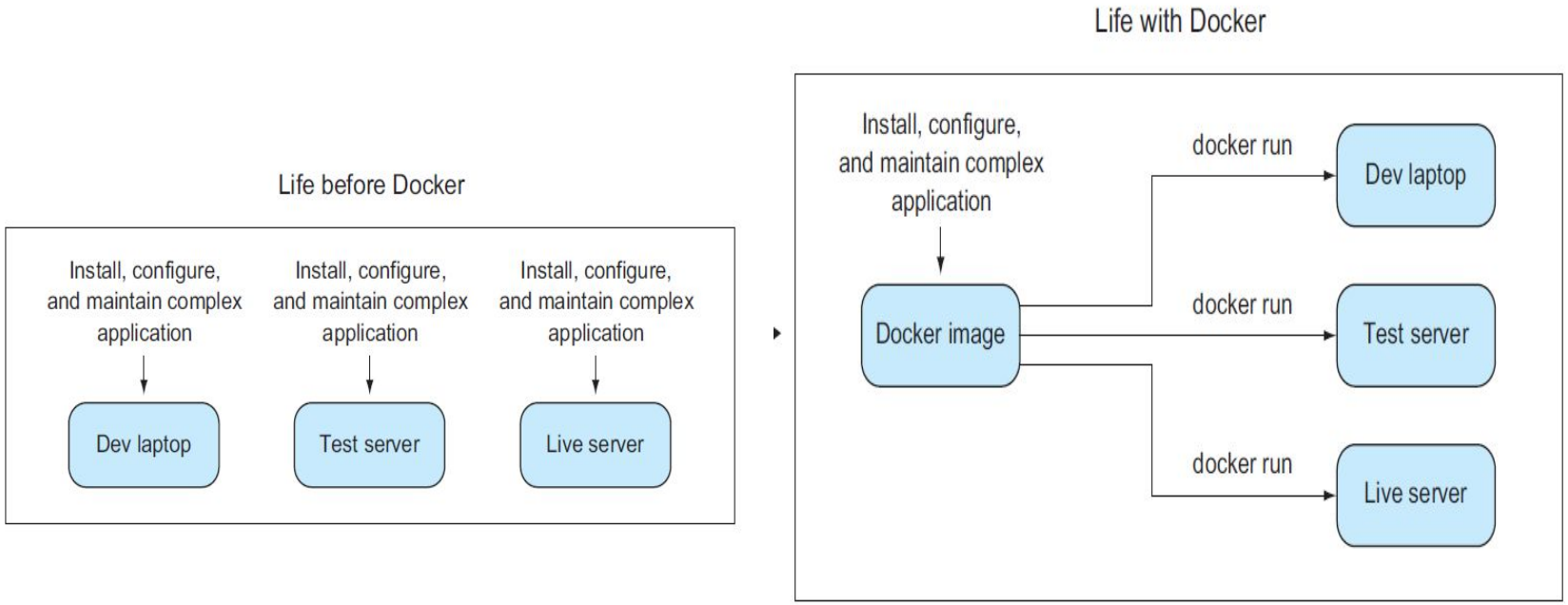
Avec Docker



Sans ou Avec Docker (2)



Sans ou Avec Docker (3)



La Philosophie de Docker

C’est la course à la version de Linux la plus légère et la plus efficace en terme de fonctionnalités.

Les images disponibles dans Docker Hub sont épurées des packages inutiles pour faire tourner vos applications et vos données.

C’est pour cette raison que des versions linux de type Alpine ou Tiny core existent pour vous permettre de faire des docker build, commit, export, save le plus rapidement possible.

Installer une application ou fonctionnalité par container, cela va nous permettre de changer l’architecture des applications monolithiques vers une architecture en microservices qui favorise l'agilité lors de la phase la plus coûteuse du cycle de vie des applicatifs: c’est-à-dire le **déploiement** .

De plus ….

Docker est un outil incroyable, peut-être avez-vous essayé de l'utiliser ou de le tester ou vous avez peut-être commencé à l'utiliser dans tout ou une partie de vos serveurs, mais la gestion et l'optimisation peuvent devenir très rapidement complexe.

Le fait que l'écosystème des conteneurs évolue rapidement est une contrainte pour la stabilité et c'est également une source de confusion.

Chez Google, tout fonctionne dans des conteneurs. Selon The Register, deux milliards de conteneurs sont lancés chaque semaine. Google utilise des conteneurs depuis une dizaine d'années, quand les technologies de conteneurisation n'étaient pas encore démocratisées;

C'est l'un des secrets de la performance et la régularité des opérations du moteur de recherche Google, GMail et autres services.

Installation de docker sous Ubuntu

```
# faire un update du repository des package ubuntu
sudo apt-get update
# installer les packages htop git et ansible
sudo apt-get -y install htop git ansible
# si docker est déjà installé on peut le retirer pour partir d'installation propre
sudo apt-get remove docker docker-engine docker.io
# installation des packages qui gèrent les certificats
sudo apt-get - y install apt-transport-https ca-certificates curl software-properties-common
# ajout du certificat de docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# ajout du repo docker
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
# mise à jour du repo
sudo apt-get update
# installation de Docker version Community Edition
sudo apt-get install docker-ce
# ajout de votre user au groupe docker
sudo usermod -aG docker hme
ou sudo gpasswd -a hme docker
Sortir de votre shell terminal pour que les changements prennent effet.
faire un docker ps
et un docker run hello-world pour tester l'installation
```

Installation de docker sous Fedora

```
sudo dnf -y install dnf-plugins-core

sudo dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo

sudo dnf -y install docker-ce

sudo systemctl start docker

sudo systemctl enable docker
sudo systemctl status docker
```

et crtl-C pour sortir

sudo usermod -aG docker <your_username>

log out and log in pour que cette commande prend effet.

voir https://docs.docker.com/engine/installation/linux/docker-ce/fedora/

Vérification de l'installation

hme@scw-f68b9f:~$ docker --version
Docker version 18.06.1-ce, build e68fc7a

Pour vérifier l'installation tapez:

```
docker run hello-world
```

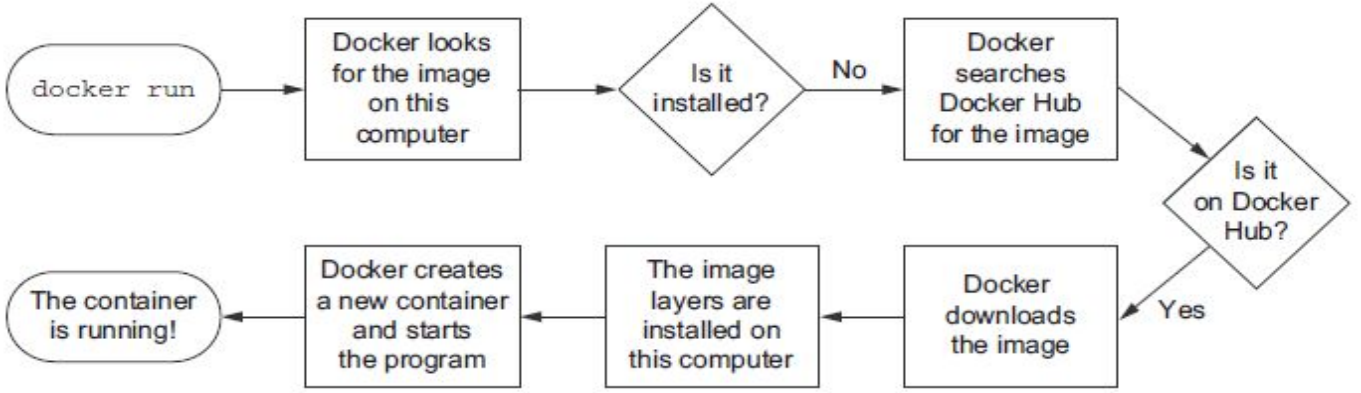
Le résultat est :

```
[fedora@ansible ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest:
sha256:be0cd392e45be79ffeffa6b05338b98ebb16c87b255f48e297ec7f98e123905c
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
```

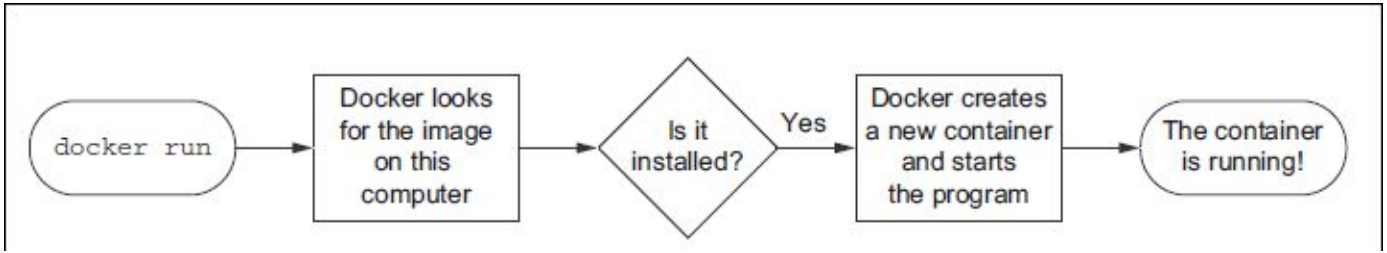
Docker run

La commande:

```
docker run hello-world
```



Lors d'un second Run



Storage driver dans Docker

hme@scw-f68b9f:~$ docker info

voir le type de storage driver

changer le storage drive de overlay2 vers aufs

Le stockage Overlay2 est utilisé comme pilote de stockage par défaut pour la version Community Edition de Docker, auparavant, c'était le stockage AUFS qui était utilisé.

Il est utile de changer le pilote de stockage et retourner à AUFS. Choisir le stockage Aufs évite des images et containers orphelins.

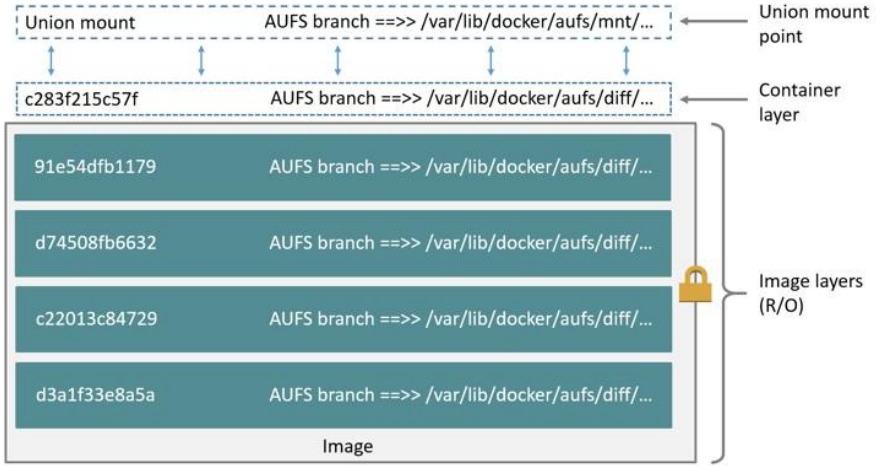
faire

sudo systemctl stop docker

```
sudo vi /etc/docker/daemon.json
et ajouter le code JSON suivant:

{
    "storage-driver": "aufs"
}
```

sudo systemctl start docker



Container et image

Une **image** est une collection ordonnée de modifications de layer Linux avec les paramètres d'exécution correspondants qui seront utilisés pour lancer un container.
Les images sont toujours Read-Only.

Un **container** est une instanciation dynamique d'une image active ou inactive si elle est terminée.

Pour des développeurs, la métaphore serait qu'une **image** est le programme exécutable, le **container** serait l'application qui s'exécute à l'écran et remplit ainsi un service.

Les commandes de base (1)

docker ps	liste les containers actifs
docker ps -a	liste tous les containers qui sont `created`, `restarting`, `running`, `removing`, `paused`, `exited` ou `dead`
docker ps -aq	liste tous les UUID des containers
docker stop <nom ou UUID>	arrête un container
docker rm <nom ou UUID>	détruit un container
docker pause <nom ou UUID>	suspend un container
docker start <nom ou UUID>	démarrer un container
docker images	liste toutes les images présentes sur le disque local
docker rmi <nom ou UUID>	détruit une image

Les commandes de base(2)

docker create	créer un container
docker build	créer une image à partir d'un fichier Dockerfile
docker import	importer un fichier tar dans un image
docker detach	detach le process du container
docker exec	Fait exécuter une commande par un container
docker save	Crée une image à partir d'un container
docker export	Crée un fichier tar d'une image
docker commit	Crée une image de l'état courant d'un container

Les commandes de base(3)

docker inspect	Retourne des information de bas niveau sur les objets docker; images, containers, network
docker load	charge dans une image un fichier tar préalable exporté avec docker export.
docker kill	kill le container
docker info	affiche l'état de Docker
docker search <nom>	rechercher dans le localhost et docker hub les images disponibles
docker logs <nom du container>	Affiche message log du container
docker pull <image_name>	importe une image d'un repository local ou de docker hub
docker pull -a <image_name>	exporte une image vers un repository

La commande docker ps

La commande `docker ps et docker ps -a` listera les détails suivants:

- ID CONTAINER: Indique l'ID du conteneur associé au conteneur.
  L'ID du conteneur est un nombre aléatoire à 64 chiffres hexadécimaux. Par défaut, le
  La sous-commande docker ps affichera seulement 12 chiffres hexadécimaux. Vous pouvez afficher tous les 64 chiffres en utilisant le drapeau --no-trunc (par exemple: sudo docker ps --no-trunc).
- IMAGE: affiche l'image à partir de laquelle le conteneur Docker a été fabriqué
- COMMAND: Ceci vous montre la commande exécutée pendant le lancement du container
- CREATED: cela vous indique quand le conteneur a été créé.
- STATUS: Ceci vous indique l'état actuel du conteneur.
- PORTS: cela vous indique si un port a été affecté au conteneur.
- NAMES: le moteur Docker génère automatiquement un nom de conteneur aléatoire en concaténant un adjectif et un nom d'une personne connue.

Soit l'identifiant du conteneur ou son nom peut être utilisé pour faire d'autre actions sur le conteneur. Le nom du conteneur peut être configuré manuellement en utilisant l'option --name dans l'exécution du docker sous-commande

Un container en mode interactif ou en tâche de fond

```
docker run -it --name mycontainer centos /bin/bash
```

[root@386ec090afdc /]# hostname
386ec090afdc
[root@386ec090afdc /]# exit
stop le container

```
docker start  mycontainer
```

docker attach mycontainer

```
[root@386ec090afdc /]#
```

ensuite pour le mettre en background
faire Crtl-p Crtl-q
et

```
docker ps
```

Pour lancer un container en tâche de fond en ligne de commande faire

```
docker run -it -d --name mycontainer centos /bin/bash
```

Docker daemon, pause et unpause

```
docker run -d --name mytest ubuntu /bin/bash -c "while true; do date
; sleep 5; done"
```

faire les commandes suivantes et vérifier chaques résultats à chaque étape

```
docker ps
docker logs mytest
docker pause mytest
docker logs mytest
docker unpause mytest
docker logs mytest
```

Commandes utiles

```
# pour arrêter tous les containers qui tournent sur votre machine
docker stop $(docker ps -aq)
# pour détruire tous vos containers
docker rm $(docker ps -aq)
# pour détruire toutes les images
docker rmi  $(docker images -q)
```

Voir les changements à l'intérieur d'un container

Par rapport à l'image sur le disque

```
[root@9e02ac8c48be /]#
      ls -ld
      touch {abc,def,ghi}
      ls
      ls -ld
      ls -alrt
[fedora@ansible ~]$ docker diff 9e02ac8c48be
A /ghi
A /abc
A /def
```

C dénote un changement , A dénote un ajout , D dénote un retrait

la commande diff répertorie les fichiers et répertoires modifiés dans le système de fichiers d'un conteneur depuis sa création.

Les 4 manières de créer une image Docker

`docker commit:` crée une image à partir d'un container

```
docker commit c3f279d17e0a  test:version3
```

`docker load` : command docker load --input < fichier tar créé par un docker export>

docker load --input fedora.tar

commandes intermediaires qui creent un fichier de type archive (tar)

```
docket save : docker pull busybox:latest
              docker save -o myfile.tar busybox:latest
```

docker export: command docker pour exporter un container

```
docker export red_panda > latest.tar
```

A Noter: Attention à la différence entre les commandes SAVE et EXPORT

`docker import` : crée une image en important un fichier tar d'une image KVM qcow2 par exemple

```
cat zorin.tgz | sudo docker import - zorin:init
```

…….et grâce à un fichier Dockerfile

Le Dockerfile

Docker peut créer des images automatiquement en lisant les instructions contenues dans fichier nommé par défaut **Dockerfile**.

Un Dockerfile est un fichier texte contenant toutes les commandes qu'un utilisateur peut appeler pour assembler une image.

L'utilisation de la construction d’image avec un Dockerfile peut créer une forme automatisée de création d’environnement de tests et/ou de pré-production.

Exemple de Dockerfile

```
FROM  centos:latest

LABEL maintainer="Herve Meftah dockerlite@gmail.com"

# install package and monitoring tools
RUN   yum -y update && \
      yum -y install epel-release && \
      yum -y install wget unzip git htop iotop iftop
```

Voir un exemple dans docker hub, l'image zabbix

Dockerfile

```
FROM ubuntu:trusty
LABEL maintainer "Alexey Pustovalov <alexey.pustovalov@zabbix.com>"

ARG APT_FLAGS_COMMON="-qq -y"
ARG APT_FLAGS_PERSISTANT="${APT_FLAGS_COMMON} --no-install-recommends"
ARG APT_FLAGS_DEV="${APT_FLAGS_COMMON} --no-install-recommends"
ARG DB_TYPE=mysql
ENV LANG=en_US.UTF-8 LANGUAGE=en_US:en LC_ALL=en_US.UTF-8 DEBIAN_FRONTEND=noninte
ENV MIBDIRS=/usr/share/mibs/iana:/usr/share/mibs/ietf:/usr/share/snmp/mibs:/var/l

RUN locale-gen $LC_ALL && \
    echo "#!/bin/sh\nexit 0" > /usr/sbin/policy-rc.d && \
    DISTRIB_CODENAME=$(/bin/bash -c 'source /etc/lsb-release && echo $DISTRIB_COD
    echo "deb http://us.archive.ubuntu.com/ubuntu/ $DISTRIB_CODENAME multiverse"
    addgroup --system --quiet zabbix && \
    adduser --quiet \
```

Les mots-clé d'un Dockerfile

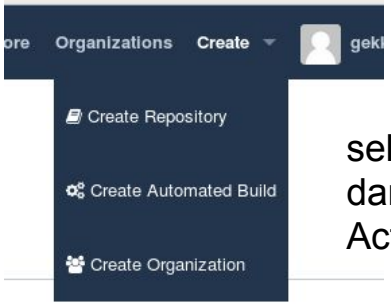
FROM , LABEL, MAINTAINER, RUN , COPY, ADD, EXPOSE, VOLUME, ENTRYPOINT, CMD, ENV , WORKDIR……

Différences entre COPY et ADD

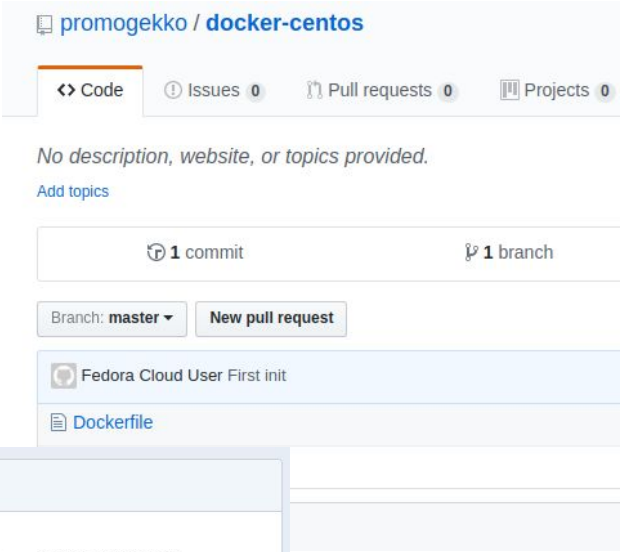
Différences entre ENTRYPOINT et CMD

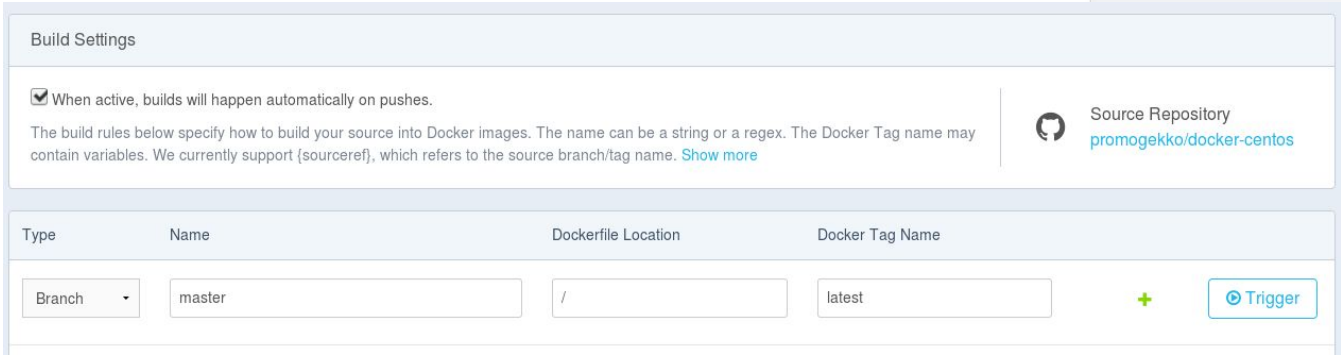
Publier vos images avec Docker Hub

Créer un repo github contenant votre dockerfile



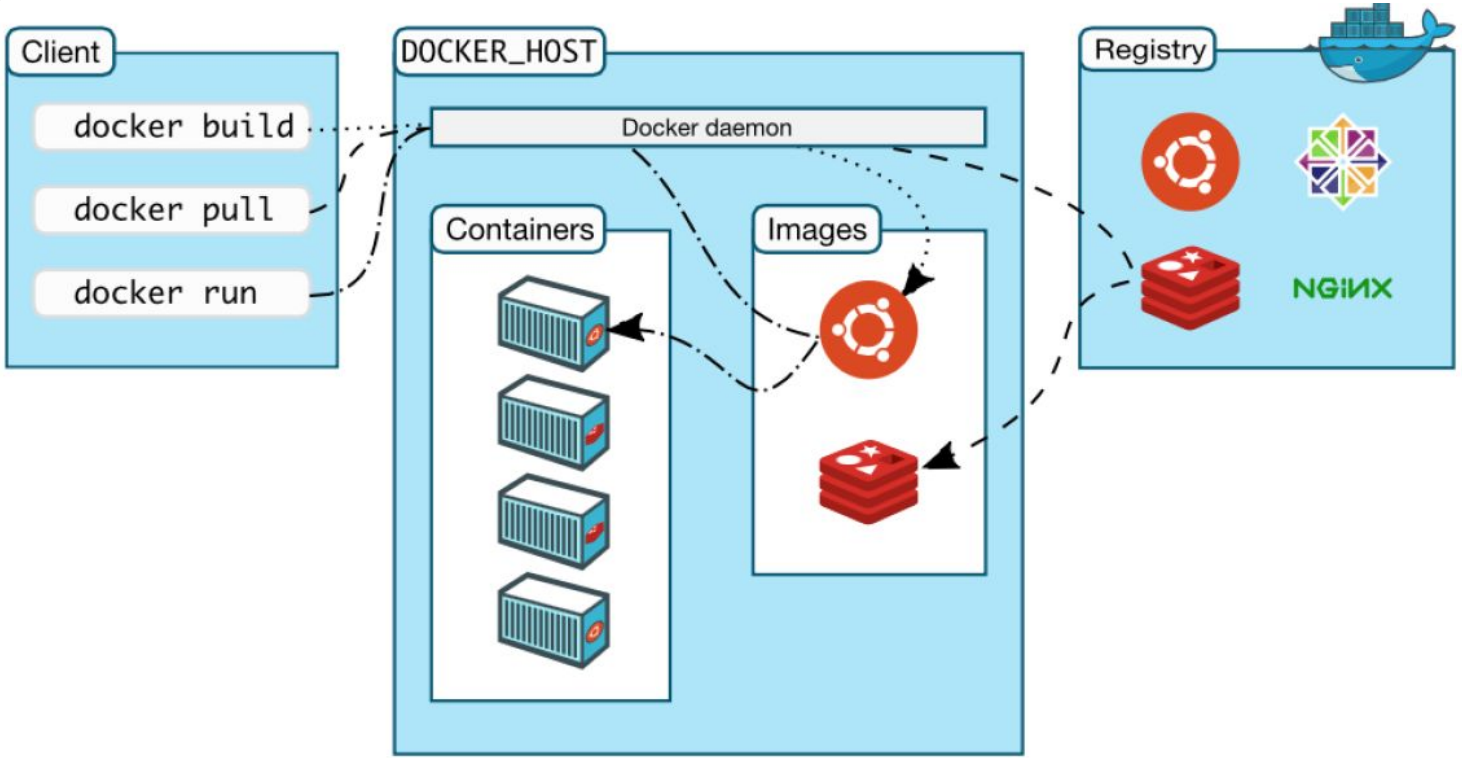
selectionner create automated build dans Docker Hub
Activer le link vers votre repo GitHub





Lancer le trigger manuellement la première fois

Docker Registry



Comment créer son repository d'images

installer le container Docker Registry

```
docker run -d -p 5000:5000 --name registry registry:2
```

download une image de docker hub

```
docker pull ubuntu
```

tagger cette image avec l'adresse IP et le port d'écoute

```
docker image tag ubuntu localhost:5000/myfirstimage
```

check

```
docker images
```

save l'image dans Docker Registry

```
docker push localhost:5000/myfirstimage
```

Attention à la définition d'un tag; le tag latest peut ne pas être la dernière version.

- Docker Hub
- Quay
- Google Container Registry
- AWS Container Registry

Tutum était un éditeur d'images Docker qui a été racheté par Docker Inc.

Voir les commandes qui ont créés une image

```
docker history web
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
1286af07562b       26 minutes ago      /bin/sh -c #(nop)  CMD ["/bin/bash"]          0B
268e74a2dcf5       26 minutes ago      /bin/sh -c apt-get -y update  &&      apt-...   257MB
20c44cd7596f       2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]          0B
<missing>          2 weeks ago         /bin/sh -c mkdir -p /run/systemd && echo '...   7B
<missing>          2 weeks ago         /bin/sh -c sed -i 's/^#\s*\(deb.*universe\...   2.76kB
<missing>          2 weeks ago         /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>          2 weeks ago         /bin/sh -c set -xe   && echo '#!/bin/sh' >...   745B
<missing>          2 weeks ago         /bin/sh -c #(nop) ADD file:280a445783f309c...   123MB
```

Attention à ne pas mettre de mot passe dans votre script Dockerfile ou d'autres informations de sécurité.

Créer une image Docker à partir d'un fichier qcow2

faire un
wget https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud-1808.qcow2

Installer les librairies KVM on fedora

```
sudo yum install kvm  virt-manager libvirt libvirt-python
appliance-tools libguestfs-tools
```

Installer les librairies KVM on ubuntu

```
sudo apt-get install libguestfs-tools
```

convertir le fichier qcow2 vers un fichier tar

```
sudo virt-tar-out -a CentOS-7-x86_64-GenericCloud-1808.qcow2 / - | gzip
--best > centos.tgz
```

conversion du fichier tar en image Docker

```
cat centos.tgz | sudo docker import - centos:base
```

Créer une image Docker à partir d'un fichier vdi

convertir le fichier vdi (VirtualBox) vers un fichier img

```
 VBoxManage  clonehd --format RAW /home/hme/VirtualBox\
VMs/alpine/alpine.vdi  vm.img
```

conversion du fichier img vers en fichier qcow2

```
qemu-img convert -f raw vm.img -O qcow2 vm.qcow2
```

conversion du fichier qcow2 vers un fichier tar

```
virt-tar-out -a vm.qcow2 / - | gzip --best > alpine.tgz
```

import du fichier tar vers une image Docker

```
cat alpine.tgz | sudo docker import - alpine:vdi
```

Création et versioning d'image

```
cat alpine.tgz | docker import - alpine:base
docker images
docker run -it --name alp alpine:base /bin/ash
(exécute un apk update et apk upgrade )
docker ps
docker ps -a
docker commit 75ca9da5c9e4 alpine:3.6
( c'est l'image avec les packages les plus récents )

docker images
docker ps -a
docker start 75ca9da5c9e4

docker ps
docker attach 75ca9da5c9e4
```

Exécuter un service dans un container (1)

```
# Faire un fork de
  ambient-docker/docker-apache2
# installer un build automatique avec Docker Hub
# Créer un container en tâche de fond a partir de l'image créée dans docker Hub
docker run -d <votre_repo>/docker-apache2
# Faire un
docker logs bc37bd357a0d
# retrouver l'adresse IP du container
docker inspect --format='{{.NetworkSettings.IPAddress}}' bc37bd357a0d
# faire
wget -qO - <ip_address>
```

Exposer les ports d'un container

Pour exposer un service fonctionnant dans un container, 3 solutions sont possibles :

- en ajoutant -p <port> dans la commande docker run
- ajouter un mot-clé EXPOSE <port> dans le Dockerfile
- ou mettre -P dans la commande docker run pour exposer tous les ports du container

Ajouter EXPOSE 80 dans le Dockerfile pour exposer le port 80 du container
Lancer une premier fois le container avec docker run -d -p 80 apache2
et faire un docker ps

faire un docker stop <container_id> et un docker rm <container_id>

et refaire docker run -d -p 80:80 apache2

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                   NAMES
d1eae14cecf4        apache2             "/usr/sbin/apache2..."   5 seconds ago       Up 4 seconds
0.0.0.0:80->80/tcp      tender_wozniak
48d9845d971f        apache2             "/usr/sbin/apache2..."   3 minutes ago       Up 3 minutes
0.0.0.0:32768->80/tcp   wizardly_darwin
```

Partager des données avec un container (1)

Data Volume
par le mot-clé VOLUME dans le Dockerfile
ou en passant le flag -v /MountPoint dans une commande docker run
Exemple:
faire un docker pull gekkopromo/docker-volume
docker inspect gekkopromo/docker-volume

docker run -it --name test gekkopromo/docker-volume
root@c185d4fccb3e:/# ls -ld /MountPointDemo/
drwxr-xr-x. 2 root root 4096 Dec 2 13:11 /MountPointDemo/
Faire un docker inspect du container


```
"Mounts": [
        {
            "Type": "volume",
            "Name": "337b3b4c884f2be635a61774cd90eca1a24a994e78e310ab86878d23d8c10950",
            "Source":
"/var/lib/docker/volumes/337b3b4c884f2be635a61774cd90eca1a24a994e78e310ab86878d23d8c10950/_data",
            "Destination": "/MountPointDemo",
            "Driver": "local",
            "Mode": "",
            "RW": true,
            "Propagation": ""
```

Partager des données avec un container (2)

```
docker run -v /MountPointDemo -it ubuntu:latest
docker ps
docker rm -v 68cfbe097a8f
docker run  --rm -v /MountPointDemo -it ubuntu:latest
```

Il y a 3 types de partage de volume

1. -v <container mount path>
2. -v <host path>:/<container mount path>
3. -v <host path>:/<container mount path>:<read write mode>

```
docker run -v /tmp/hostdir:/MountPoint -it ubuntu
root@04f88adfe408:/MountPoint# touch a b c
[fedora@ansible tmp]$ cd hostdir/
[root@ansible hostdir]# touch t
```

Note: Pour trouver les volumes orphelins faire

```
docker volume ls -qf dangling=true
```

et pour les detruire faire

```
docker volume rm `docker volume ls -qf dangling=true`
```

Partager des données avec un container (3)

```
docker run -it -d  -p 80:80 -p 443:443 -p 2222:22 -v /opt/gitlab/config:/etc/gitlab -v
/opt/gitlab/logs:/var/log/gitlab -v /opt/gitlab/data:/var/opt/gitlab --name gitlab
gitlab/gitlab-ce:8.14.4-ce.0
```

Vérifier que les fichiers log sont présents dans la directory /opt/gitlab/logs

Egalement on peut constater que les données sont présentes sur le localhost

Comment démarrer plusieurs services

- Un container n'est pas une VM
- Car il n'y a pas d'INIT ou SYSTEMD process.
- Vous ne pouvez démarrer qu'un processus par container

Une solution existe mais elle n'est pas conseillée, il faut, en effet, rester dans la philosophie de Docker.

Faire un fork de

https://github.com/ambient-docker/supervised.git

Partager des données entre les containers

Nous avons appris comment le moteur Docker permet de partager de données entre le localhost et les conteneurs. C'est une solution efficace, elle couple étroitement le conteneur au système de fichiers hôte. Parfois les répertoires peuvent être laisses par l'utilisateur qui doit ensuite les supprimer manuellement. Pour résoudre ce problème, c'est de créer des conteneurs de données uniquement en tant que conteneur de base, puis montez le volume de données de ce conteneur à d'autres conteneurs en utilisant l'option **--volume-from** de Docker.

```
docker run --name datavol -v /DataMount busybox:latest /bin/true
docker run -it --volumes-from datavol ubuntu /bin/bash
```

Only-Data container

Dans le passé, l'approche la plus recommandée consistait à utiliser un «Only-Data-container» pour le stockage de persistance, par exemple. une base de données dans docker. Le conteneur de données ne fait rien d'autre que d'exposer un «volume de données». Par exemple. une fois que vous avez exécuté le «conteneur de données», vous pouvez le lier à un autre conteneur comme ceci:

```
docker run --volumes-from data-only-container other-container my-cmd
```

Mais depuis la version de Docker 1.9.0, il y a une nouvelle api pour le volume et maintenant le modèle plus ancien de «conteneur réservé aux données» a été abandonné au profit de ces nouveaux volumes. Cette nouvelle API des volumes est un meilleur moyen d'atteindre les résultats finaux du «conteneur de données».

```
docker volume create --name helloworld
docker run -d -v helloworld:/container/path/for/mapped/volume image-name command-to-execute
```

Cas particulier Docker-In-Docker

Nous pouvons utiliser le daemon Docker de votre machine hôte à l'intérieur d'un container pour pouvoir lancer des commandes Docker et faire ainsi des containers dans vos containers.
Cette technique est très utile pour mettre en place une plateforme d'intégration continue.

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock ubuntu:latest sh
-c "apt-get update ; apt-get install docker.io -y ; bash"
```

Par un dockerfile

```
FROM  ubuntu:latest
RUN  apt-get -y update  && \
     apt-get install docker.io -y
CMD ["/bin/bash"]
```

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock  docker-in-docker
```

La liaison de containers (1)

L'une des principales caractéristiques de la technologie Docker est la liaison entre conteneurs. C'est-à-dire que les conteneurs coopérants peuvent être reliés entre eux pour offrir des services. Les conteneurs liés ont une sorte relation source vers destinataire dans lequel le conteneur source est lié au conteneur destinataire. Le destinataire reçoit en toute sécurité une variété d'informations de la source. Cependant, le conteneur source ne sait rien des destinataires auquel il est lié. Une autre caractéristique notable de la liaison des conteneurs, dans un configuration sécurisée, est que les conteneurs liés peuvent communiquer en utilisant des tunnels sécurisés sans exposer les ports utilisés.

Le moteur Docker fournit l'option --link dans la sous-commande de docker run à lier d'un conteneur source vers un conteneur de destination.

Le format de l'option --link est le suivant:

--link <conteneur>: <alias>

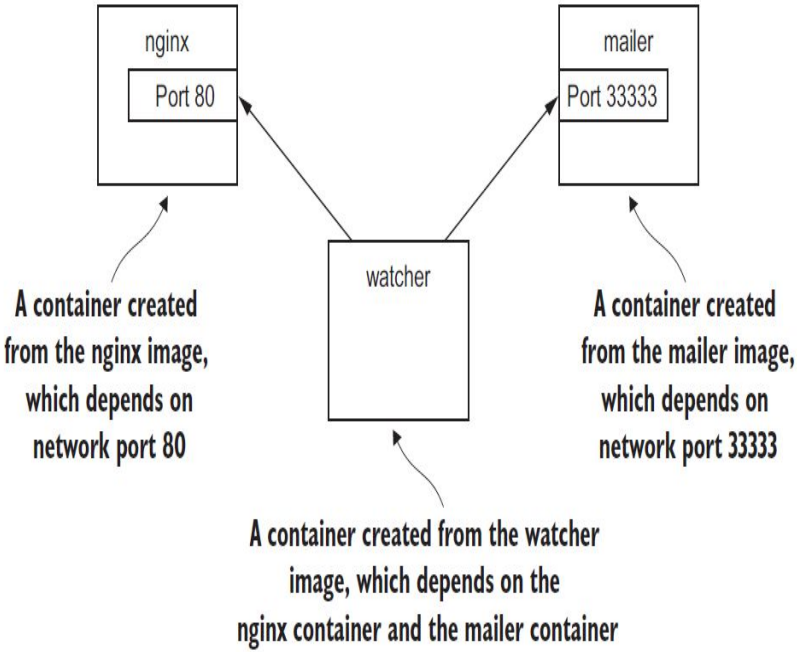
La liaison de containers (2)

```
 docker run --rm --name example -it
busybox:latest
/ # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
172.17.0.3      5fd1686c77c1
/ # env
HOSTNAME=5fd1686c77c1
SHLVL=1
HOME=/root
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n
PWD=/
```

```
docker run --rm --link example:ex -it
busybox:latest
/ # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
172.17.0.3      ex 5fd1686c77c1 example
172.17.0.4      554a057fb13a
/ # env
HOSTNAME=554a057fb13a
SHLVL=1
HOME=/root
EX_NAME=/sad_poincare/ex
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
```

Travaux Pratiques: Monitorer un site web

Supposons qu'un nouveau client entre dans votre bureau et vous fait une offre de leur construire un nouveau site web.
Ils veulent un site Web étroitement surveillé en envoyant un e-mail à leur équipe lorsque le serveur est arrêté.
Ils ont également entendu parler de ce logiciel serveur appelé NGINX et ont spécifiquement demandé que vous l'utilisez.
Ayant compris les mérites de travailler avec Docker, vous avez décidé de l'intégrer pour ce projet.



La solution

```
docker run -d --name web nginx:latest

docker run -d  --name mailer gekkopromo/mailer

docker run -it --link web:web  --name web_test  busybox:latest /bin/sh
```

cat /etc/hosts
env
wget -O - http://web:80

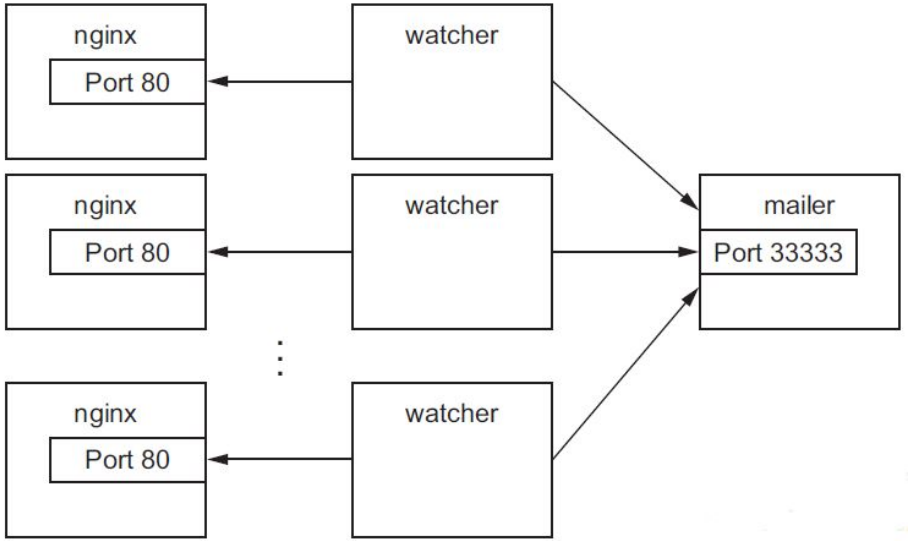
```
docker run -it --name agent --link web:insideweb --link mailer:insidemailer gekkopromo/agent
```

dans un autre terminal
faire
docker ps
docker logs web
docker logs mailer
docker logs agent

docker stop web
docker logs mailer

Travaux Pratiques: Monitorer plusieurs sites web

Considérons un autre exemple où un client vous a demandé de construire un système où vous pouvez héberger un nombre variable de sites Web. Il aimerait aussi utilisez la même technologie de surveillance que vous avez développée précédemment.L'extension du système que vous avez construit plus tôt serait le moyen le plus simple de faire ce travail sans personnaliser la configuration de NGINX.



La solution (partie 1)

La solution est de démarrer le plus de containers que vous souhaitez, mais ce n'est pas aussi simple que ca, identifier les containers est plus compliqué quand leur nombre augmente.

Faire

```
docker run -d --name webid nginx
docker run -d --name webid nginx
```

Vous donne une erreur de conflit entre les noms de container, vous pouvez changer le nom du container en faisant

```
docker rename webid webid-old
docker run -d --name webid nginx
```

Mais cette opération n'est pas efficace si vous avez une *ferme* de containers d'une plusieurs centaines.
Docker assigne un unique identifiant en hexadécimal de 128-bits , avec cet identifiant vous etes sure de ne pas avoir de conflit. Donc vous pouvez assigner cet identifiant a une variable Linux en faisant :

```
CID=$(docker create nginx:latest)
echo $CID
```

Cette commande va créer le container mais pas le démarrer, il est donc inactif.

La solution (partie 2)

Démarrer votre mailer en premier

```
MAILER_CID=$(docker run -d gekkopromo/mailer)
```

et ensuite créer les autres containers sans les démarrer

```
WEB_CID=$(docker create nginx)
AGENT_CID=$(docker create --link $WEB_CID:insideweb --link \
$MAILER_CID:insidemailer gekkopromo/agent)
```

et il faut ensuite les démarrer dans le bon ordre

```
docker start $WEB_CID
docker start $AGENT_CID
```

Le script le plus concis est de faire :

```
MAILER_CID=$(docker run -d gekkopromo/mailer)
WEB_CID=$(docker run -d nginx)
AGENT_CID=$(docker run -d \
--link $WEB_CID:insideweb \
--link $MAILER_CID:insidemailer \
gekkopromo/agent)
```

TP: Volume et Persistent Storage (ancienne version)

Data-only container

```
 docker run -d --volume /var/lib/cassandra/data --name cass-shared   alpine echo
Data Container
 docker logs  cass-shared
```

Cassandra application

```
 docker run -d --volumes-from cass-shared --name cass1 cassandra:2.2
```

Cassandra client

```
 docker run  -it --link cass1:cass cassandra:2.2 cqlsh cass
create keyspace docker_hello_world with replication = {'class' :
'SimpleStrategy','replication_factor': 1};

Vérification
select * from system.schema_keyspaces where keyspace_name = 'docker_hello_world';
 docker stop cass1
 docker rm -vf cass1
 docker ps
 docker ps -a
 docker run -d --volumes-from cass-shared --name cass2 cassandra:2.2
 docker run -it --rm --link cass2:cass cassandra:2.2 cqlsh cass
```

TP: Volume et Persistent Storage (récente version)

Création d'un volume

```
docker volume create --name cass-shared
```

Cassandra application

```
docker run -d -v cass-shared:/var/lib/cassandra/data  --name cass1 cassandra:2.2
```

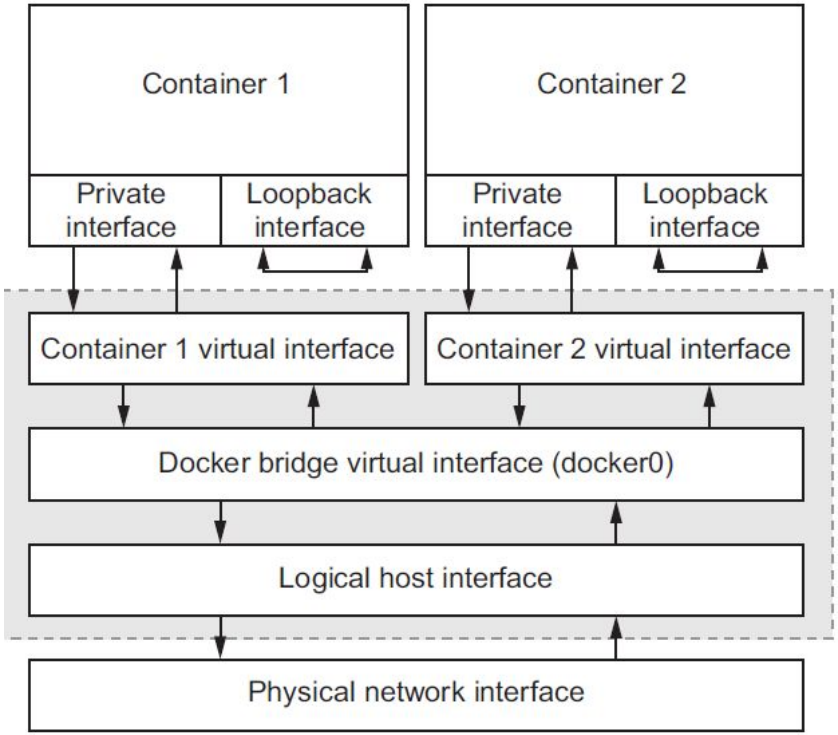
Cassandra client

```
docker run  --rm -it --link cass1:cass cassandra:2.2 cqlsh cass
create keyspace docker_hello_world with replication = {'class' :
'SimpleStrategy','replication_factor': 1};

Vérification
select * from system.schema_keyspaces where keyspace_name = 'docker_hello_world';
docker stop cass1
docker rm -vf cass1
docker ps
docker ps -a
docker run -d -v cass-shared:/var/lib/cassandra/data --name cass2 cassandra:2.2
docker run -it --rm --link cass2:cass cassandra:2.2 cqlsh cass
```

Docker Network

Docker utilise les fonctionnalités du système d'exploitation sous-jacent pour construire une topologie de réseau virtuel. Le réseau virtuel est local à la machine où Docker est installé et est composé d'itinéraires entre les conteneurs participants et le réseau plus large où l'hôte est attaché. Vous pouvez changer le comportement de ce réseau structure et dans certains cas, changer la structure elle-même en utilisant la ligne de commande pour démarrer le démon Docker et chaque conteneur.

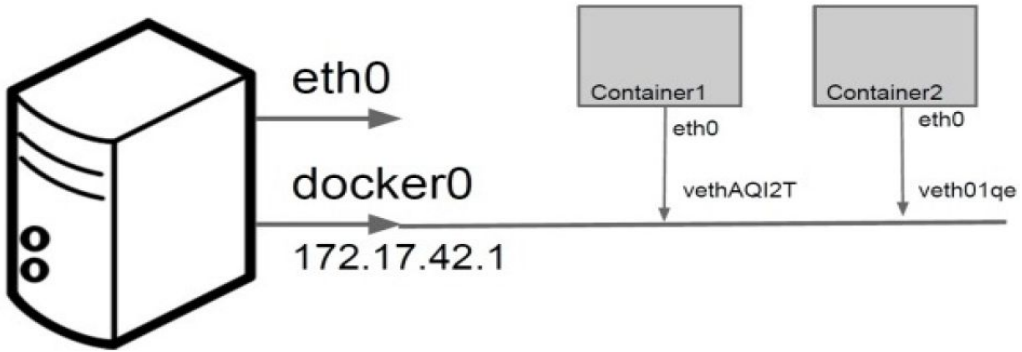


Docker network

Le bridge `docker0` est le coeur du réseau interne par défaut de Docker. Quand le daemon Docker est lancé, un bridge réseau est créé sur la machine hôte. Les interfaces réseau des containers communiquent avec ce bridge et ensuite avec le réseau extérieur.

`docker0` peut être configuré avec le flag `--net` qui a 4 modes de fonctionnement :

```
--net                          # mode par défaut.
--net=none                     # ne peut pas être connecté au réseau.
--net=container:$container2 # le container créé va partager son réseau avec le
container existant $container2.
--net=host                     # le container créé va partager son réseau avec la
machine hôte
```



Docker network

```
docker network ls
```

vous pouvez voir les 3 networks par défaut, bridge , host, none

docker run --name mariadb_storage -v /var/lib/mysql -it -d busybox

```
docker network inspect bridge
```

vous pouvez voir que le container mariadb_storage est dans le network bridge


```
 "691baf9517ec6f2a8229a97c450b210996d6d9bf289b3113db5af12772284c43": {
             "Name": "mariadb_storage",
             "EndpointID": "91a01719ea4b4735363d0beff872e5db4a5432bb149129b36a94ce20afbe6ed2",
             "MacAddress": "02:42:ac:11:00:05",
             "IPv4Address": "172.17.0.5/16",
             "IPv6Address": ""
         },
```


```
docker network create db_network
docker run --name mariadb -e MYSQL_ROOT_PASSWORD=password --network
db_network -d mariadb
```

Overlay network

overlay network est construit au-dessus dans autre réseau, comme par exemple VoIP, VPN, CDN sont des overlay networks.
Docker utilise des overlay networks , précisément en Swarm mode ou en sauvegardant une cle-valeur externe a Docker. Overlay networks permettent la création d'un réseau multi-hôte.

```
docker swarm init --advertise-addr <ip_machine>
```

Vous pouvez créer un nouveau network

```
docker network create --driver overlay --subnet 10.0.9.0/16 --opt encrypted
webapps_network
docker network ls
docker network inspect webapp_network
```

Orchestration de container: Docker-compose

Alors que nous nous dirigeons rapidement vers des environnements informatiques conteneurisés; application et conteneurs de données doivent être intelligemment agencés pour réaliser la nouvelle génération services logiciels.
Cependant, pour produire des conteneurs hautement fiables, les conteneurs spécifiques et les conteneurs génériques doivent être sélectionnés et démarrés en séquence afin de créer des conteneurs orchestrés.
Docker propose un outil d'orchestration standardisé et simplifié, nommé docker-compose pour réduire les charges de travail des développeurs ainsi que des administrateurs système.

Installation de docker-compose sous Ubuntu :

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.22.0/docker-compos
e-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose -version
```

Docker-compose

Faire un fork du repo Github
https://github.com/ambient-docker/python2-aws

Les commandes de base:

- docker-compose up
- docker-compose up -d
- docker-compose build
- docker-compose stop
- docker-compose rm
- docker-compose scale php=5

67 lines (62 sloc) | 1.03 KB

```
version: '2'

networks:
  lan_network:
    driver: bridge

services:
 digitaltoys:
  hostname: digit
  build: ./DigitalToys
  ports:
    - 8787:8080
  networks:
    - lan_network
 flask_login:
  hostname: flask
  build: ./flask_log_in
  links:
    - db:mysql
  depends_on:
    - db
  ports:
    - 5000:5000
  environment:
    DB_USER_USERNAME: hme
    DB_USER_PASSWORD: password
  networks:
    - lan_network

 web_application:
  hostname: web
  image: centos/httpd
  links:
    - db:mysql
  depends_on:
    - db
```

L'écosystème autour de Docker

- **Docker machine:**

  Docker Machine est un outil qui vous permet d'installer Docker (Engine) sur des hôtes virtuels et de gérer les hôtes avec des commandes docker-machine. Vous pouvez utiliser Machine pour créer des hôtes Docker sur votre Mac ou Windows local, sur votre réseau d'entreprise, dans votre centre de données ou sur des fournisseurs de cloud tels qu'Azure, AWS ou Digital Ocean.

- **Docker swarm:**

  Un swarm est un groupe de machines qui exécutent Docker et qui font partie d'un cluster. Une fois que vous êtes connecté, vous continuez à exécuter les commandes Docker auxquelles vous êtes habitué, mais elles sont maintenant exécutées sur un cluster par un Swarm manager. Les machines Swarm peuvent être physiques ou virtuelles. Après avoir rejoint un Swarm, elles sont appelés des nœuds.

- **Docker registry:**

  Docker Registry est une application côté serveur sans état et hautement évolutive qui vous permet de distribuer et de stocker des images Docker.

L'écosystème autour de Docker(2)

- **Kubernetes:**

  Kubernetes est un système open source permettant d'automatiser le déploiement, la scalabilité et la gestion des applications conteneurisées.

- **Openshift:**

  Openshift est basé sur Kubernetes, Docker et des outils Devops définissant ainsi un PaaS, une Plateforme As A Service pour améliorer la création et le déploiement d’applications.

Formation F2i

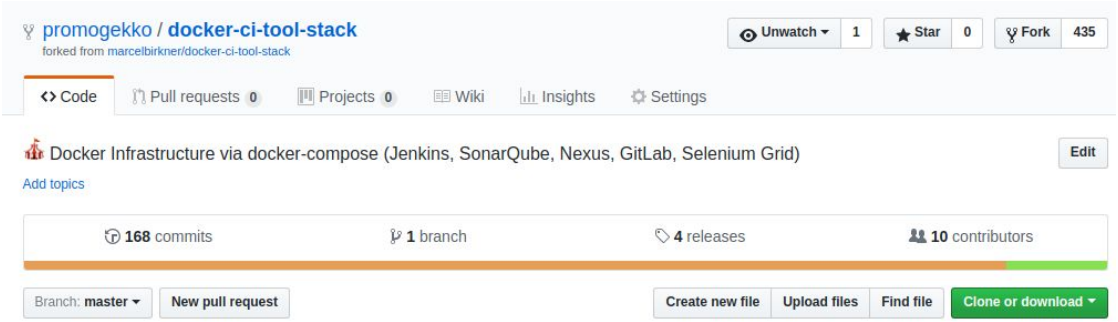
Les 9 Décisions à prendre avant d'utiliser Docker en production

1. Gestion des images de containers
2. Sélectionner un fournisseur Cloud
3. Acces au reseau et patch de sécurité
4. Load balancing de containers et de serveurs
5. Mise en place du process de déploiement
6. Inventaire des services
7. Gestion des logs
8. Monitoring de containers
9. Gestion des bases de données

La gestion des images des containers

Bien que la configuration d'un Dockerfile et d'un docker-compose.yml pour la création d'images dans votre environnement de développement soit assez simple, vous devez créer un processus cohérent pour créer des fichiers image Docker. Cela permettra d'éliminer toute préoccupation concernant les environnements locaux, tout en évitant les dépendances sur un serveur de développement en tant que votre seul moyen de construire de nouvelles images. Il vous permettra également de créer un pipeline de déploiement continu pour passer de la validation de code à l'image Docker sans intervention manuelle de votre équipe de développement.

Allez sur le projet promogekko sur github, faire une fork du repository docker-ci-tool-stack et ensuite un git clone de ce repository, et en fin faire un docker-compose up pour lancer la plateforme d'intégration continue.



Sélectionner un fournisseur de cloud

Une fois que vous avez une image Docker, vous devez la déployer dans une Docker Registry. De nombreux fournisseurs de cloud prennent désormais en charge le déploiement des conteneurs Docker. Étant donné que la plupart des ressources sont utilisées plutôt que les instances de conteneur.

Le processus de déploiement de conteneur varie selon les fournisseurs de cloud, ce qui peut compliquer le changement de fournisseur à l'avenir. Si vous souhaitez utiliser plusieurs fournisseurs de cloud ou empêcher le verrouillage, vous devez intégrer la prise en charge de plusieurs fournisseurs de cloud ou trouver une solution adaptée à vos besoins.

Accès au réseau et patch de sécurité

L'exécution de conteneurs dans un environnement de développement local ne crée aucun risque de sécurité sérieux. Tous les processus s'exécutent sur un seul hôte, isolé du risque d'intrusion réseau et de divers vecteurs d'attaque communs aux serveurs de production.

Les paramètres de développement sont assez ouverts pour permettre la mise au point pendant le développement. Dans un environnement de production, les paramètres réseau nécessitent plus d'attention. L'accès public ne devrait pas être possible à certains conteneurs, qui ne devraient être accessibles que par d'autres conteneurs au sein d'un réseau privé.

Vous devrez également suivre les correctifs de sécurité au fur et à mesure qu'ils sont annoncés, puis déterminer si vos hôtes et vos conteneurs sont tous sécurisés et si le correctif doit être installé.

Le déplacement de vos conteneurs vers la production nécessite une réflexion sur l'accès au réseau et le maintien de vos conteneurs et des hôtes Docker corrigés. Ne négligez pas cette étape critique pour votre environnement de production.

Sécuriser docker (partie 1)

Plus ou moins complexes à mettre en place et assurant des niveaux de sécurité différents, les axes de sécurisation d'une infrastructure « Dockerisée » sont les suivants :

- Durcir les serveurs hôtes hébergeant le démon Docker et les conteneurs.

- Les images doivent être à jour et ne posséder que le strict minimum. Éviter le stockage de secrets au sein d'une image, la présence de ssh ou d'outils d'administration. Moins le conteneur possède d'outils et de paquets, plus la tâche de compromission sera difficile.

- Construire un réseau cloisonné dont chaque composant Docker se voit attribuer un segment particulier (communication hôte-hôte, conteneurs, administration, servitudes, etc.). Définir des politiques sécurisées de communication et d'administration au sein l'infrastructure.

- Vérifier les sources de construction des images utilisées par les conteneurs et assurer un suivi de leurs évolutions en appliquant une signature numérique interne.

- Un scan de vulnérabilité de l'image doit être implémenté avant la validation de celle-ci en tant qu'image de confiance.

.

Sécuriser docker (partie 2)

- Limiter les appels système, les ressources d'un conteneur et minimiser l'exécution des conteneurs en mode privilégié. Le UserMapping du User NameSpace est vivement conseillé.
- Intégrer un mécanisme de contrôle d'accès et de gestion de rôle/profil des utilisateurs/administrateur de la solution (sécuriser l'API par un outil RBAC).
- Suivre les évolutions de configuration et centraliser les journaux d'événement.
- Surveiller le niveau de sécurité global régulièrement à l'aide d'audit et de scanneur réseau et applicatif. Cela permet d'identifier les vulnérabilités des serveurs hôtes et des applications hébergées dans des conteneurs et de suivre leur correction.
- Ne pas exposer directement un conteneur sur Internet, les mécaniques habituelles de protection doivent s'appliquer Reverse Proxy, Firewall/ WAF, etc.

Load balancing et containers et serveurs

Une fois que nous passons d'un seul service conteneur à plusieurs conteneurs sur un ou plusieurs hôtes, nous avons besoin de load balancer pour distribuer les demandes entrantes. L'utilisation d'outils tels que nginx ou HAProxy est une approche courante. La difficulté réside dans la mise à jour de leur configuration à mesure que les conteneurs sont créés et détruits, ainsi que lorsque de nouveaux hôtes Docker sont ajoutés à votre environnement pour une capacité supplémentaire. Notez que, à moins que vous ne prévoyez de déconnecter votre déploiement actuel pendant la mise à niveau, vous devez prendre en charge plusieurs versions en cours d'exécution en même temps. Votre stratégie d'équilibrage de charge doit en tenir compte, pour éviter de perdre des connexions ou d'acheminer le trafic vers la mauvaise version.

La tendance du marché actuel est vers l'utilisation de Paas, c'est-à-dire de plateformes complètes à base de containers qui contient un load balancer en natif comme Kubernetes, Rancher ou des OS spécifiques comme CoreOS, SmartOS et RancherOS.

Mise en place du process de déploiement

De nombreux développeurs pensent que les outils qu'ils utilisent dans un contexte de développement fonctionnent en production. Ce n'est pas le cas. Les configurations de Docker Compose varient considérablement d'un développement à l'autre. Les link de volumes aux port forwarding et aux configurations de réseau, le “câblage” de vos conteneurs va changer. La complexité augmentera au fur et à mesure que vous évoluerez vers un environnement multi-hôte. Vous aurez également des conteneurs supplémentaires que l'on ne trouve généralement pas en développement, tels que les agrégateurs de fichier logs, les bases de données externes, les ESB de messages HA et des containers pour renforcer la sécurité.

La coordination des différences dans les paramètres d'environnement nécessite un effort sur le scripting considérable. Ce ne sera pas aussi simple que de créer un container docker dans un environnement de développement. Prévoyez suffisamment de temps pour élaborer ces détails lorsque vous passez d'une application simple à conteneur unique à un ensemble complexe de conteneurs, chacun avec plusieurs instances devant être connectées à des load balancer pour distribuer les charges de travail. Au fur et à mesure que votre application évolue et que le trafic augmente, des mises à niveau progressive ou des stratégies de déploiement devront être utilisées pour éviter les pannes.

Inventaire de services

Au fur et à mesure que le nombre de conteneurs augmentera, le coût augmentera aussi car vous enregistrez plus de services pour votre application. Il existe une variété d'outils pour gérer ce processus, la plupart nécessitant une intégration et une configuration dans votre environnement de production Docker.
Cloud 66, https://www.cloud66.com/ , a trouvé un moyen simple pour gérer les registres de service en utilisant un serveur DNS interne.

Quel que soit votre choix, veillez à synchroniser vos enregistrements de service avec vos instances de conteneur et intégrez une stratégie d'équilibrage de charge lorsque les conteneurs sont répartis sur plusieurs hôtes Docker. Cela garantira que votre application peut être codée avec un nom de service général (par exemple, myservice.mycluster.local) qui peut être utilisé pour le routage vers l'instance de conteneur spécifique pour traiter cette demande.

Gestion des logs

L'utilisation de Docker Compose en développement rend la visualisation des journaux triviale et permet un dépannage rapide. Lorsque vous traitez plusieurs instances de conteneur sur un nombre illimité d'hôtes, il devient plus difficile de repérer les problèmes.

Les logs distribués permettent aux serveurs de collecter et d'agréger les entrées des logs sur un ou plusieurs serveurs de logs. Votre infrastructure de production nécessitera la prise en charge de l'agrégation des logs dans les conteneurs. Vous devrez également prendre en compte la manière dont vous envisagez d'afficher et de rechercher ces fichiers logs pour prendre en charge le dépannage.

Monitoring de containers

Surveiller vos conteneurs en production est essentiel. Des hôtes Docker aux conteneurs, vous devez connaître la disponibilité de chaque service et de l'ensemble du système. La sélection des bons outils et des stratégies de surveillance vous permettra de minimiser l'impact des pannes et de maximiser les ressources de votre hébergeur, ce qui se traduira par des clients plus satisfaits.
Zabbix XXL propose une solution toute faite de monitoring de containers et également la plateforme Red Hat openshift Enterprise inclut des extensions pour monitorer les pods de containers.

Formation F2i

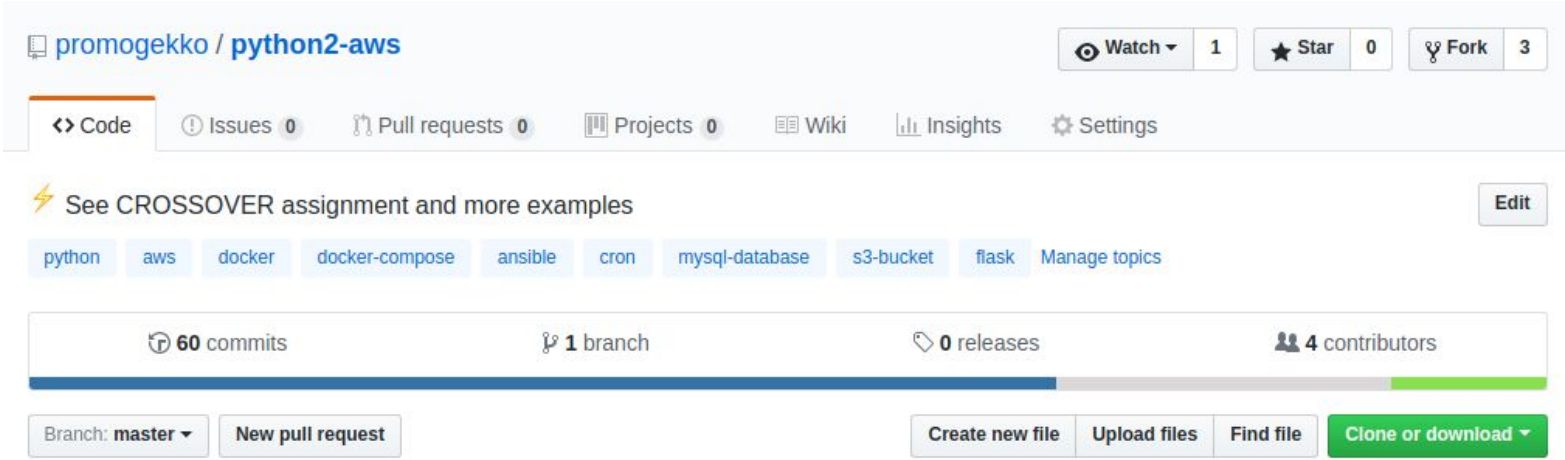
Database management

Dans un environnement de développement, les bases de données peuvent être hébergées dans un conteneur sans se soucier des performances d'E / S. Les environnements de production ne peuvent pas tolérer de mauvaises performances, en particulier si nous voulons offrir une expérience client exceptionnelle. La mise à l'échelle de la base de données pour gérer les E / S accrues en fonction de la demande, ainsi qu'une haute disponibilité et une stratégie de sauvegarde / restauration fiable sont essentielles pour l'exécution d'une application web moderne ou d'une API mobile. La stratégie que vous sélectionnez pour votre environnement de production aura un impact positif ou négatif sur vos clients.

En mettant en place une gestion des fichiers de base de données sur des volumes qui sont partagés avec les containers on peut avoir une redondance qui permet un securite accrue sur ces données.

Database

Allez sur le projet promogekko sur github, faire une fork du repository python2-aws et ensuite un git clone de ce repository, et en fin faire un docker-compose up pour lancer les containers liés



Database migration

Nous pouvons utiliser des containers Docker pour simuler une migration de base de données d'Oracle vers Postgresql.

Deux containers sont utilisés, l'un contenant une image Oracle 11g freeware , l'autre une image tutum de postgresql .
Installée sur le localhost, une version de Ora2Pg, un script Perl, effectue la conversion des objets Oracle vers Postgresql.

A chaque étape de la migration un docker commit est effectué pour prouver la validité de la migration.