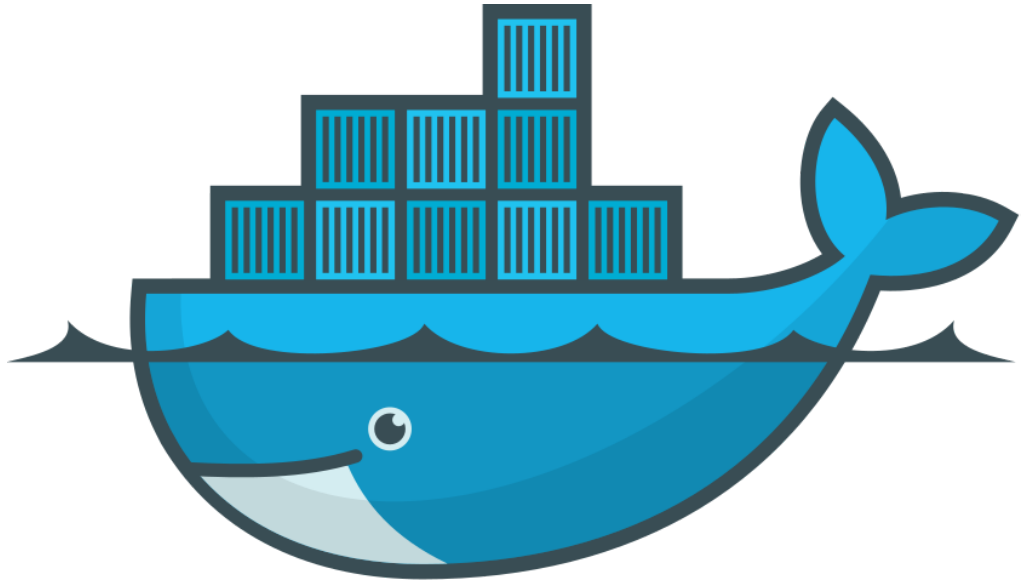


Docker : Mise en œuvre et déploiement de conteneurs virtuels

André Abramé

andre.abrame@laposte.net

Plan I

Introduction aux conteneurs

Présentation du concept de conteneur Linux

Définition et formats

Un conteneur, c'est quoi ?

Les mêmes idées que la virtualisation, mais sans virtualisation :

- Agnostique sur le contenu et le transporteur
- Isolation et automatisation
- Principe d'infrastructure consistante et répétable
- Peu d'overhead par rapport à une VM

Formats de container

- libcontainer (docker)
- BSD Jails
- Solaris Zones

Présentation du concept de conteneur Linux

Technologie sous-jacente

Namespaces

Permet d'isoler un container du reste du système.

- pid namespace : isolation des prcessus (PID : Process ID).
- net namespace : gestion des interfaces réseaux (NET : Networking).
- ipc namespace : gestion des accès aux ressources IPC (IPC : InterProcess Communication).
- mnt namespace : gestion des points de montage des systèmes de fichier (MNT : Mount).
- uts namespace : isolation du nom d'hote et de la version du noyau. (UTS : Unix Timesharing System).

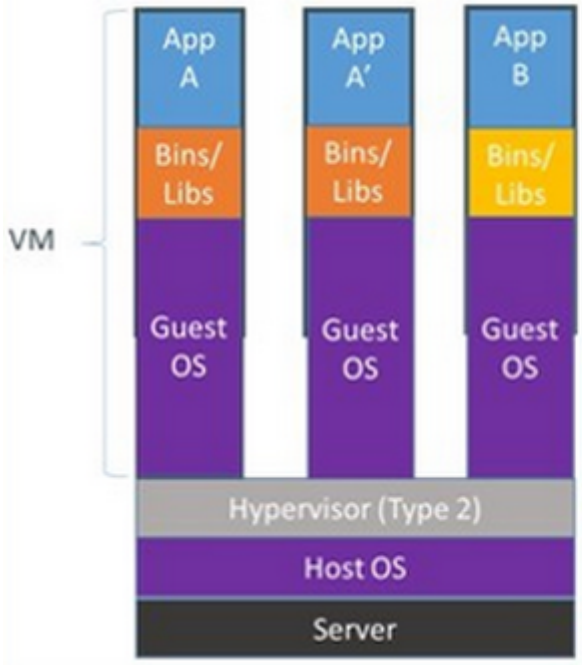
Control groups

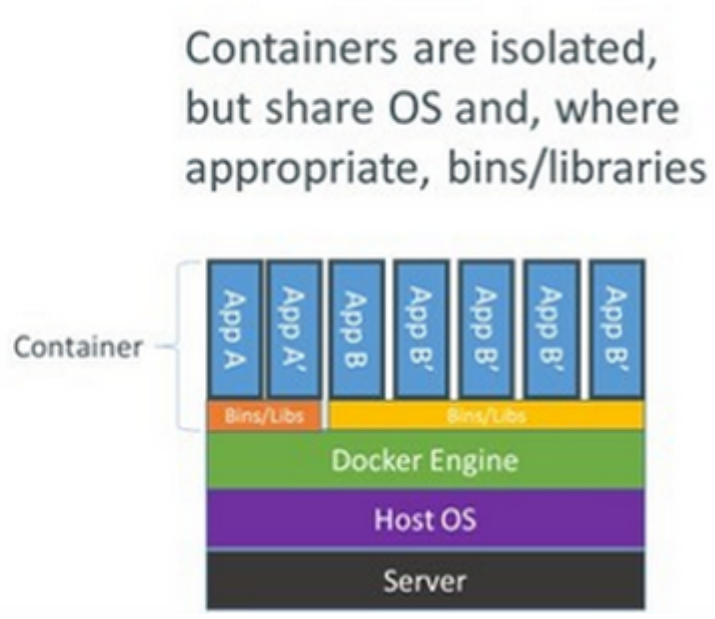
- permet de limiter les ressources (processeur, mémoire vive) utilisables par une application

Union file systems

- système de fichiers fonctionnant par couches

Les différences entre conteneurs et machines virtuelles





Machines virtuelles :

- lourde
- performances limitées
- exécute son propre OS
- virtualisation materielle
- démarrage en minutes
- réserve la mémoire
- entièrement isolé du système

Containers :

- légère
- performances natives
- partage l'OS hote
- virtualisation par l'OS
- démarrage en millisecondes
- nécessite moins de mémoire
- processus isolé du système, potentiellement moins sécurisé

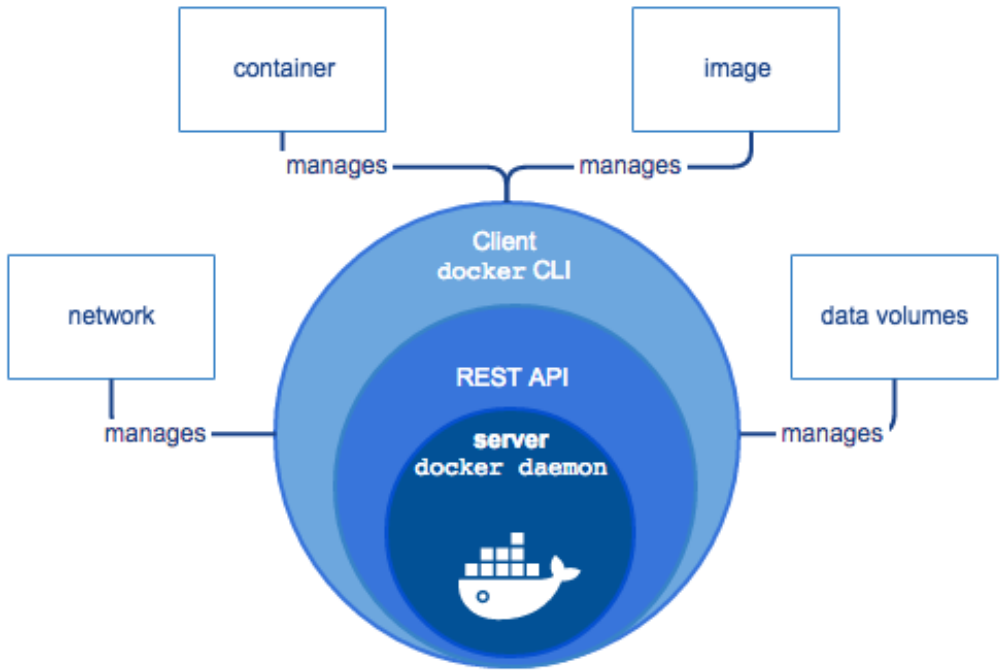
Présentation de Docker et de son architecture

Docker permet de :

- Proposer/diffuser des applications dans un environnement maitrisé (Docker/Dockerfiles)
  - Problèmes de versions et de dépendances pour une application
  - Versions de dépendances : Version spécifique d'OS,
  - installations personnelles ...
  - Proposer un package binaire universel
  - Documenter la procédure d'installation pour soi-même ou les curieux
  - Répétabilité, consistance
- Déployer un ensemble d'applications (docker-compose)
  - Description du fonctionnement d'un ensemble de services et de leurs interactions
  - Par exemple : Apache + PHP + Mariadb + Redis
- Passage à l'échelle d'un ensemble d'applications (docker-compose/swarm) : multi-hôtes
  - Augmenter le nombre de service lancé avec répartition de charge entre plusieurs machines.

Présentation de Docker et de son architecture

Docker Engine

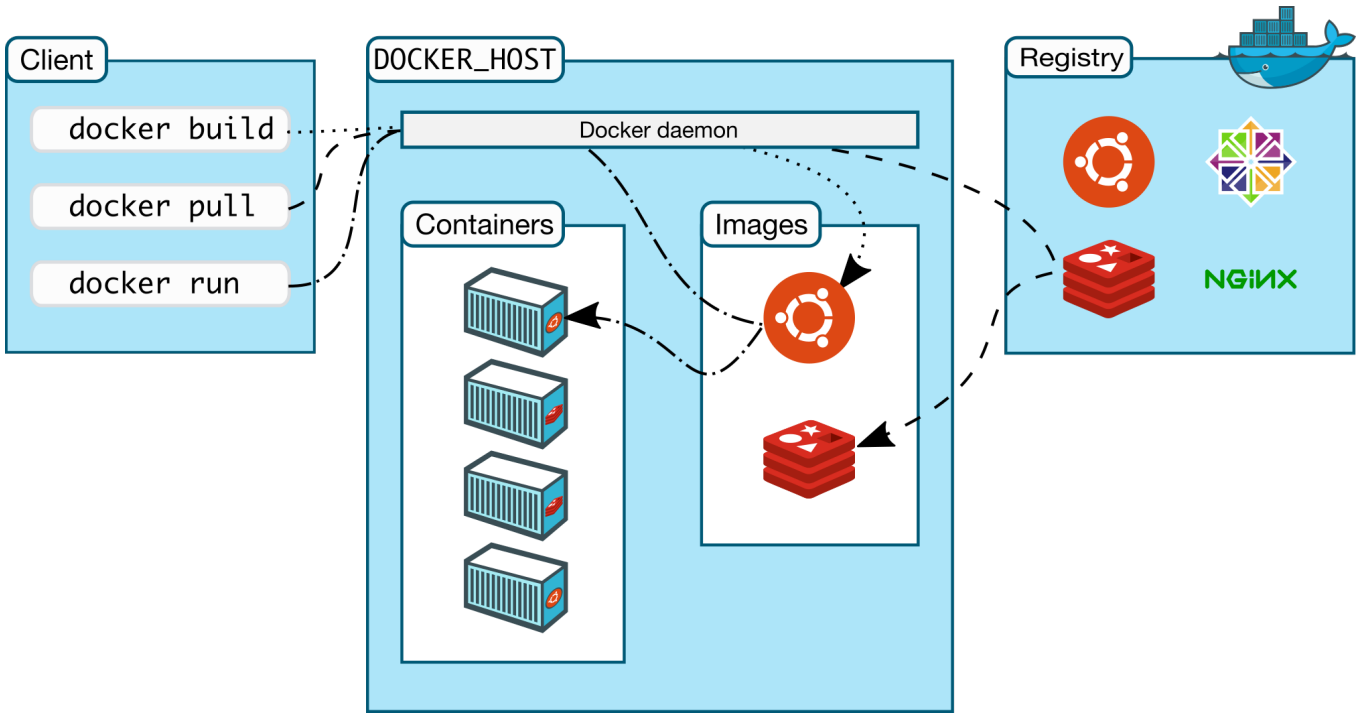


Le *Docker Engine* est une application client-serveur composée de :

- un serveur, le démon docker
- une API REST pour intéragir avec le démon
- une interface en ligne de commande

Présentation de Docker et de son architecture

Architecture de docker

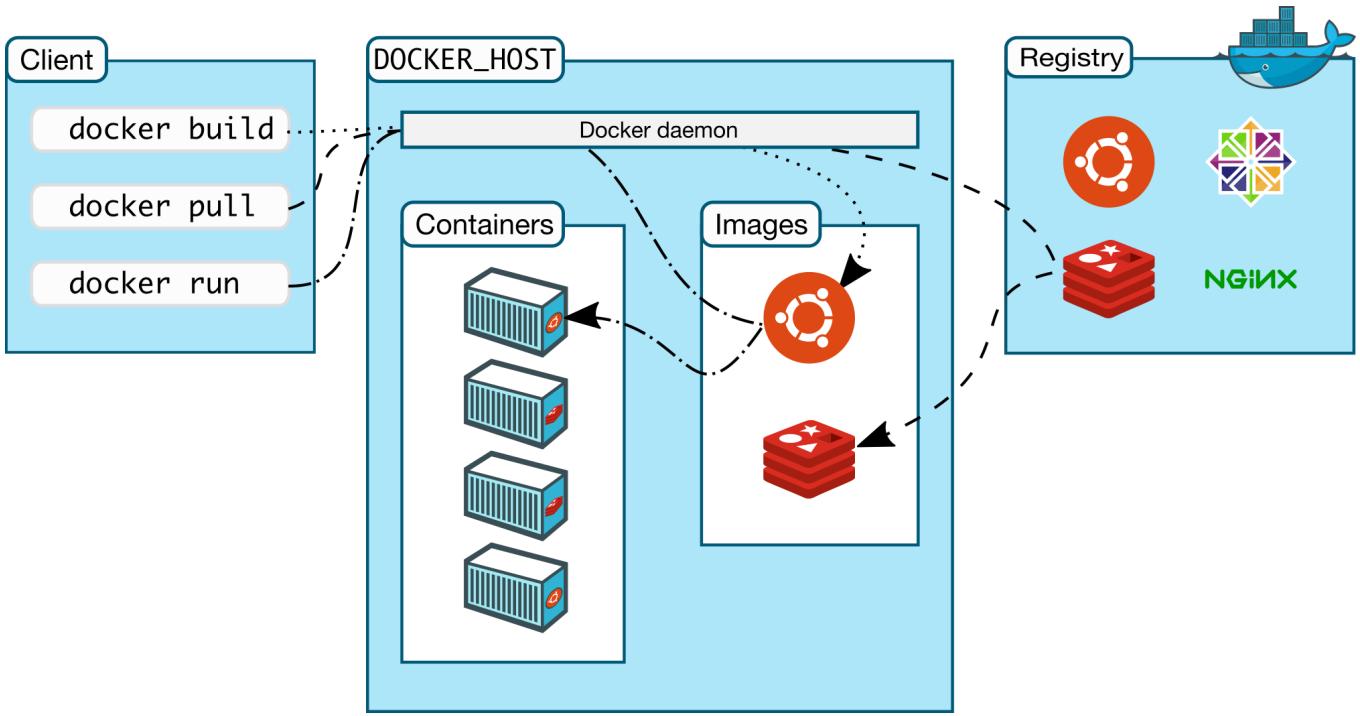


Le démon docker

- écoute les requètes de l'API docker
- gère les objets docker (images, containers, networks, volumes)
- communique avec d'autres démons docker

Présentation de Docker et de son architecture

Architecture de docker

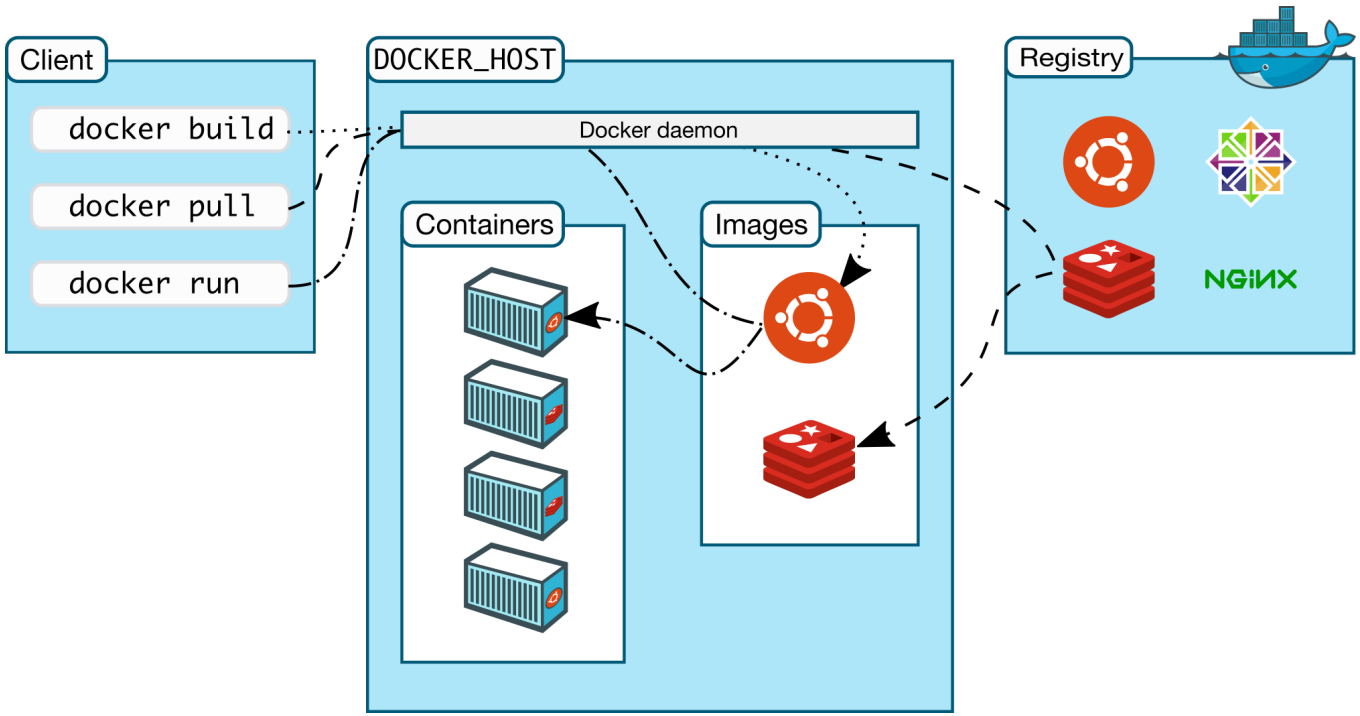


Le client docker

- permet à l'utilisateur d'intéragir avec docker
- peut communiquer avec plus d'un démon docker

Présentation de Docker et de son architecture

Architecture de docker



Les registres docker

- permet d'enregistrer / récupérer des images docker
- peut etre un registre publique (e.g. Docker Hub) ou privé

Présentation de Docker et de son architecture

Les objets docker

Image

- un template permettant de créer un container
- souvent basé sur une autre image (qu'on personnalise)
- crées via un *Dockerfile*

Container

- une instance exécutable d'une image
- exécute un seul processus
- par défaut relativement isolé du reste du système et des autres containers
- peut être connecté à un ou plusieurs réseaux, attaché à des espaces de stockage, ...
- peut prendre des paramètres lors de sa création

Service

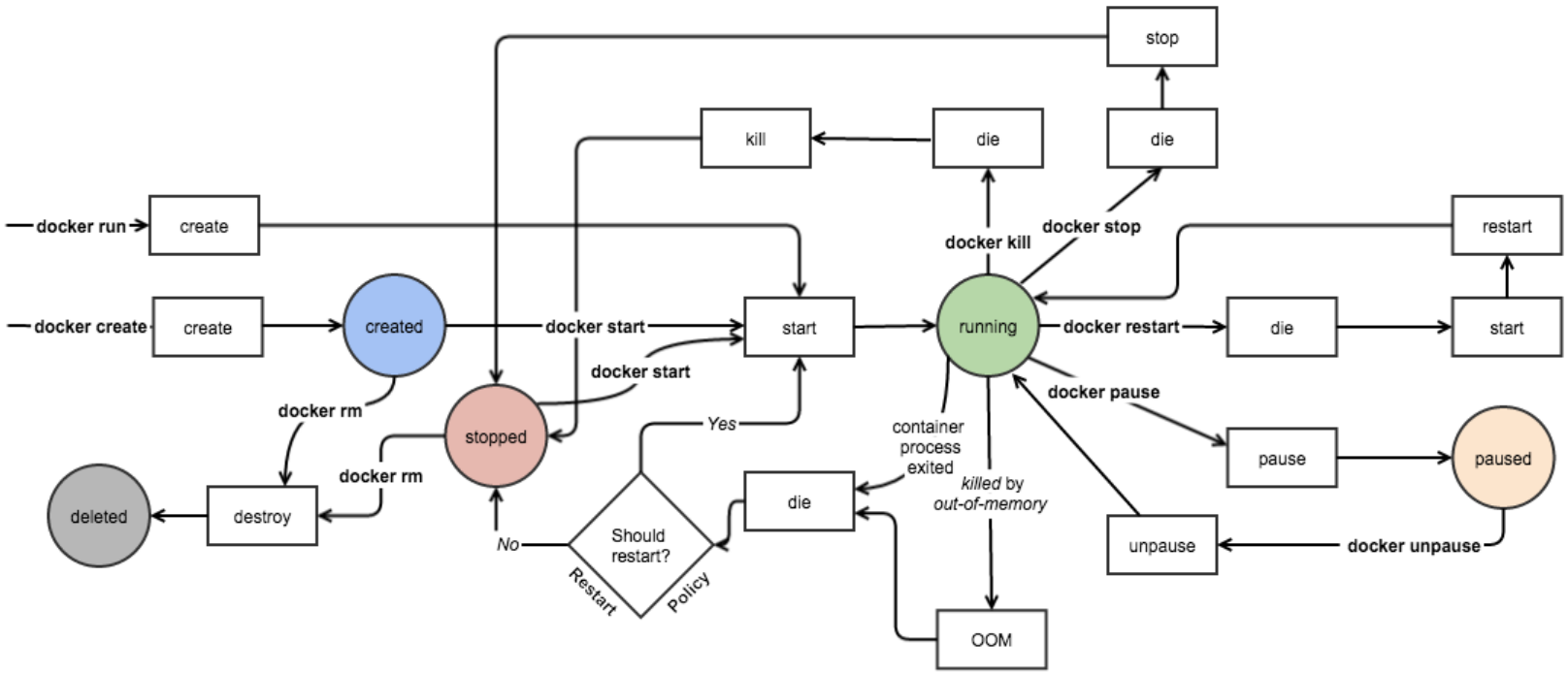
- permet le passage à l'échelle en exécutant des containers sur plusieurs démons docker
- *swarm* (nuée) : plusieurs *managers* et *workers*
- équilibre la charge entre les *workers*
- vu par l'utilisateur comme une unique application

Créer ses premiers conteneurs Docker

Installation de Docker

Suivre les instructions : `https://docs.docker.com/install/`

Le cycle de vie d'un conteneur

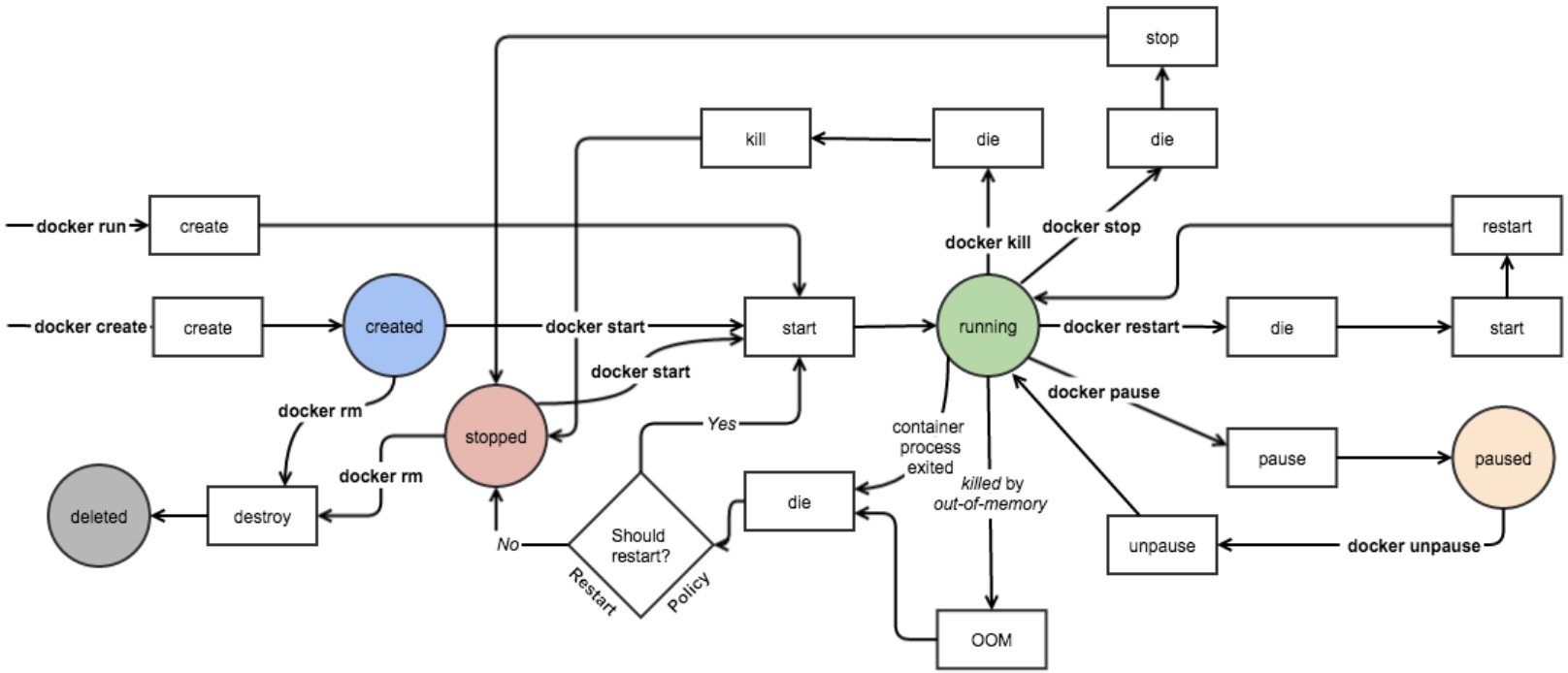


Création d'un container

Création d'un container qui sera exécuté plus tard :

```
docker container create --name <container-name> <image-name>
```

Le cycle de vie d'un conteneur

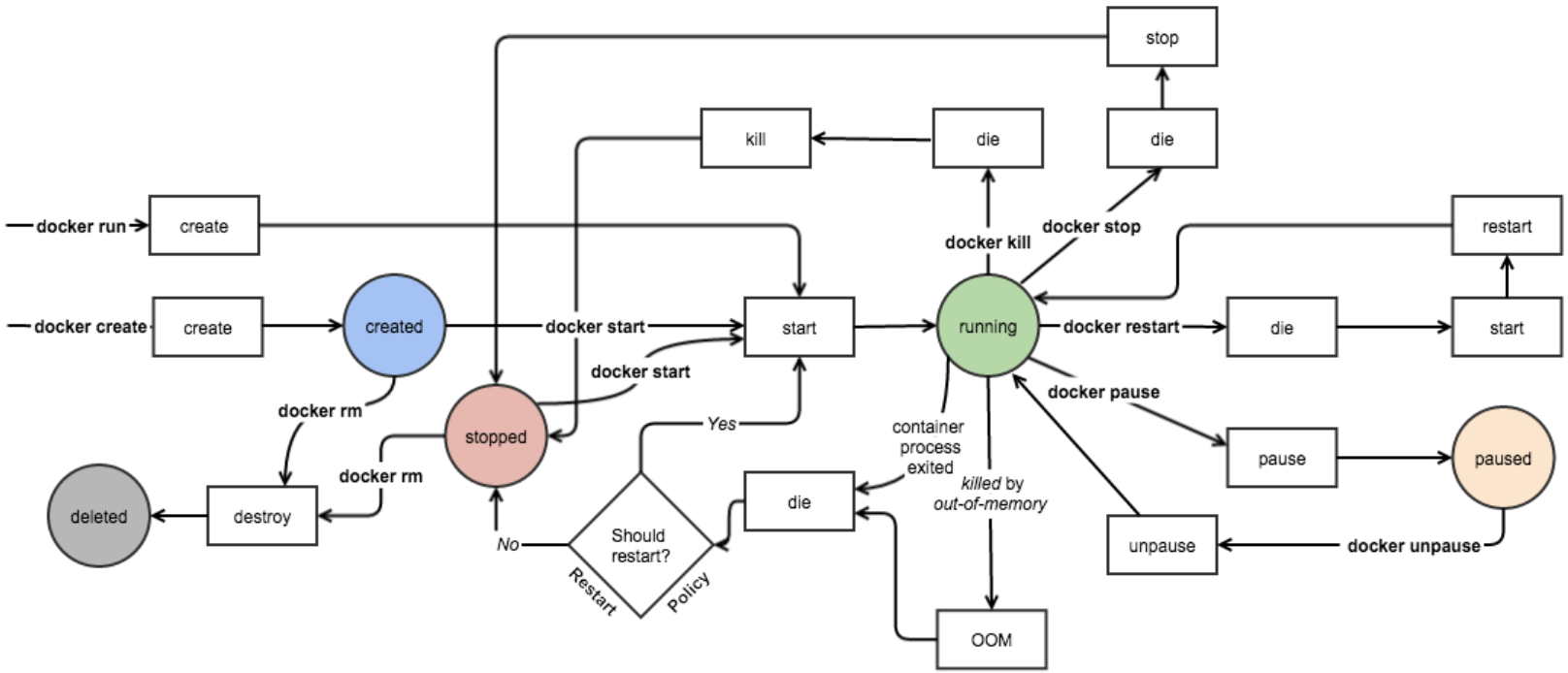


Exécution d'un container

Exécution d'un container en spécifiant l'image à utiliser et la commande à exécuter :

```
docker container run -it -d --name <container-name> <image-name> bash
```

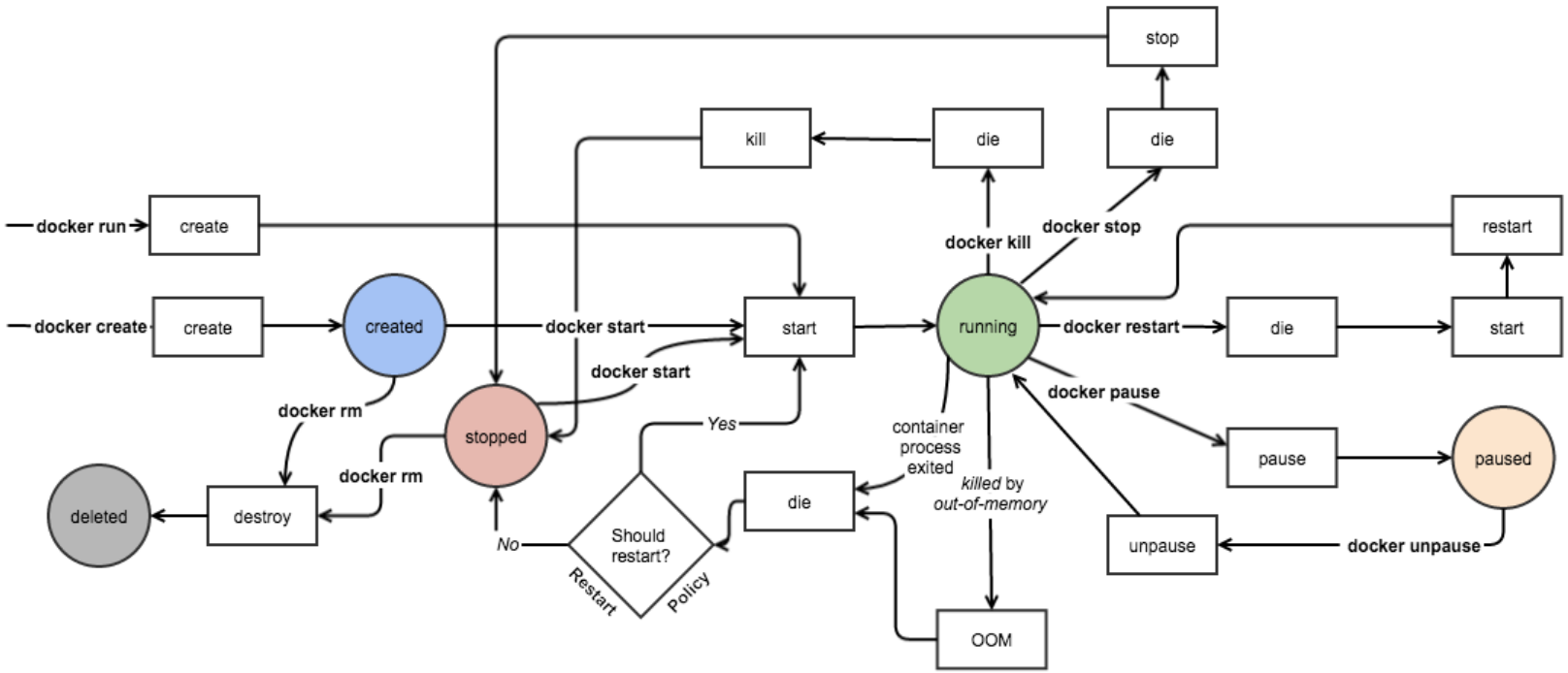
Le cycle de vie d'un conteneur



Mettre en pause un container

```
docker container pause <container-id/name>
```

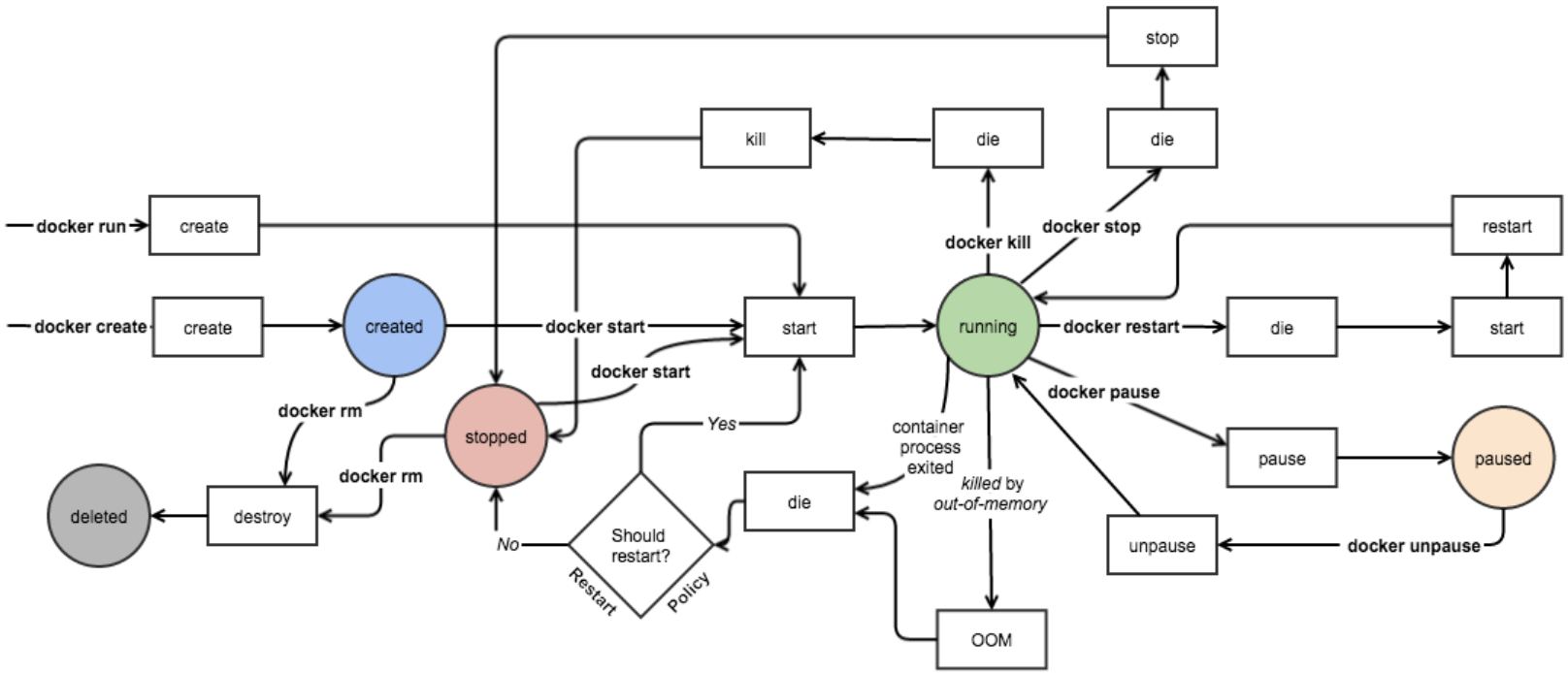
Le cycle de vie d'un conteneur



Relancer un container

```
docker container unpause <container-id/name>
```

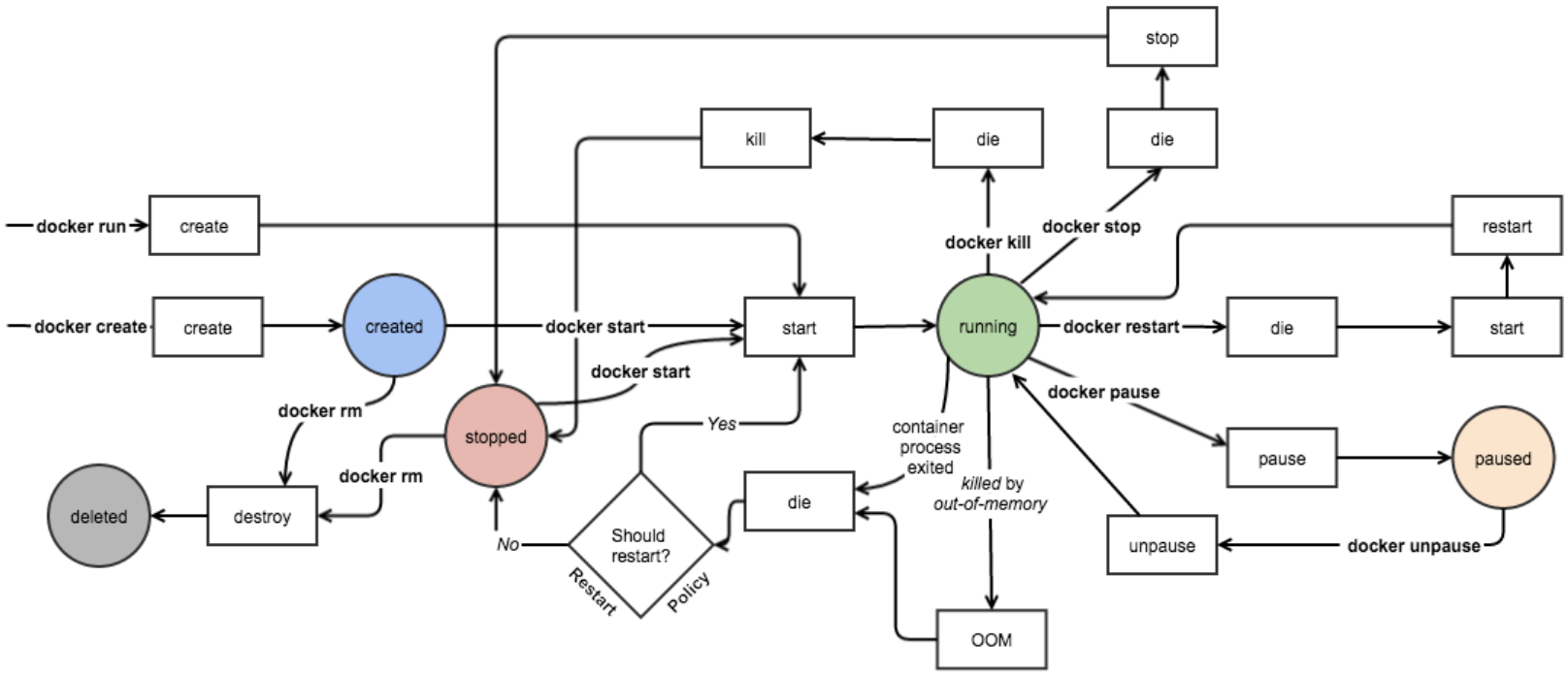
Le cycle de vie d'un conteneur



Démarrer l'exécution d'un container

```
docker container start <container-id/name>
```

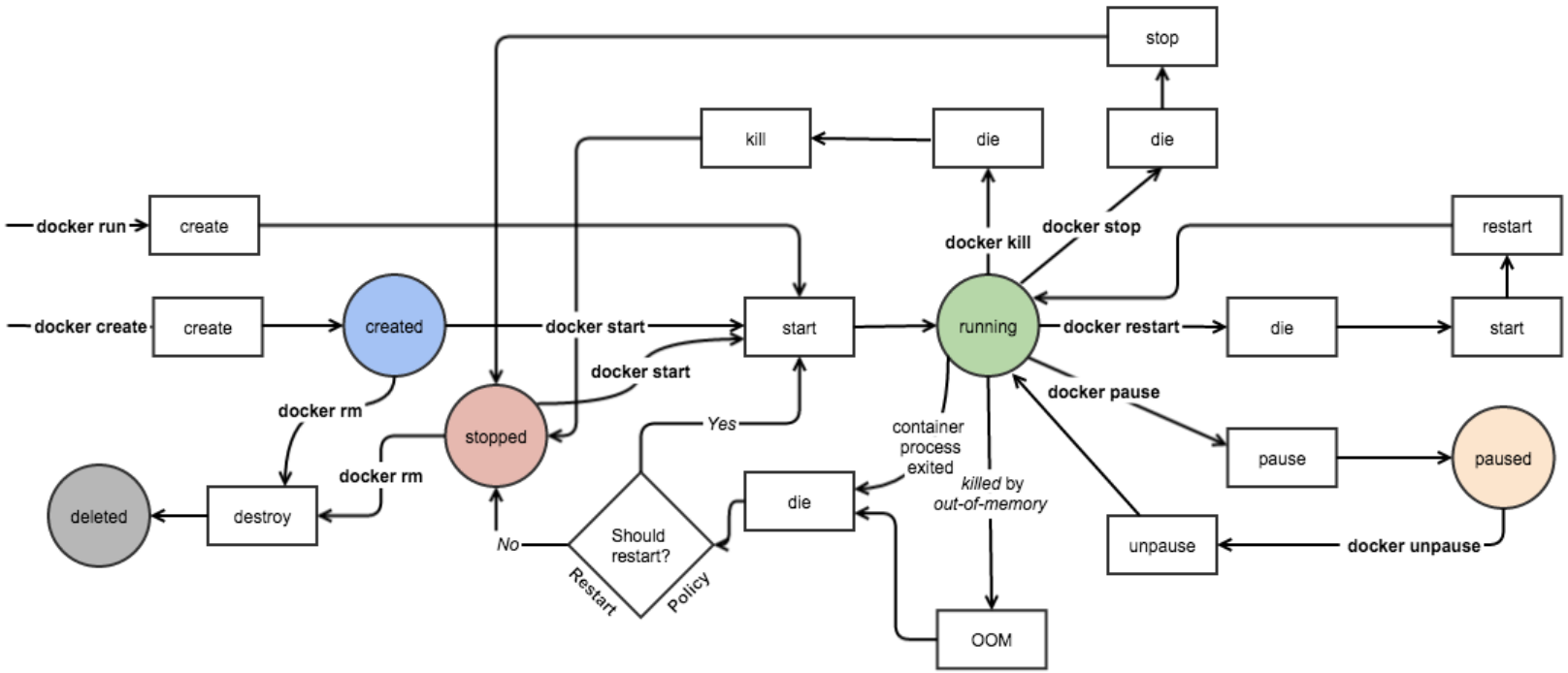
Le cycle de vie d'un conteneur



Arrêter l'exécution d'un container

```
docker container stop <container-id/name>
```

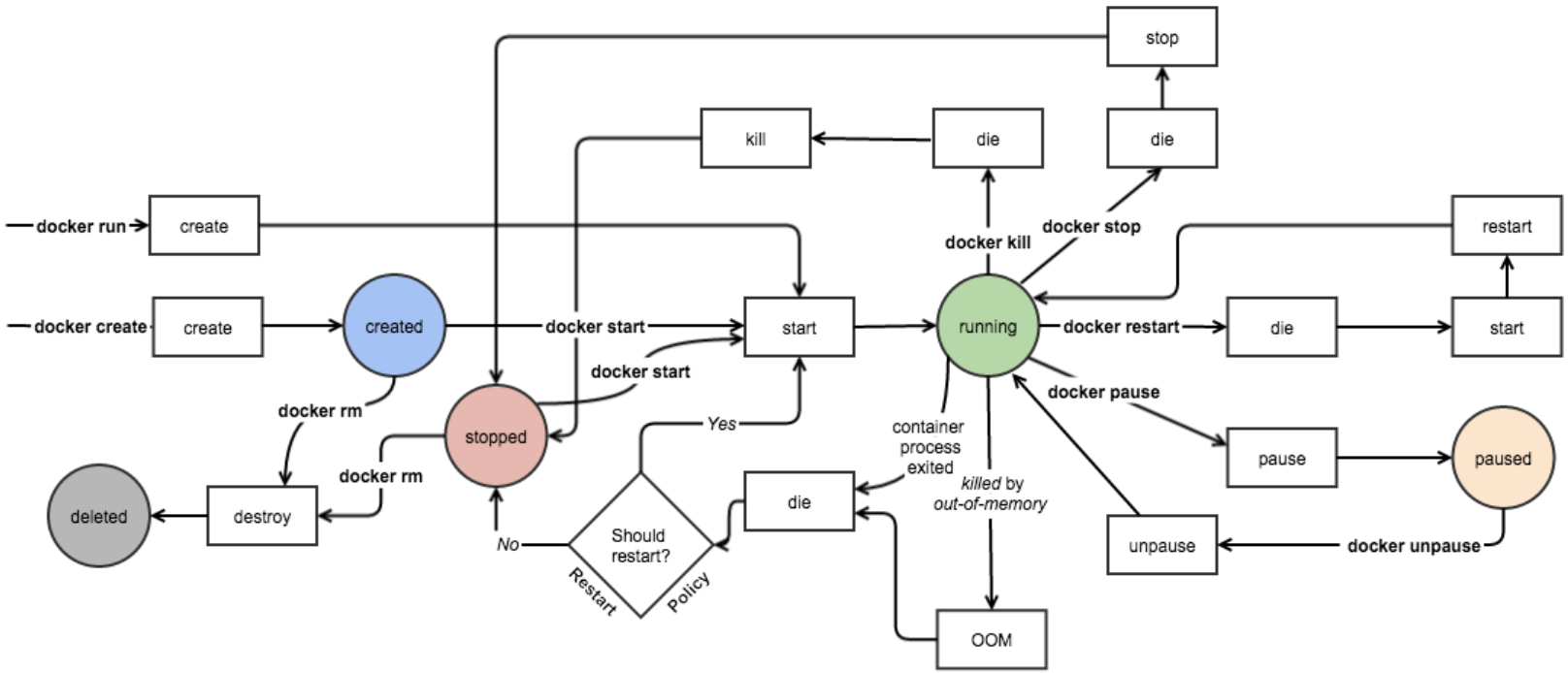
Le cycle de vie d'un conteneur



Redémarrer l'exécution d'un container

```
docker container restart <container-id/name>
```

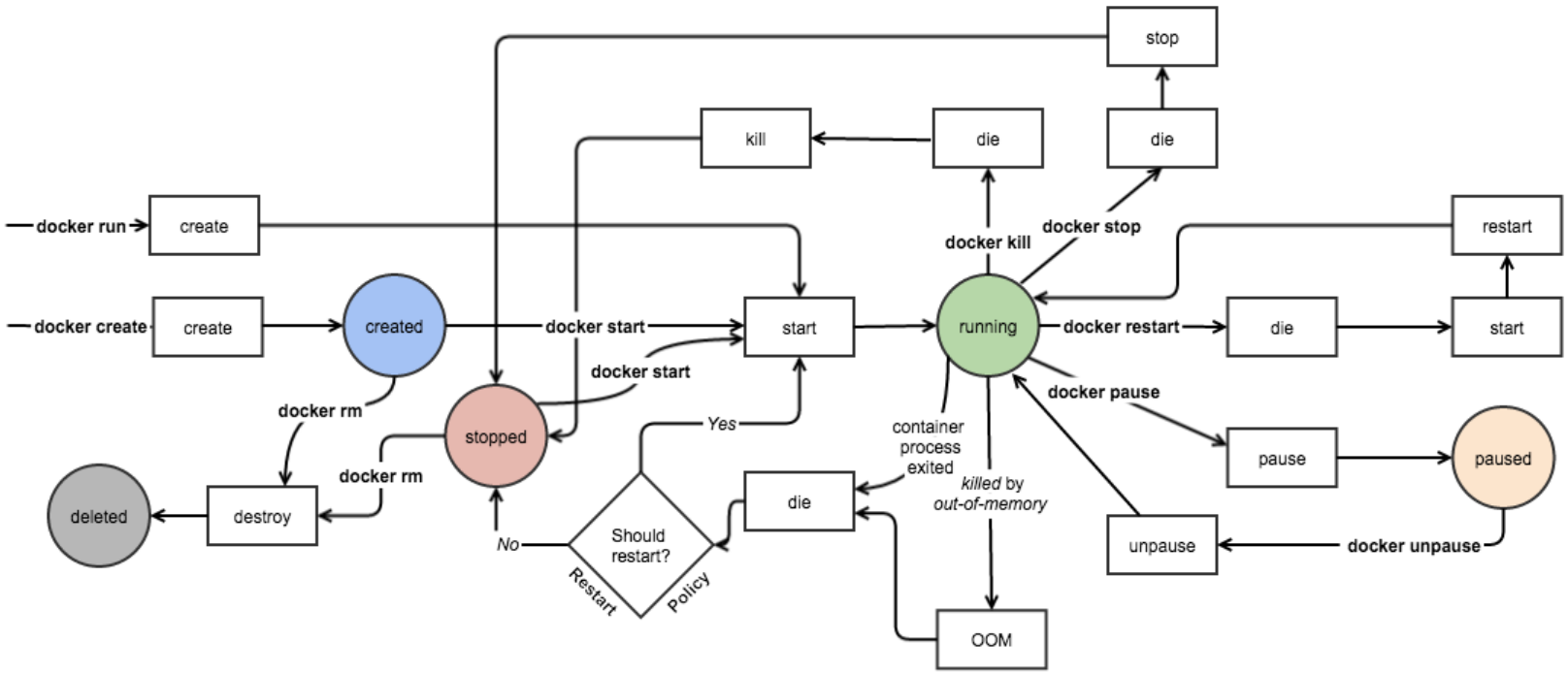
Le cycle de vie d'un conteneur



Tuer un container

```
docker container kill <container-id/name>
```

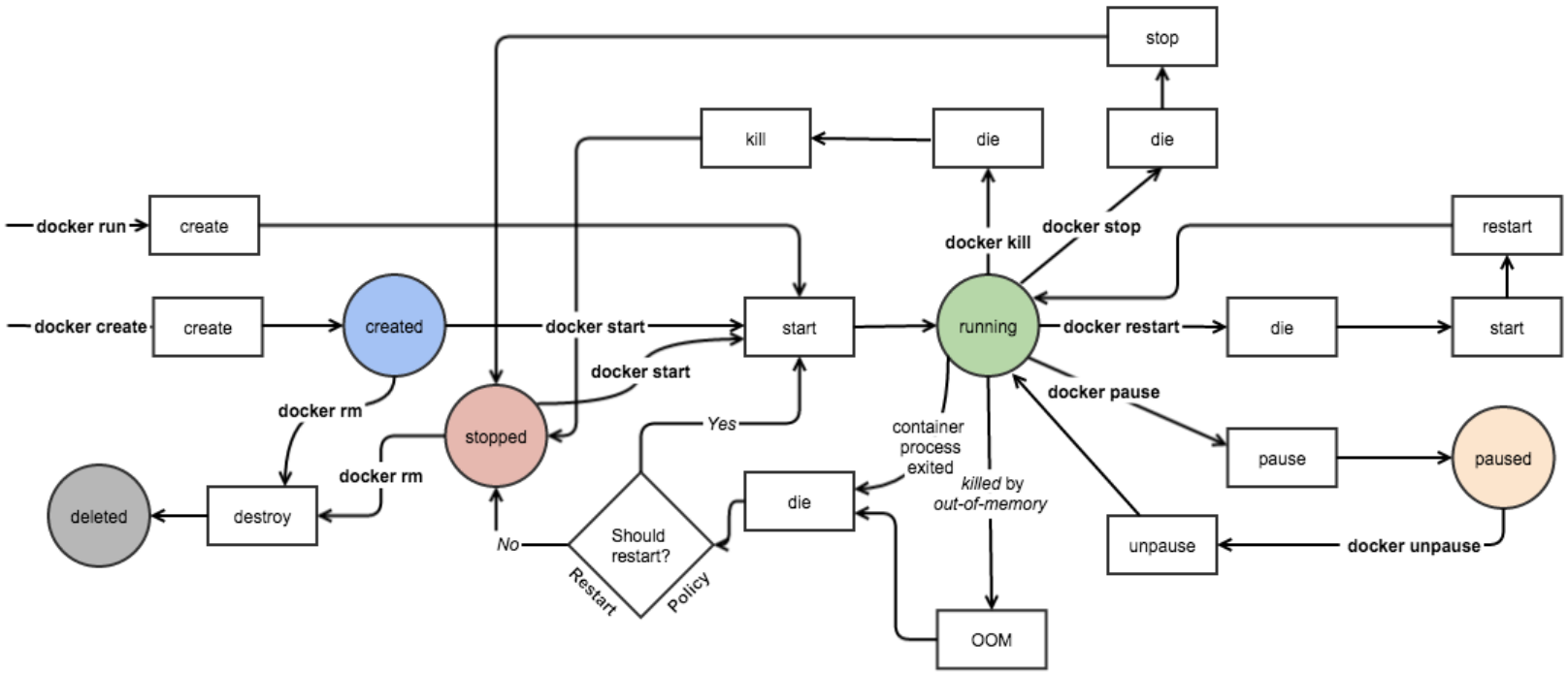
Le cycle de vie d'un conteneur



Suppression d'un container

```
docker container rm <container-id/name>
```

Le cycle de vie d'un conteneur



Afficher la liste des containers

```
docker container ls
```

Lancer un conteneur avec Docker Run (mode interactif, détaché...)

Docker run

```
docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

L'image et les paramètres de la commandes permettent de spécifier :

- si l'exécution doit être en tache de fond
- l'identification du container
- la configuration réseau
- les contraintes sur le CPU ou la RAM

Lancer un conteneur avec Docker Run (mode interactif, détaché...)

Mode détaché

- en arrière plan
- accès à stdout/stderr par `docker logs -f <container_ID>`

```
-d=true
```

ou

```
-d
```

Mode interactif

- par défaut
- peut attacher la console aux stdin, stdout et stderr du processus du container

```
-a=[]           : Attach to `STDIN`, `STDOUT` and/or `STDERR`
-t              : Allocate a pseudo-tty
--sig-proxy=true: Proxy all received signals to the process (non-TTY mode only)
-i              : Keep STDIN open even if not attached
```

Lancer un conteneur avec Docker Run (mode interactif, détaché...)

Identification du container

- permet d'y faire référence dans les commandes CLI
- par défaut, le démon docker choisira un nom aléatoire

```
--name <name>
```

Redirection de ports

- permet d'associer des ports de la machine hôte à ceux du container

```
-p 8080:80
```

Lancer un conteneur avec Docker Run (mode interactif, détaché...)

Exemple

```
docker container run -it ubuntu /bin/bash
```

- `run` : on veut lancer le container
- `-it` : mode interactif
- `ubuntu` : l'image à utiliser pour ce container
- `/bin/bash` : on lance bash

```
root@f7d1f0e2a800:/# cat /etc/issue
Ubuntu 18.04.3 LTS \n \l
root@f7d1f0e2a800:/# exit
exit
```

Interagir avec un conteneur depuis le host (exec, inspect, logs...)

Exécuter une commande dans un container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Options :

```
--detach , -d      Detached mode: run command in the background
--detach-keys      Override the key sequence for detaching a container
--env , -e        Set environment variables
--interactive , -i    Keep STDIN open even if not attached
--privileged      Give extended privileges to the command
--tty , -t        Allocate a pseudo-TTY
--user , -u        Username or UID (format: <name|uid>[:<group|gid>])
--workdir , -w      Working directory inside the container
```

Interagir avec un conteneur depuis le host (exec, inspect, logs...)

Récupérer les informations sur un objet docker

```
docker inspect [OPTIONS] NAME|ID [NAME|ID...]
```

Options :

```
--format , -f     Format the output using the given Go template
--size , -s     Display total file sizes if the type is container
--type     Return JSON for specified type
```

Interagir avec un conteneur depuis le host (exec, inspect, logs...)

Afficher les logs d'un container

```
docker logs [OPTIONS] CONTAINER
```

Options :

```
--details     Show extra details provided to logs
--follow , -f     Follow log output
--since     Show logs since timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for 42
↪  minutes)
--tail   all   Number of lines to show from the end of the logs
--timestamps , -t     Show timestamps
--until     Show logs before a timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for
↪  42 minutes)
```

Les images Docker

Qu'est-ce qu'une image Docker

Image

- un template permettant de créer un container
- souvent basé sur une autre image (qu'on personnalise)
- crées à partir d'un container ou d'un *Dockerfile*

Créer une image à partir d'un conteneur

Principe

Créer un nouveau modèle de container à partir d'un container existant.

Usage

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

- OPTIONS :
  - `--author , -a` Auteur
  - `--change , -c` Instructions pour modifier la configuration de l'image
  - `--message , -m` Message de commit
  - `--pause , -p` Mettre le container en pause lors du commit
- CONTAINER : le nom du container servant de base à la nouvelle image
- REPOSITORY : le dépôt dans lequel enregistrer l'image

Références

`https://docs.docker.com/engine/reference/commandline/commit/`

Créer une image à partir d'un conteneur

Exemple de commit d'un container

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
↪  PORTS               NAMES
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  desperate_dubinsky
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  focused_hamilton

$ docker commit c3f279d17e0a  svendowideit/testimage:version3

f5283438590d

$ docker images

REPOSITORY                      TAG                 ID                  CREATED
↪  SIZE
svendowideit/testimage          version3            f5283438590d        16 seconds ago
↪  335.7 MB
```

Créer une image à partir d'un conteneur

Exemple de commit d'un container avec changement de la configuration (1)

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
↪  PORTS               NAMES
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  desperate_dubinsky
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  focused_hamilton

$ docker inspect -f "{{ .Config.Env }}" c3f279d17e0a

[HOME=/ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin]

$ docker commit --change "ENV DEBUG true" c3f279d17e0a  svendowideit/testimage:version3

f5283438590d

$ docker inspect -f "{{ .Config.Env }}" f5283438590d

[HOME=/ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin DEBUG=true]
```

Créer une image à partir d'un conteneur

Exemple de commit d'un container avec changement de la configuration (2)

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
↪  PORTS               NAMES
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  desperate_dubinsky
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
↪  focused_hamilton

$ docker commit --change='CMD ["apachectl", "-DFOREGROUND"]' -c "EXPOSE 80" c3f279d17e0a
↪  svendowideit/testimage:version4

f5283438590d

$ docker run -d svendowideit/testimage:version4

89373736e2e7f00bc149bd783073ac43d0507da250e999f3f1036e0db60817c0

$ docker ps

CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
↪  PORTS               NAMES
89373736e2e7        testimage:version4  "apachectl -DFOREGROU"   3 seconds ago       Up 2
↪  seconds          80/tcp              distracted_fermat
c3f279d17e0a        ubuntu:12.04        /bin/bash                7 days ago          Up 25
↪  hours                                desperate_dubinsky
197387f1b436        ubuntu:12.04        /bin/bash                7 days ago          Up 25
↪  hours                                focused_hamilton
```

Créer une image à partir d'un "Dockerfile"

Commande build

Principe

Utilisation d'un fichier Dockerfile décrivant la nouvelle image.

Usage

```
docker build [OPTIONS] PATH | URL | -
```

- OPTIONS (principales) :

`--file , -f`	Nom du Dockerfile (défaut : 'PATH/Dockerfile')
`--cpu-shares , -c`	Limite processeur
`--memory , -m`	Limite mémoire vive
`--tag , -t`	Nom et tag de l'image, au format `name:tag`

- PATH : spécifie le chemin du contexte de l'image
- URL : spécifie l'url à utiliser comme contexte de l'image (dépôt GIT, archive, ...)
- - : lit le Dockerfile depuis STDIN

Références

`https://docs.docker.com/engine/reference/commandline/build/`

Créer une image à partir d'un "Dockerfile"

Exemples

Exemple avec PATH

```
$ docker build .
```

Utilise le répertoire courant comme contexte.

Exemple avec URL (archive)

```
$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz
```

Le démon télécharge et extrait l'archive, puis utilise le contenu comme contexte. Le chemin du Dockerfile est spécifié explicitement par `-f`.

Exemple avec URL (git)

```
$ docker build https://github.com/docker/rootfs.git#container:docker
```

Utilise comme contexte le répertoire `docker` de la branche `container` du dépôt.

Créer une image à partir d'un "Dockerfile"

Fichier .dockerignore

Role

Permet d'exclure certains fichiers / répertoires du contexte.

Syntaxe

- commentaire

```
# comment
```

- exclusion

```
*/temp*      # exclu les fichiers et repertoires commençant par temp situés dans un
↪  sous-répertoire direct de la racine
*/*/temp*    # exclu les fichiers et repertoires commençant par temp situés dans un
↪  sous-répertoire situés à deux niveau de la racine
temp?        # exclu les fichiers et repertoires commençant par temp et suivis d'un
↪  unique caractère situés directement à la racine
```

- exception à une exclusion

```
*.md          # règle d'exclusion : tous les fichiers ayant l'extension .md
!README.md    # exception à la règle précédente
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Role

Document texte contenant les commandes permettant de construire une nouvelle image.

Références

`https://docs.docker.com/engine/reference/builder/`

Contenu

- Commentaires :

```
# comment
```

- Directives pour le parser, placées au tout début du Dockerfile :

```
# directive=value
```

- Instructions, une par ligne, multiligne avec \ :

```
INSTRUCTION arguments
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Directive syntax

Permet de préciser le builder à utiliser pour construire l'image.

```
# syntax=[remote image reference]
```

Directive escape

Change le caractère d'échappement (\ par défaut). Utile en particulier pour la manipulation de chemins sous Windows.

```
# escape=\ (backslash)
```

Instruction FROM

Précise l'image de base à utiliser. Doit être la première instruction (commentaire et directive exceptés) du Dockerfile.

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Instruction RUN

Exécute la commande sur l'image actuelle et commit le résultat (ajoute une couche).

```
# shell form
RUN <command>
# exec form
RUN ["<executable>", "<param1>", "<param2>"]
```

Instruction ENV

Initialise des variables d'environnement.

```
ENV <key> <value>
ENV <key>=<value> [<key>=<value> ...]
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Instruction CMD

Configure la commande devant être exécutée par les containers basés sur cette image. Ne peut être présent qu'une fois dans un Dockerfile. Sera écrasé par ENTRYPOINT ou par les arguments en ligne de commande.

```
# exec form, this is the preferred form
CMD ["<executable>","<param1>","<param2>"]
# as default parameters to ENTRYPOINT
CMD ["<param1>","<param2>"]
# shell form
CMD <command> <param1> <param2>
```

Instruction ENTRYPOINT

Configure la commande devant être exécutée par les containers basés sur cette image. Ne peut être présent qu'une fois dans un Dockerfile.

```
# exec form, preferred
ENTRYPOINT ["<executable>", "<param1>", "<param2>"]
# shell form
ENTRYPOINT <command> <param1> <param2>
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Instruction ADD

Copie des fichiers et répertoires locaux ou distant dans le système de fichier de l'image.

```
ADD <src> [<src> ...] <dest>
#  the following form is required for paths containing whitespace
ADD ["<src>", ... "<dest>"]
```

Instruction COPY

Copie des fichiers et répertoires locaux dans le système de fichier de l'image.

```
COPY <src> [<src> ...] <dest>
#  the following form is required for paths containing whitespace
COPY ["<src>", ... "<dest>"]
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Instruction EXPOSE

Précise que le container écoutera un port à l'exécution.

```
EXPOSE <port> [<port> ...]
```

Instruction VOLUME

Créer un point de montage.

```
VOLUME ["<path>", ...]
VOLUME <path> [<path> ...]
```

Créer une image à partir d'un "Dockerfile"

Fichier Dockerfile

Autres instructions

- `MAINTAINER` : précise l'auteur de l'image

```
MAINTAINER <name>
```

- `LABEL` : ajoute des méta-données à l'image

```
LABEL <key>=<value> [<key>=<value> ...]
```

- `USER` : précise l'utilisateur ou l'UID lors de l'exécution de l'image

```
USER <username | UID>
```

- `WORKDIR` : précise le répertoire de travail pour les commandes `RUN`, `CMD`, `ENTRYPOINT`, `COPY` et `ADD`

```
WORKDIR </path/to/workdir>
```

- ...

Créer une image à partir d'un "Dockerfile"

Exemples

Image avec PostgreSQL (1/2)

Dockerfile

```
#
# example Dockerfile for https://docs.docker.com/engine/examples/postgresql_service/
#

FROM ubuntu:16.04

# Add the PostgreSQL PGP key to verify their Debian packages.
# It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc
RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
↪  B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8

# Add PostgreSQL's repository. It contains the most recent stable release of PostgreSQL,
↪  ``9.3``.
RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" >
↪  /etc/apt/sources.list.d/pgdg.list

# Install ``python-software-properties``, ``software-properties-common`` and PostgreSQL 9.3
#  There are some warnings (in red) that show up during the build. You can hide
#  them by prefixing each apt-get statement with DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y python-software-properties software-properties-common
↪  postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3

# Note: The official Debian and Ubuntu images automatically ``apt-get clean``
# after each ``apt-get``

# Run the rest of the commands as the ``postgres`` user created by the ``postgres-9.3`` package
↪  when it was ``apt-get installed``
USER postgres
```

Créer une image à partir d'un "Dockerfile"

Exemples

Image avec PostgreSQL (2/2)

Dockerfile

```
# Create a PostgreSQL role named ``docker`` with ``docker`` as the password and then create
# a database `docker` owned by the ``docker`` role. Note: here we use ``&&\`` to run commands
# one after the other - the ``\`` allows the RUN command to span multiple lines.
RUN    /etc/init.d/postgresql start &&\
psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
createdb -O docker docker

# Adjust PostgreSQL configuration so that remote connections to the database are possible.
RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/9.3/main/pg_hba.conf

# And add ``listen_addresses`` to ``/etc/postgresql/9.3/main/postgresql.conf``
RUN echo "listen_addresses='*'" >> /etc/postgresql/9.3/main/postgresql.conf

# Expose the PostgreSQL port
EXPOSE 5432

# Add VOLUMEs to allow backup of config, logs and databases
VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

# Set the default command to run when starting the container
CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c",
↪  "config_file=/etc/postgresql/9.3/main/postgresql.conf"]
```

Construction de l'image

```
$ docker build -t eg_postgresql .
```

Execution dans un container

```
$ docker run --rm -P --name pg_test eg_postgresql
```

Les registres docker

Une registre docker, qu-est ce que c'est ?

- système de stockage et de récupération
- contient des images dockers
- indexation par nom et tag

Le hub public

- Dépôt public (push/pull gratuit)
- Dépôt d'images officielles (sans "/") et d'images tiers
  - Systèmes d'exploitation : debian, ubuntu, centos, alpine, ...
  - Services conteneurisés : php, nginx, mariadb, ...

Registre privé

- Registre docker hébergé par l'utilisateur
- Déployé dans un container docker

Les registres docker

Principales commandes

- `docker search [OPTIONS] TERM`
  Recherche d'images dans le Docker Hub.
- `docker image pull [OPTIONS] NAME[:TAG|@DIGEST]`
  Récupération d'une image depuis un dépôt.
- `docker image push [OPTIONS] NAME[:TAG]`
  Envoi d'une image à un dépôt.

Les registres docker

Stocker et récuperer des images depuis le "Docker Hub"

Création d'un compte et d'un dépôt

1. Créer un compte sur le Docker Hub : `https://hub.docker.com/`
2. Créer un nouveau dépôt.

Création d'une image et stockage sur Docker Hub

1. Se connecter au Docker Hub :

```
docker login
```

2. Créer une image (avec un tag) :

```
docker build -t <your_username>/<your_repo> - <<EOF
FROM busybox:latest
CMD echo "Hello world!"
EOF
```

3. Pusher l'image :

```
docker push <your_username>/<your_repo>
```

4. Se déconnecter du Docker Hub :

```
docker logout
```

Les registres docker

Stocker et récuperer des images depuis le "Docker Hub"

Récupération d'une image depuis Docker Hub

1. Supprimer les copies locales de l'image :

```
docker image rm <your_username>/<your_repo>
```

2. Se connecter au Docker Hub :

```
docker login
```

3. Récupérer l'image depuis Docker Hub

```
docker pull <your_username>/<your_repo>
```

4. Vérifier que l'image est présente localement :

```
docker image ls
```

5. Se déconnecter du Docker Hub :

```
docker logout
```

Les registres docker

Mettre en place un registry privé et y stocker ses images

Mise en place du registre privé

1. Lancer un container basé sur l'image `registry:2` :

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

Création d'une image et stockage sur Docker Hub

1. Créer une image (avec un tag) :

```
docker build -t <host:port>/<your_tag> - <<EOF
  FROM busybox:latest
  CMD echo "Hello world!"
  EOF
```

2. Pusher l'image :

```
docker push <host:port>/<your_tag>
```

Les registres docker

Mettre en place un registry privé et y stocker ses images

Récupération d'une image depuis le registre privé

1. Supprimer les copies locales de l'image :

```
docker image rm <host:port>/<your_tag>
```

2. Récupérer l'image depuis Docker Hub

```
docker pull <host:port>/<your_tag>
```

3. Vérifier que l'image est présente localement :

```
docker image ls
```

Stopper et nettoyer le registre privé

1. Stopper le registre :

```
docker container stop registry
```

2. Supprimer le container et le volume associé

```
docker container rm -v registry
```