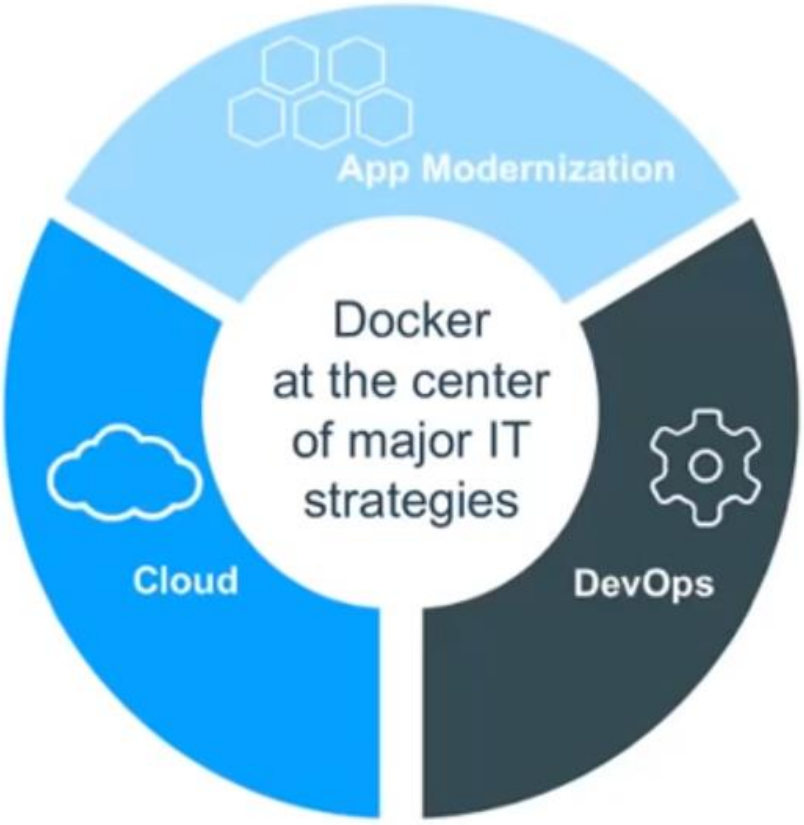
Les conteneurs avec Docker

1) Présentation de Docker

Docker est un produit développé par la société du même nom. Initialement développé par un ingénieur français, **Solomon Hykes**, le produit a été dévoilé en mars 2013. Depuis cette date, **Docker** est devenu un produit phare pour l'utilisation des conteneurs. Docker est un logiciel libre permettant facilement de **lancer des applications dans des conteneurs logiciels**. C'est un outil qui peut **empaqueter une application et ses dépendances dans un conteneur isolé**, qui pourra être exécuté sur n'importe quel serveur. Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prédictible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc.

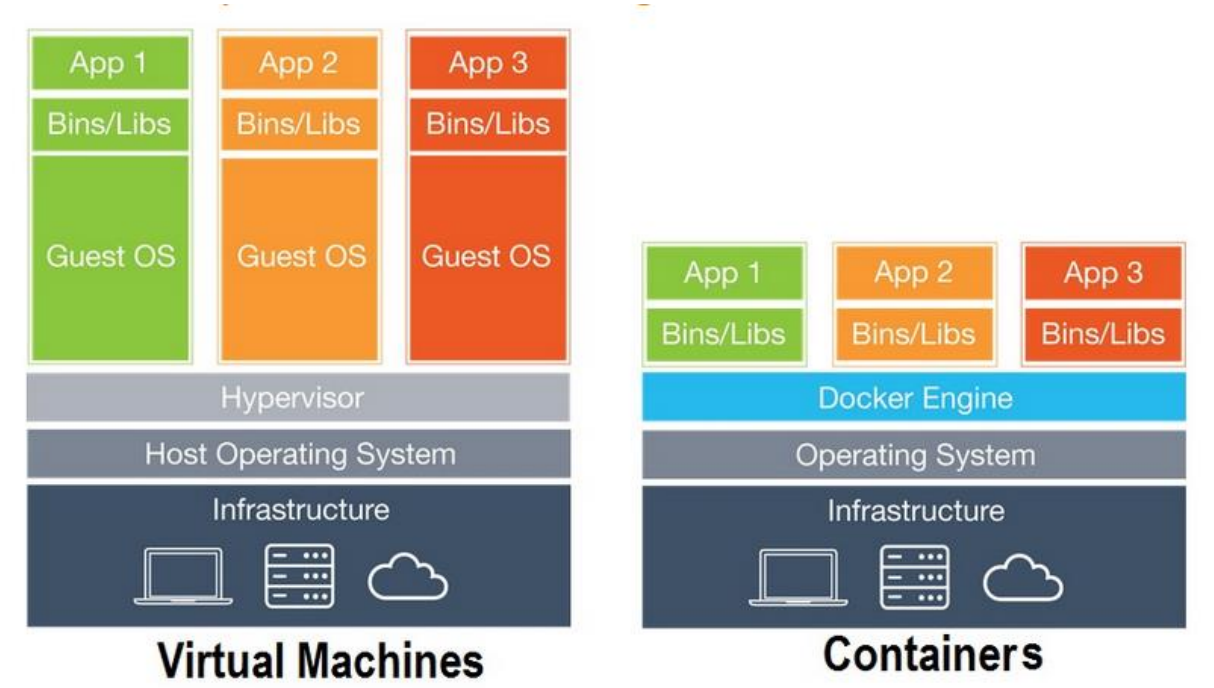


Docker permet de **créer des environnements (appelées conteneurs)** de manière à isoler des applications. L'idée est de lancer du code (ou d'exécuter une tâche) dans un environnement isolé. Docker repose sur le kernel Linux et étend le format de conteneur Linux standard LXC avec une API de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée. Outre **LXC**, Docker utilise **cgroups** qui va avoir pour objectif de **gérer les ressources (utilisation de la RAM, CPU entre autres)** et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d'exploitation fournies par la machine hôte.

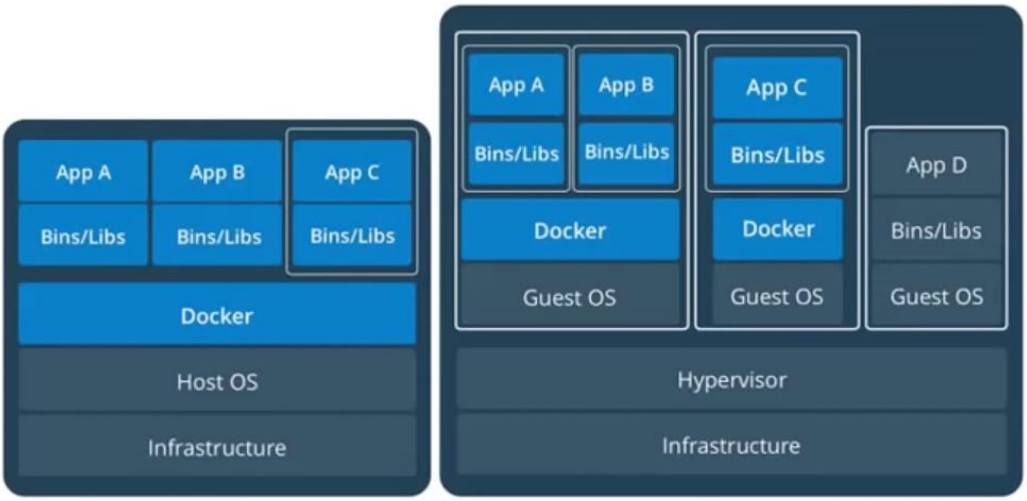
La technologie de conteneur de Docker peut être utilisée pour étendre des systèmes distribués de façon qu'ils s'exécutent de manière autonome depuis une seule machine physique ou une seule instance par nœud. Cela permet aux nœuds d'être déployés au fur et à mesure que les ressources sont disponibles, offrant un déploiement transparent et similaire aux PaaS pour des systèmes comme **Apache Cassandra, Riak** ou d'autres systèmes distribués.

Un conteneur permet d'isoler chaque service, le serveur web, la base de données, l'application qui peuvent être exécutés de façon indépendante dans leur conteneur dédié, contenant uniquement les dépendances nécessaires. Chaque conteneur est relié par des réseaux virtuels. Il est possible de monter des volumes de disque de la machine hôte dans un conteneur.

Une VM est un environnement de système d'exploitation ou d'application installé sur logiciel. Elle permet à l'utilisateur de profiter de **la même expérience que sur une machine physique**, avec plusieurs avantages.



Il est notamment possible de **lancer plusieurs environnements d'OS sur la même machine**, en les isolant les uns des autres. De même, la virtualisation permet de réduire les coûts au sein d'une entreprise en diminuant le nombre de machines virtuelles nécessaires. Les besoins en énergie s'en trouvent aussi atténués. Les backups et les restaurations s'en trouvent aussi simplifiés.



Cependant, les hyperviseurs de machines virtuelles reposent sur une émulation du hardware, et **requièrent donc beaucoup de puissance de calcul**. Pour remédier à ce problème, de nombreuses entreprises se tournent vers les conteneurs, et par extension vers Docker.

La **plateforme Docker** présente de nombreux avantages. Elle permet de composer, de créer, de déployer et d'échelonner rapidement des conteneurs sur les hôtes Docker. Elle permet également de **développer des applications de façon plus efficiente et efficace**, en utilisant moins de ressources.

Elle offre aussi **un haut degré de portabilité**, ce qui permet aux utilisateurs de s'enregistrer et de partager des conteneurs sur une large variété d'hôtes au sein d'environnements publics et privés.

Par rapport aux machines virtuelles, Docker présente également plusieurs avantages mais présente aussi plusieurs inconvénients. Il peut être **difficile de gérer de façon efficiente un grand nombre de conteneurs simultanément**. De plus, la sécurité devient un problème. Les conteneurs sont isolés, mais partagent le même système d'exploitation. De fait, une attaque ou une faille de sécurité sur l'OS peut compromettre tous les conteneurs. Pour minimiser ce risque, certaines entreprises exécutent leurs conteneurs au sein d'une machine virtuelle.

2) Les composants docker (les fonctionnalités docker)

La plateforme de conteneurisation repose sur **sept composants principaux**. Le **Docker Engine** est un outil client-serveur sur lequel repose la technologie de conteneur pour prendre en charge les tâches de création d'applications basées sur les conteneurs. Le moteur crée un processus daemon **server-side** permettant d'héberger les images, les conteneurs, les réseaux et les volumes de stockage. Ce daemon fournit aussi une interface **SLI client-side** permettant aux utilisateurs d'interagir avec le daemon via l'API de la plateforme.

Le **démon docker (daemon)** est un démon qui gère les conteneurs LXC qui s'exécutent sur la machine hôte.

La **docker CLI (Command Line Interface)** permet de taper une série de commandes utiles pour interagir avec le démon.

Docker images est une base d'images binaires qui peuvent être publiques ou privées comme par exemple CentOS ou Ubuntu.

Docker containers sont des répertoires contenant l'application.

Les **conteneurs créés sont appelés Dockerfiles**. Ces derniers sont des scripts qui automatisent la manipulation d'images.

Docker Compose est l'outil officiel proposé par Docker, Inc en ayant racheté **Fig**. Pour **Docker Compose**, un fichier **docker-compose.yml** doit être créé. Le composant Docker Compose permet de définir la composition des composants au sein d'un conteneur dédié. Chaque conteneur doit y être décrit : image, commande de lancement, volumes, ports, etc.

Le **Docker hub** est un outil SaaS permettant aux utilisateurs de publier et de partager des applications basées conteneur via une bibliothèque commune.

Le mode **Docker Swarm** du Docker Engine **prend en charge l'équilibrage des charges des clusters**. Ainsi, les ressources de plusieurs hôtes peuvent être rassemblées pour agir comme un seul ensemble. Ainsi, les utilisateurs peuvent rapidement échelonner le déploiement de conteneurs.

Docker n'est pas la seule plateforme de conteneurs sur le marché, mais elle demeure la plus utilisée. Son **principal concurrent est CoreOS rkt**. Cet outil est principalement réputé pour

sa sécurité, avec notamment le support de **SELinux**. Parmi les autres plateformes majeures, on compte **Canonical LXD**, ou encore **Virtuozzo OpenVZ**, la plus ancienne plateforme de conteneurs.

3) La manipulation de conteneurs

3.1) Les commandes de base

La commande suivante devrait vous permettre de télécharger la première fois une image légère CentOS à partir du Cloud (repository), la démarrer dans un conteneur avant de lancer une session Shell : **$ sudo docker run -i -t centos /bin/bash**

Conteneur interactif :

-i : utilisation interactive

-t : affichage sur terminal

$ sudo docker run -t -i centos /bin/bash

root@be7dbae53faa3:/#

root@be7dbae53faa3:/# **pwd**

/

root@be7dbae53faa3**:/# ls**

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

Pour quitter : **root@be7dbae53faa3:/# exit**

Un exemple de conteneur en arrière-plan :

Un conteneur lancé en **arrière-plan (-d)** qui lance un script. Cette commande retourne l'ID du conteneur.

$ sudo docker run -d centos /bin/sh -c "while true; do echo hello world; sleep 1; done"

1e696ebefdabcdeb654efbce414eeced55577acbeadcea7 is the container ID provided by the docker daemon

3.2) Quelques commandes de manipulation de conteneurs

Affiche la **version de Docker** et donne en plus la version du langage Go utilisé :

$ sudo docker version

Affiche la **liste des conteneurs** :

$ sudo docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1e5535038e28 ubuntu:14.04 /bin/sh -c 'while tr 2 minutes ago Up 1 minute moon_babbage

Affiche la **liste des conteneurs déjà lancés** : **$ sudo docker ps –l**

Affiche la **liste des conteneurs actifs** : **$ sudo docker ps**

Affiche **tous les conteneurs** y compris ceux qui sont arrêtés ou terminés : **$ sudo docker ps –a**

Affiche les **logs** : **$ sudo docker logs moon_babbage**

hello world

hello world

hello world

Arrête un conteneur : **$ sudo docker stop moon_babbage**

Le client Docker peut effectuer plusieurs actions :

$ sudo docker

attach : attache à un conteneur en exécution.

build : construit (build) une image à partir d'un Dockerfile.

commit : crée une nouvelle image à partir d'un conteneur modifié.

On peut passer l'option **--help**, comme par exemple :

$ sudo docker attach --help

Si on veut réaccéder à un conteneur, on commence par le relancer le conteneur en précisant son nom et ensuite on l'attache :

$ sudo docker start moon_babbage

$ sudo docker attach moon_babbage

3.3) La manipulation d'images binaires

La commande suivante affiche la liste des images binaires dont nous disposons en local :

$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

training/webapp latest fc77f57ad303 2 weeks ago 280.5 MB

ubuntu 13.10 5e019ab7bf6d 3 weeks ago 180 MB

ubuntu saucy 5e019ab7bf6d 4 weeks ago 180 MB

ubuntu 12.04 74fe38d11401 4 weeks ago 209.6 MB

On peut rechercher une image sur le Cloud (Docker repository) :

$ sudo docker search opensuse

Pour obtenir une nouvelle image, on peut en précharger une comme par exemple opensuse :

$ sudo docker pull opensuse

Pulling repository opensuse

b7de3133ff98: Pulling dependent layers

5cc9e91966f7: Pulling fs layer

511136ea3c5a: Download complete

ef52fb1fe610: Download complete

.

Ainsi lors de cette commande, il ne sera pas nécessaire de faire du téléchargement :

$ sudo docker run -t -i opensuse /bin/bash

bash-4.1#

3.4) La création d'un compte et un espace de stockage (repository ou Hub) sur le cloud : https://hub.docker.com/

On peut créer un compte en lançant la commande suivante :

$ sudo docker login

Un fichier. dockercfg contient les authentifications.

La création du repository personnel nécessite une adresse mail, un login (par exemple jeanmichel) et un mot de passe.

3.5) Voici la liste des instructions d'un Dockerfile :

1. **FROM # Pour choisir l'image sur laquelle on se base, toujours en premier**
2. **RUN # Permet d'exécuter une commande**
3. **CMD # Commande exécutée au démarrage du conteneur par défaut**
4. **EXPOSE # Ouvre un port**
5. **ENV # Permet d'éditer des variables d'environnement**
6. **ARG # Un peu comme ENV, mais seulement le temps de la construction de l'image**
7. **COPY # Permet de copier un fichier ou répertoire de l'hôte vers l'image**
8. **ADD # Permet de copier un fichier de l'hôte ou depuis une URL vers l'image, permet également de décompresser une image**
9. **LABEL # Des métadonnées utiles pour certains logiciels de gestion de conteneurs, comme rancher ou swarm**

10. ENTRYPOINT # Commande exécutée au démarrage du conteneur, non modifiable, utilisée pour package une commande

11. VOLUME # Crée une partition spécifique

12. WORKDIR # Permet de choisir le répertoire de travail

13. USER # Choisit l'utilisateur qui lance la commande du ENTRYPOINT ou du CMD

14. ONBUILD # Crée un step qui sera exécuté seulement si notre image est choisie comme base

15. HEALTHCHECK # Permet d'ajouter une commande pour vérifier le fonctionnement de votre conteneur

16. STOPSIGNAL # permet de choisir le [signal](http://man7.org/linux/man-pages/man7/signal.7.html) qui sera envoyé au conteneur lorsque vous ferez un docker container stop

Exemple de dockerfile :

```
FROM ubuntu

ENV APACHE_RUN_USER www-data

ENV APACHE_RUN_GROUP www-data

ENV APACHE_LOG_DIR /var/web/log/apache2

ENV APACHE_PID_FILE /var/run/apache2.pid

ENV APACHE_RUN_DIR /var/run/apache2

ENV APACHE_LOCK_DIR /var/lock/apache2

RUN export DEBIAN_FRONTEND=noninteractive && apt-get update && apt-get -y -q upgrade && apt-get -y -q install apache2

EXPOSE 80 443

CMD ["apache2ctl","-D","FOREGROUND"]
```

Un autre exemple de dockerfile :

```
FROM debian:jessie

ENV GID=991

ENV UID=991

ENV CONTACT=contact@domain.tld

ENV WEBROOT=/

ENV SECRET=e7c0b28877f7479fe6711720475dcbbd

ENV MAX_FILE_SIZE=10000000000


LABEL description="lutim based on debian" \

maintainer="xataz <https://github.com/xataz>"

RUN apt-get update && apt-get install -y --no-install-recommends --no-install-suggests perl

ca-certificates shared-mime-info perlmagick make gcc ca-certificates libssl-dev git

RUN cpan install Carton

RUN cd / && git clone https://git.framasoft.org/luc/lutim.git

RUN cd /lutim && carton install

VOLUME /lutim/files /data

EXPOSE 8181

COPY lutim.conf /lutim/lutim.conf

COPY startup /usr/bin/startup

RUN chmod +x /usr/bin/startup

CMD ["startup"]
```

Un exemple de fichier docker-compose.yml

```
version: '3.3'

services:
 cont:
  image: nginx:1.10.1
  environment:
   - APP_VERSION=1.0-SNAPSHOT
   - SITE_NAME=mySite
volumes:
  - ./mySite/conf:/etc/nginx/conf.d
  - data
ports:
  - 8081:80
  - 8443:443
labels:
  - 'com.example.version=1.0'
  - 'com.example.appname=myApp'
volumes:
 data:
```

4) Résumé des instructions Docker

4.1) La gestion des conteneurs

Exécuter en mode détaché un conteneur nommé **cont**, à partir de l'image **nginx** version **1.10.1**, mappé sur le port interne **80** en lui partageant un répertoire local.

sudo docker run -d --name cont -p 80:80 -v $(pwd)/myConf.d:/etc/nginx/conf.d nginx:1.10.1

Récupérer le fichier **myConf** à l'intérieur d'un conteneur nommé **cont** pour le copier en local.

sudo docker cp cont:/etc/nginx/conf.d/myConf ~

Exécuter un conteneur nommé **cont** en lui passant des variables d'environnement.

sudo docker run --env VERSION=1.0 --env WEBHOST=myhost --name cont nginx:1.10.1

Exécuter une commande au sein du conteneur **cont** en mode interactif.

sudo docker exec -ti cont $CMD

Afficher les processus de mon conteneur **cont**.

sudo docker top cont

Visualiser en continu les logs de mon conteneur **cont**.

sudo docker logs --follow cont

Envoyer les logs de mon conteneur **nginx** vers le serveur **greylog** en lui spécifiant le driver **gelf**.

sudo docker run –log-driver gelf –log-opt gelf-address=udp://1.2.3.4:12201 nginx:1.10.1

Supprimer les conteneurs arrêtés.

sudo docker rm $(docker ps -a -q --filter «status=created» --filter «status=exited» --filter «status=dead» --filter «status=paused»)

Inspecter certains champs du **JSON** qui décrit mon conteneur **cont**.

sudo docker container inspect –format= «{{json.State.Status}}» cont

Afficher tous les conteneurs en état **Running** dont le nom contient **cont**.

sudo docker ps --filter name=cont

Visualiser les changements opérés dans mon conteneur **cont**.

sudo docker diff cont

Détruire tous les conteneurs tournant actuellement.

sudo docker kill $(docker ps -q)

4.2) La gestion des images

Supprimer l'image **redis** même si elle est utilisée.

sudo docker rmi -f redis

Créer un tag stable pour l'image **myapp**.

sudo docker tag myapp:0.1.1 myapp:stable

Se connecter à une **registry** existante, avec mon **login**.

sudo docker login -u jmm@corp.com registry.corp:5000

Récupérer une image depuis ma **registry**.

sudo docker pull registry.corp/myapp:stable

Publier une image dans ma **registry**.

sudo docker push registry.corp/myapp:0.1.2

Nettoyer les images **de plus de 24 heures** qu'elles soient utilisées ou non par des conteneurs.

sudo docker image prune --all –filter «until=24h»

Nettoyer les conteneurs **arrêtés/images obsolètes/volumes inutilisés**.

sudo docker container prune

sudo docker image prune

sudo docker volume prune

Sauvegarder l'image **cont** sous forme de **tarball**, pour la transmettre sans passer par une **registry** puis l'importer sur la machine d'un partenaire.

sudo docker save -o cont.tar cont

sudo docker load -i cont.tar

4.3) Docker Compose

Exécuter en mode détaché une stack de conteneurs depuis une configuration **docker-compose** en s'assurant que les conteneurs seront recréés même s'ils sont inchangés.

sudo docker-compose up --force-recreate -d

Valider la configuration du fichier **docker-compose-dev.yml.**

sudo docker-compose --file ./docker-compose-dev.yml config

Détruire les conteneurs d'une stack et **nettoyer** tous les volumes associés.

sudo docker-compose down –volumes

Construire les images de mon **docker-compose** tout en supprimant les conteneurs intermédiaires construits lors du **build**.

sudo docker-compose build –force-rm

Stopper les conteneurs d'une stack sans les supprimer.

sudo docker-compose stop