BD
INGÉNIERIE DES SI

Master 1 MIAGE – Année 2020/2021

Programme

- Module de 35h
  - 15h de cours
  - 20h de pratique
  - Pré-requis : maîtrise de SQL / BDr

- Plan
  - Intro : Docker
  - NoSQL : exemple de MongoDB
  - BD Stream : exemple de Kafka

Sommaire du module

1. Outils pour les Tps : Docker
2. La suite : NoSQL / SQL

VIRTUALISATION ET CONTENEURISATION

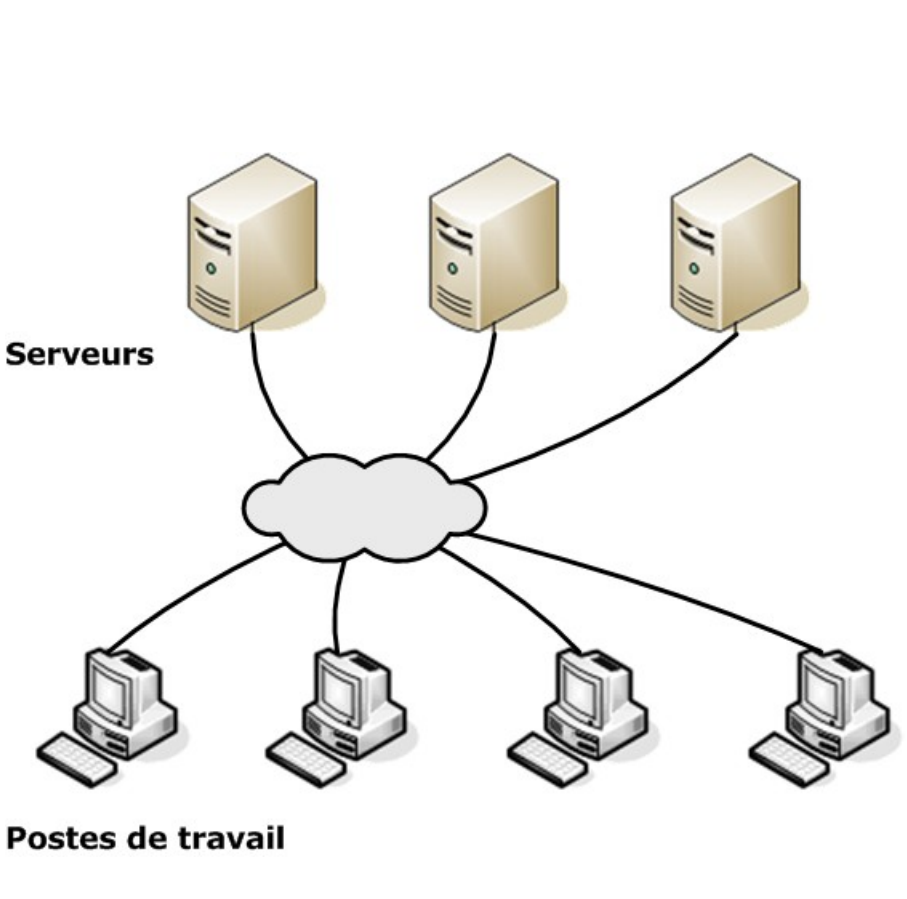
Cours 1

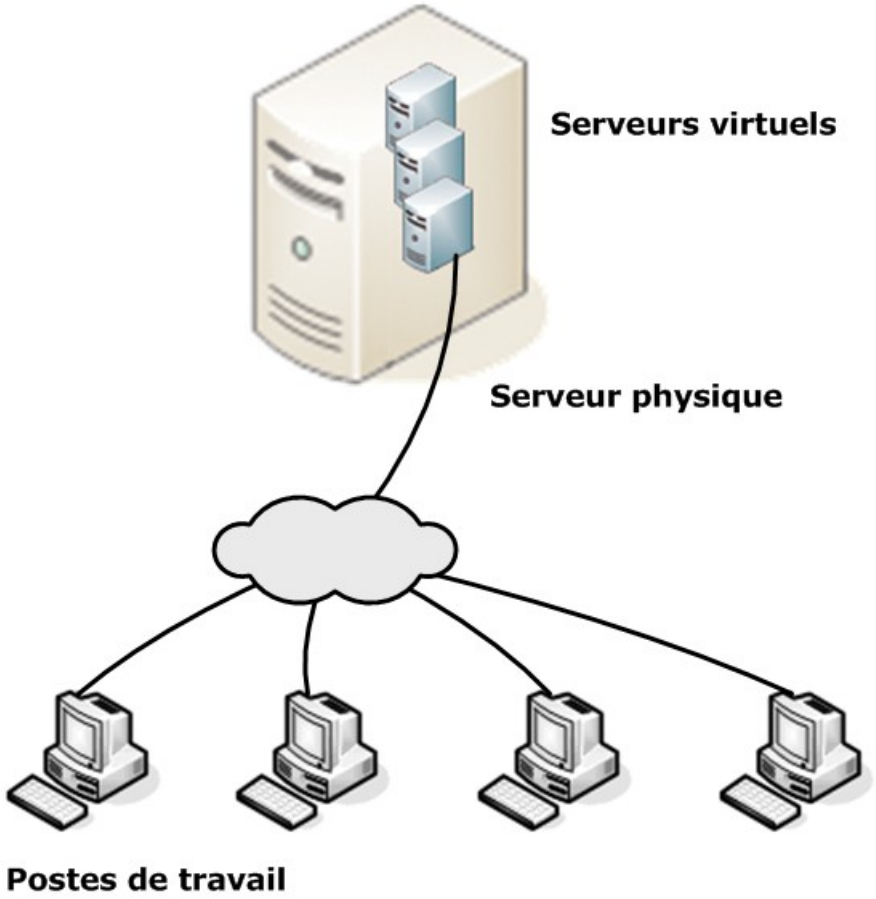
Master 1 MIAGE – BD&ISI

Partage d’un serveur

- Un serveur possède des ressources matérielles (CPU, mémoire, disques, réseau…) utilisées par les applications en utilisant un système d’exploitation
- Virtualisation : ensemble de techniques et d’outils permettant d’exécuter plusieurs systèmes d’exploitations sur un même serveur physique
- Le principe est donc de partager les ressources d’un serveur

Architectures





Objectifs

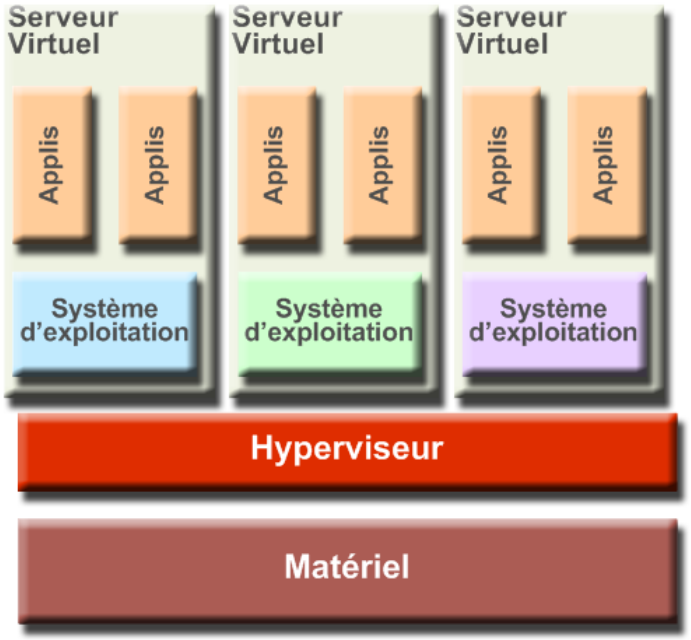
- économique : moins de serveurs physiques
- sous-utilisation : 10% -> 35%
- élasticité dans les besoins (eg Noël site marchand)
- facilité de maintenance et de sauvegarde des serveurs virtualisées
- répartition dynamique sur un parc de serveur : répartition de charge aisée

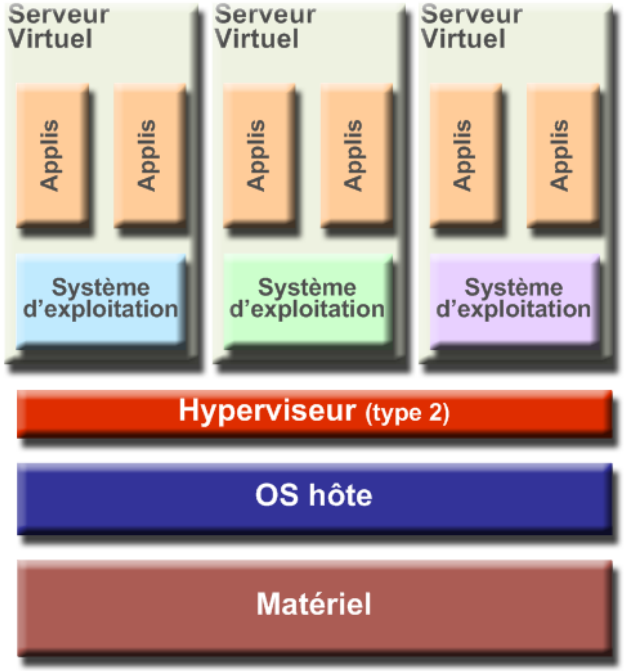
Historique

- Multi-users et contextes dans les années 70 (CICS années 80)
- IBM précurseur dès les années 60 avec CM/CMS, maintenant z/VM permettant d’exécuter AIX et Linux
- 1990 : micro-informatique, avec émulateurs (Atari, Amiga, Consoles)
- Fin 90, Vmware virtualisation sur x86
- Solutions Open Source Qemu, Xen, KVM…
- 2006, AMD et Intel ajoutent des instructions spécifiques sur x86 pour améliorer les perfs

Vocabulaire

- Hyperviseur : couche logicielle qui s'insère entre le matériel et les différents OS
- Gère lui-même le matériel ou s'appuie sur un OS existant





Différentes techniques

- Isolation

  Mise en place sur un même noyau d'une séparation forte entre les contextes logiciels (LXC, Docker)

- Paravirtualisation

  Présente aux OS une machine générique spéciale, intégrée aux OS invités sous forme de drivers ou de modif du noyau

- Virtualisation complète

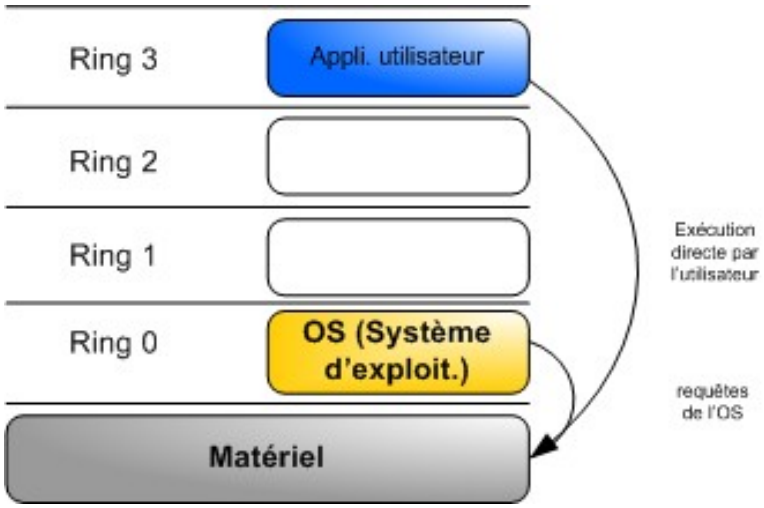
  Hyperviseur intercepte de façon transparente tous les appels des OS invités (donc non modifiés)

- Partitionnement matériel

  Séparation des ressources au niveau de la carte mère (eg SUN)

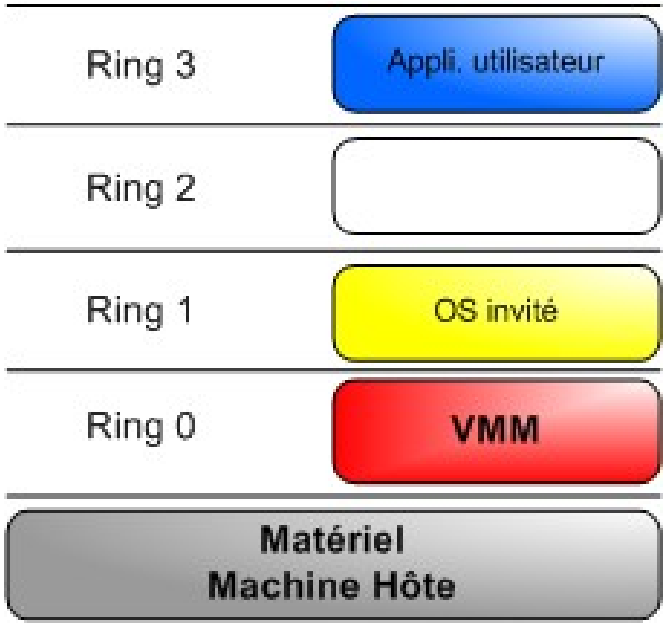
Comment ? Sur x86

4 niveaux de protection sur une machine physique



- **Ring 3** : niveau le plus bas attribué aux applications de l'utilisateur
- **Ring 2** : généralement non utilisé
- **Ring 1** : généralement non utilisé
- **Ring 0** : niveau le plus élevé réservé au Système d'Exploitation pour exécuter les instructions privilégiées

X86 ? **abaissement de privilège**



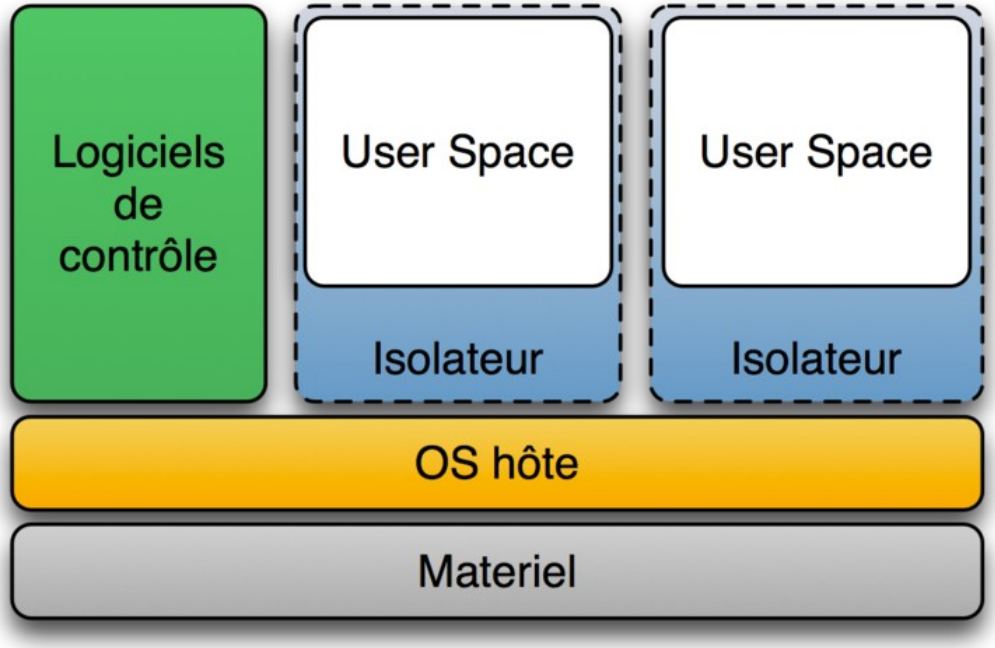
- **Ring 3** : niveau le plus bas toujours attribué aux applications de l'utilisateur
- **Ring 2** : généralement non utilisé
- **Ring 1** : niveau inférieur dans lequel est relégué **à son insu** le Système d'Exploitation de la machine virtuelle : abaissement
- **Ring 0** : niveau le plus élevé occupé par l'hyperviseur (VMM) qui intercepte les instructions critiques privilégiées de la VM et les émule (Trap & Emulate)

Isolation (cloisonnement)

- Un seul noyau d’OS
- Séparation en plusieurs contextes ou environnements
- Les processus de chaque contexte ne peuvent voir/communiquer qu’avec ceux du même contexte
- Utilisé depuis longtemps sous Unix (chroot ou jail) pour limiter les ressources accessibles aux process ; possibilité d’avoir plusieurs distrib (si même noyau)
- Solution légère, très utilisée pour avoir différents environnements (eg dev/tests/prod)
- Mais robustesse moindre et limitations sur les OS possibles car un seul noyau

Isolation (cloisonnement)

- Sous Linux, solutions principales :
  - OpenVZ : mature et fonctionnalités
  - LXC : relativement jeune mais dynamique
  - Docker : en plein essor, n’utilise plus LXC



Pourquoi ???

Des opportunités pour vous ?

- Coté ops :
  - Meilleure utilisation des ressources (serveur, place, électricité, etc)
  - Installation, déploiement et migration facilités
  - Isolation (sécurité, modification locale, etc)
- Coté dev :
  - Disposer de plusieurs environnements de développement
  - Tester son code dans un environnent normalisé
  - Simuler la production

Eviter le « Ben chez moi ça marche » !

Plateforme de validation et dev

- Test de la compatibilité d'une appli avec de nombreuses configs

  => environnements virtualisés

  - Avec émulateur, test possible sur des processeurs autres que ceux des machines physiques
- Administration/installation de plateformes de dev/intégration/recette/…

  eg SSII, mise en place de l'environnement des clients

  eg dev java+mysql, prod java+oracle,…

Docker

Les origines

- Chroot 1979
- Bsd jail 2000
- Vserver 2001-2008
- Linux Namespaces 2002
- Solaris zones 2005
- OnpenVZ 2005
- Process containers 2006 (cgroups)
- LXC 2008
- Docker 2013
- ... Open Container Initiative ...

Docker : historique

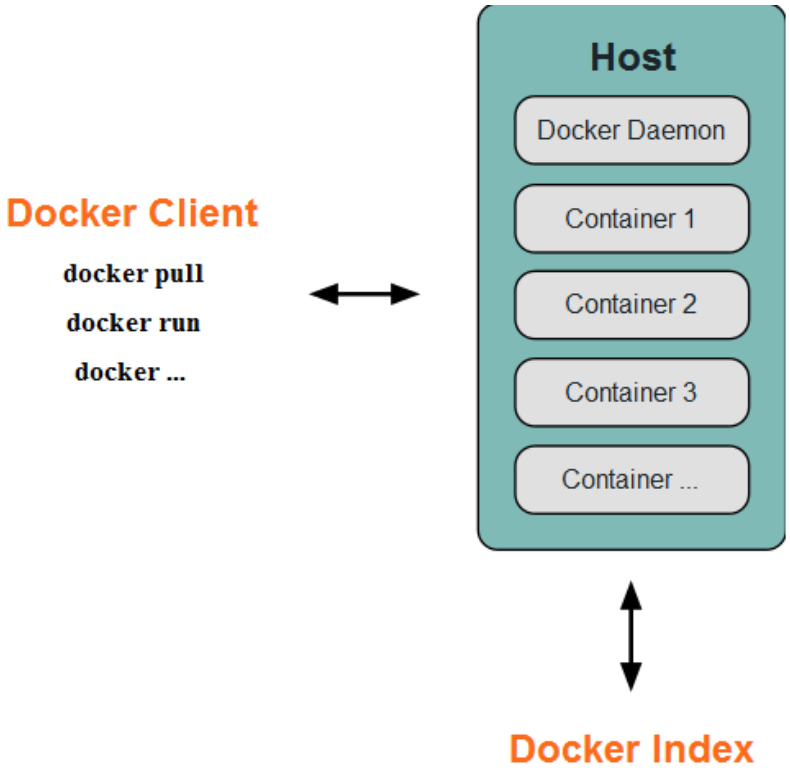
- Initialement projet interne d’un fournisseur cloud (dotCloud)
- Distribué en projet open source à partir de mars 2013
- V0 en mars 2013
- Utilise LXC au début ; maintenant son propre système d’isolation
- V1.7.0 en juin 2015
- V18.03 current

Les conteneurs

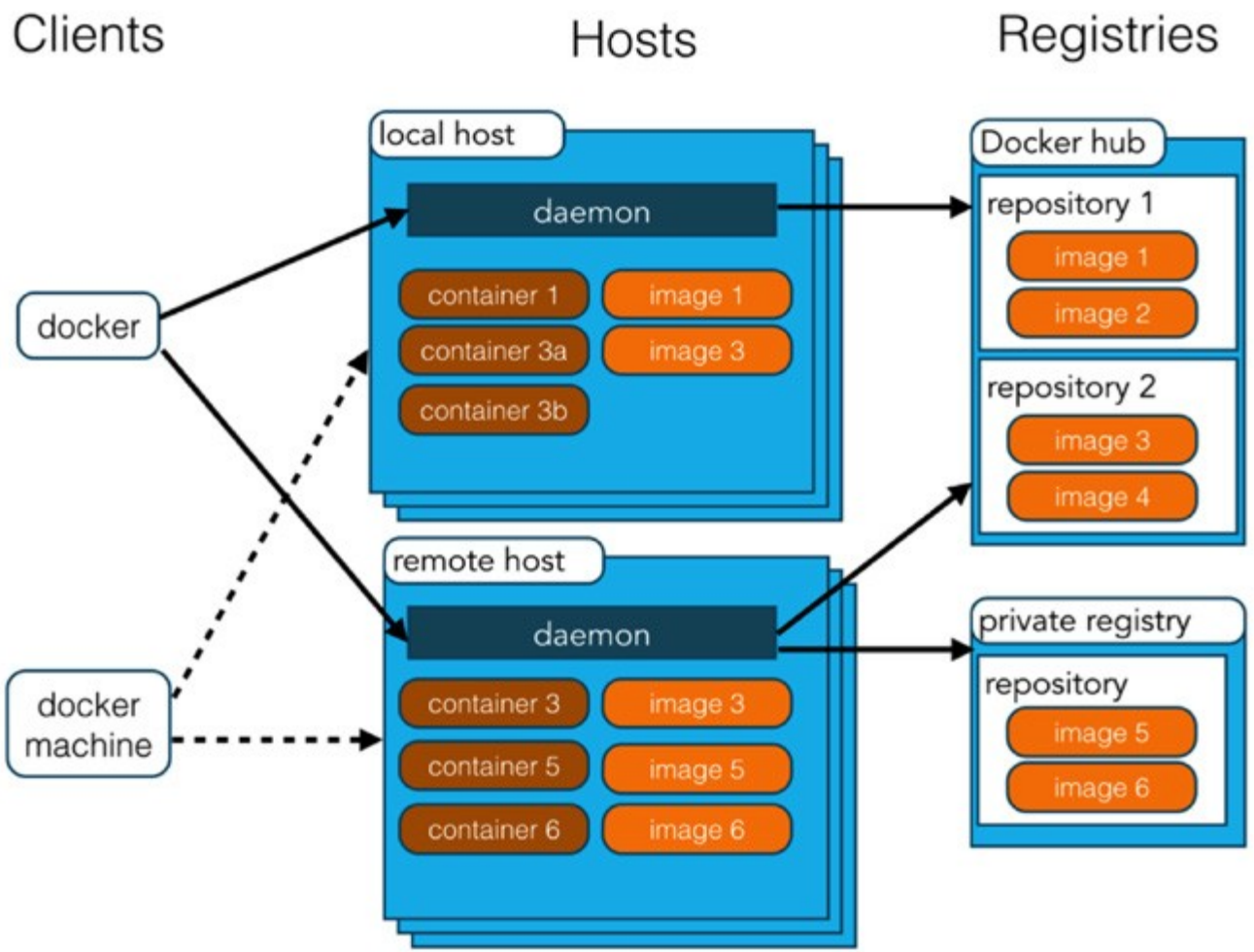
- Les conteneurs fournissent un environnement isolé sur un système hôte, semblable à un chroot sous Linux ou une jail sous BSD, mais en proposant plus de fonctionnalités en matière d'isolation et de configuration
- Ces fonctionnalités sont dépendantes du système hôte et notamment du kernel

Docker : caractéristiques

- Isolation par conteneurs reposant sur les namespaces et cgroups (control groups) du noyau Linux



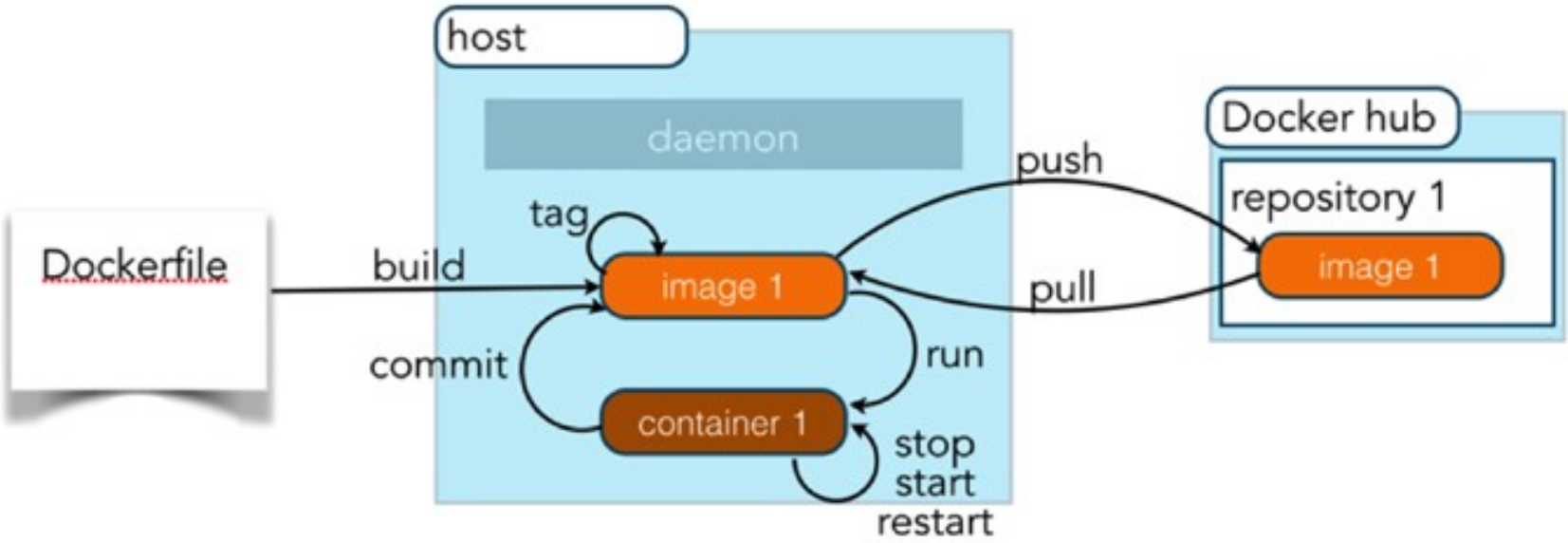
Docker : concepts



Docker : concepts

Docker	Définition
Images	Un template permettant de créer des containers
Container	Une machine virtuelle « ligth » crée à partir d'une image
Client	Utilitaire en ligne de commande (ou autre) permettant d'accèder à l'API du docker daemon
Host	Machine physique ou virtuelle qui execute le daemon docker et qui contient des images et des containers

Docker : workflow



Docker : commandes

- Docs en ligne : https://docs.docker.com/
- Toutes les commandes docker sont lancées en ligne de commande par :

```
docker COMMANDE [-options] PARAMETRES
```

- Sous Windows/OSX : voir docker machine ou docker pour windows/mac

- !!! Le man :

```
docker COMMANDE --help
```

Docker : commandes

attach Attach to a running container
build Build an image from a Dockerfile
commit Create a new image from a container's changes
cp Copy files/folders between a container and the local filesystem
create Create a new container
diff Inspect changes on a container's filesystem
events Get real time events from the server
exec Run a command in a running container
export Export a container's filesystem as a tar archive
history Show the history of an image
images List images
import Import the contents from a tarball to create a filesystem image
info Display system-wide information
inspect Return low-level information on a container, image or task
kill Kill one or more running containers
load Load an image from a tar archive or STDIN
login Log in to a Docker registry.
logout Log out from a Docker registry.
logs Fetch the logs of a container
network Manage Docker networks
node Manage Docker Swarm nodes
pause Pause all processes within one or more containers
port List port mappings or a specific mapping for the container
ps List containers
pull Pull an image or a repository from a registry
push Push an image or a repository to a registry
rename Rename a container
restart Restart a container
rm Remove one or more containers
rmi Remove one or more images
run Run a command in a new container
save Save one or more images to a tar archive (streamed to STDOUT by default)
search Search the Docker Hub for images
service Manage Docker services
start Start one or more stopped containers
stats Display a live stream of container(s) resource usage statistics
stop Stop one or more running containers
swarm Manage Docker Swarm
tag Tag an image into a repository
top Display the running processes of a container
unpause Unpause all processes within one or more containers
update Update configuration of one or more containers
version Show the Docker version information
volume Manage Docker volumes
wait Block until a container stops, then print its exit code

Docker : gestion des images

- 3 manières d’obtenir une image :
  - Récupérer localement une image [publique] sur Docker Hub
  - Sauver l’état d’un conteneur dans une image locale
  - Définir une image par un DockerFile
  - [importer une image exportée en .tar]

Docker : gestion des images

- Afficher les images locales disponibles :

  `docker images`

- Suppression d'une image locale :

  `docker rmi IMAGE`

- Nommage d'image locale :

  `docker tag IMAGE nom`

```
docker tag monmysql fred/monmysql:1.0
```

Docker : gestion des images

- Recherche d'une image sur le hub public docker :

  `docker search debian`

  - Attention, n'importe qui peut envoyer des images sur ce hub (format usuel user/image:tag) ; les images officielles sont sans le user/
  - Indispensable : https://hub.docker.com
- Téléchargement sur le disque local d'une image sur le hub public docker :

  `docker pull debian:wheezy`

  - si vous ne donnez pas de tag, c'est lastest qui est choisi [Attention!]
- Attention, la commande run recherche et télécharge une image sur docker hub s’il la trouve

Docker : créer vos propres images

- Commit d'un conteneur (run ou pause) dans lequel vous avez fait des modifs

```
docker run –i –t --name demo debian
      apt-get update
      apt-get install –y git
…
docker commit -m "Added git" -a "Fred Moal" demo fred/git:v2
```

Docker : créer vos propres images

- Description de votre image par un fichier Dockerfile

```
# This is a comment
FROM ubuntu:14.04
MAINTAINER Kate Smith <ksmith@example.com>
RUN apt-get update && apt-get install -y ruby ruby-dev
RUN gem install sinatra
```

Puis lancement du build :

```
docker build -t ouruser/sinatra:v2 .
```

Docker : créer vos propres images

- Envoi sur Docker Hub (GitHub pour Git) :

```
docker login
```

```
        Login with your Docker ID to push and pull images from
    Docker Hub. If you don't have a Docker ID, head over to
    https://hub.docker.com to create one.
    Username:
```

```
docker push ouruser/sinatra
```

Docker : gestion des images

- chargement d'une image sauvée par la commande save dans un fichier local (au format tar) :

```
docker load -i /tmp/imagesave.tar
```

Docker : gestion des conteneurs

- Lancement d'une image :

  `docker run -i -t --rm -p 8080:9000 debian:wheezy /bin/bash`

  De TRES nombreuses options, cf doc docker

  En particulier –d –v –p –rm –t –i …

- Liste des conteneurs en cours d'exe :

  `docker ps`

- Liste de tous les conteneurs :

  `docker ps –a`

Docker : gestion des conteneurs

- Arrêt / (re)-Démarrage d'un conteneur :

```
docker stop ID/NOM
docker start ID/NOM
```

- Exécution d'une commande dans un conteneur :

```
docker exec ID/NOM COMMANDE
```

- Liste des ports :

```
docker port ID/NOM
```

- Liste des process :

```
docker top ID
```

- Suppression d'un conteneur :

```
docker rm ID
```

Docker : partages

- De ports entre l'hôte et le conteneur :

```
docker run –p 80:8080 …
```

- De répertoires pour les fichiers :

```
docker run -d -P --name web -v
/src/webapp:/opt/webapp
```

- Entre conteneurs : nommage et link

```
docker run -d --name db mysql:latest
docker run -d -P --name web --link db:db tomcat
```

Docker machine

- Sous win 7/8 ou vieux OSX ou …
- Commandes lancées dans GitBash :
  - docker-machine create –d virtualbox mamachine
  - docker-machine ls
  - docker-machine start mamachine
  - docker-machine ssh
  - docker-machine stop
- Une machine default créée à l’install
- Upgrade pour mettre à jour docker

Docker ++

Questions...

- Intérêts de Docker vs Virtualisation
- Inconvénients de Docker vs Virtualisation

Dockerfile : détails

- Série d’instructions pour construire une image
- FROM : baseimage utilisée
- RUN : Commandes effectuées lors du build de l'image
- EXPOSE : Ports exposées lors du run (si -P est précisé)
- ENV : Variables d'environnement du conteneur à l'instanciation
- CMD : Commande unique lancée par le conteneur
- ENTRYPOINT : "Préfixe" de la commande unique lancée par le conteneur

Dockerfile

```
FROM debian
RUN apt-get update && apt-get install -y curl
```

- docker build : générer une image depuis un Dockerfile

```
mkdir /tmp/mycurl
vim /tmp/mycurl/Dockerfile
cd /tmp/mycurl/
docker build -t fred/mycurlimg .
```

- Génération de conteneurs intermédiaires à chaque RUN, commit en images, puis suppression à la fin
- Donc on évite RUN apt-get update RUN apt-get install…

Dockerfile

- History des images :

```
docker history fred/mycurlimg
IMAGE               CREATED             CREATED BY                                      SIZE
b8d5d64ce694        25 seconds ago      /bin/sh -c apt-get update && apt-get install    26.77 MB
1b088884749b        3 months ago        /bin/sh -c #(nop) CMD ["/bin/bash"]             0 B
<missing>           3 months ago        /bin/sh -c #(nop) ADD file:76679eeb94129df23c   125.1 MB
```

```
docker history debian
IMAGE               CREATED             CREATED BY                                      SIZE
1b088884749b        3 months ago        /bin/sh -c #(nop) CMD ["/bin/bash"]             0 B
<missing>           3 months ago        /bin/sh -c #(nop) ADD file:76679eeb94129df23c   125.1 MB
```

Dockerfile : CMD

- L’instruction peut-être placée n’importe où dans le Dockerfile, la dernière CMD du Dockerfile aura précédence,
- Plusieurs CMD sont inutiles,
- On place la CMD en fin de Dockerfile

```
FROM debian
RUN apt-get install -y curl
CMD curl http://httpbin.org/ip
```

- Rem : docker build --no-cache -t fred/mycurlimg .
- docker run -it --rm fred/mycurlimg

```
{
  "origin": "193.49.83.102"
}
```

Dockerfile : ENTRYPOINT

- Limitation de CMD à une commande
- ENTRYPOINT définit la commande de base (et ses paramètres) d'un conteneur :
  - Force la commande du conteneur
  - Les paramètres de la commande docker run sont ajoutés à cette ENTRYPOINT
  - ENTRYPOINT peut/doit être placé comme CMD

Dockerfile : ENTRYPOINT

```
FROM debian
RUN apt-get install -y curl
ENTRYPOINT ["curl"]
```

- Lancement avec paramètres :

```
docker run -it fred/mycurlimg
   http://httpbin.org/user-agent
   {
     "user-agent": "curl/7.38.0"
   }
```

Dockerfile : défaut

- On peut utiliser ENTRYPOINT et CMD ensemble
  - ENTRYPOINT définit la commande de base à exécuter
  - CMD définit les paramètres par défaut :
- si aucun paramètre n’es passé à docker run, CMD sera utilisé par défaut
- sinon, ces paramètres remplacent CMD

```
FROM debian
RUN apt-get install -y curl
ENTRYPOINT ["curl"]
CMD ["--help"]
```

Dockerfile

- Et mon application php/java/ada… alors ??
- ADD dans Dockerfile : copie d’un ficher dans l’image du container [en général .tgz]
- COPY : copie d’un répertoire

- Exercice : affichez php info d’un conteneur

Docker : réseau

- Dans le conteneur :

```
root@594223dd8f2b:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1200 errors:0 dropped:0 overruns:0 frame:0
          TX packets:422 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:10091769 (9.6 MiB)  TX bytes:25963 (25.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Docker : réseau

- En dehors :

```
docker0   Link encap:Ethernet  HWaddr 02:42:7F:08:A0:21
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:7fff:fe08:a021/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2766 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7441 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:170234 (166.2 KiB)  TX bytes:63498389 (60.5 MiB)

...
veth5d5965c Link encap:Ethernet  HWaddr 02:58:FA:88:34:19
          inet6 addr: fe80::58:faff:fe88:3419/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:25218 (24.6 KiB)  TX bytes:3294 (3.2 KiB)
```

Docker : réseau

- docker inspect -f '{{.NetworkSettings.Gateway}}' ID

172.17.0.1

- docker inspect -f '{{.NetworkSettings.IPAddress}}' ID

172.17.0.2

- docker ps

PORTS 0.0.0.0:8080->80/tcp

- docker port ID
- Fixés par docker run (-P ou –p local:conteneur) ou EXPOSE dans Dockerfile

Docker : volumes

- En run par -v ou VOLUME
- Partage entre conteneurs ou avec l’hôte
- --volumes-from ID dans run autre conteneur
- Exemple :

```
docker run -d -v /var/log --name nginx nginx
docker run -it --rm --volumes-from nginx debian
root@e06119ac7d89:/# ls /var/log/nginx/
access.log  error.log
```

- -v /rep/hote:/rep/conteneur pour l’hôte

Docker : liens entre conteneurs

- Problème : un conteneur php, un mysql … Comment les lier ?
- Utilisation de variables d'env avec run -e
- Option --link au run
- Exemple :

```
docker run -d -P --name mysql -e \
   MYSQL_ROOT_PASSWORD=secret mysql
docker run -it --rm --link mysql:bd debian
```

Docker : liens entre conteneurs

root@ceb7bddec892:/# env

```
BD_NAME=/happy_spence/bd
BD_PORT_3306_TCP_PROTO=tcp
HOSTNAME=ceb7bddec892
TERM=xterm
BD_PORT_3306_TCP_PORT=3306
BD_ENV_MYSQL_MAJOR=5.7
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
BD_ENV_GOSU_VERSION=1.7
BD_PORT_3306_TCP_ADDR=172.17.0.4
BD_PORT_3306_TCP=tcp://172.17.0.4:3306
BD_PORT=tcp://172.17.0.4:3306
SHLVL=1
HOME=/root
BD_ENV_MYSQL_ROOT_PASSWORD=secret
BD_ENV_MYSQL_VERSION=5.7.12-1debian8
```

root@ceb7bddec892:/# ping bd

```
PING bd (172.17.0.4): 56 data bytes
64 bytes from 172.17.0.4: icmp_seq=0 ttl=64 time=0.486 ms
--- bd ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.486/0.486/0.486/0.000 ms
```

Docker : liens entre conteneurs

- Donc :

```
docker run -it --link mysql:server --rm mysql sh -c \
   'exec mysql -h"$SERVER_PORT_3306_TCP_ADDR"  -
  P"$SERVER_PORT_3306_TCP_PORT" \
   -uroot -p"$SERVER_ENV_MYSQL_ROOT_PASSWORD" ‘
```

```
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.12 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

Exercices

- Lancez un wordpress

- Déployez votre application php sous Apache ou Nginx
  - En manuel
  - Depuis IntellijJ

Orchestration

Orchestration ?

- Euh les run -------- il faut les taper à chaque fois ???

- Simple : liens entre conteneurs
- Complexe : connection, coordination et ordonancement de conteneurs répartis sur plusieurs hôtes

Outils d’orchestration

- Outillage pour gérer plusieurs conteneurs
  - Compose : fichier yaml de config de services (=conteneurs)
  - Machine : provisionning de machines (locales+cloud)
  - Swarm : cluster d’hôtes vu comme un hôte
  - Kubernetes [Google] : swarm++, plusieurs technos
  - Mesos [Apache] : swarm++
  - CoreOS + Fleet + etcd
  - …

Compose

- Outil en CLI(python).
- Outil pour définir et gérer des applications multi-conteneurs sous Docker
- Vous définissez votre application dans un fichier unique
- Puis vous lancez une simple commande qui s’occupe de toute la mise en place de l’environnement d’exécution des conteneurs de votre application
  - conteneur
  - env
  - volume
  - lien
- Conteneurs inter-dépendants
- Conteneurs gardent leur isolation
- Notion de services

Compose

- Définir les services de son application dans un fichier YAML
- Lancer docker-compose up pour démarrer son application
- Utiliser docker-compose pour gérer son application : ps, stop, rm, ... Commandes équivalentes à docker

Compose

- Exemple : stack wordpress avec 2 services

```
wp :
  image : wordpress
  links :
    - db :mysql
  ports :
    - 8080 :80

db :
  image : mysql
  environment :
    MYSQL_ROOT_PASSWORD : example
```

Compose

- Dans le dossier qui contient le fichier de configuration docker-compose.yml, on lance :

  $ docker-compose up

- Compose va
  - Configurer l’environnement des conteneurs
  - Lancer les conteneurs
  - Assembler les logs des conteneur en un seul flux

Compose

- Les commandes :
  - docker-compose up -d
  - docker-compose ps
  - docker-compose ps db
  - docker-compose logs
  - docker-compose logs wp
  - docker-compose scale db=5
  - docker-compose ps
  - docker-compose scale db=2
  - docker-compose stop
  - docker-compose rm
- Voir doc en ligne !
- Exercice : déployez une stack LAMP avec compose

Machine

- Création d’hôtes Docker
- Plusieurs pilotes disponibles :
  - locaux
  - distants
  - cloud
- Configuration du client docker
- Gestion de Windows et Mac comme clients
- Outil en CLI : docker-machine

Machine

- création d’une machine :
  - docker-machine create --driver virtualbox host1
  - docker-machine create --driver virtualbox host2

- Créer un dossier de configuration pour notre machine(~/.docker/machine/host1)
- Créer une machine (virtuelle, locale, …)
- D’y installer Docker
  - VirtualBox : Boot2Docker (~/.docker/machine/iso)
  - Cloud : Ubuntu 12.04
- De configurer les clés ssh pour utiliser notre machine(~/.docker/machine/host1/ssh)

Machine

- docker-machine ls

```
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER    ERRORS
default   -        virtualbox   Running   tcp://192.168.99.100:2376           v1.12.1
host2     -        virtualbox   Running   tcp://192.168.99.101:2376           v1.12.1
```

- docker-machine inspect host1
- docker-machine ip host1
- docker-machine ssh host1

Machine

- Changement d’hôte depuis client :

```
docker-machine env host1
eval "$(docker-machine env host1)"
env | grep -i docker
docker info
docker-machine ls
docker-machine active
```

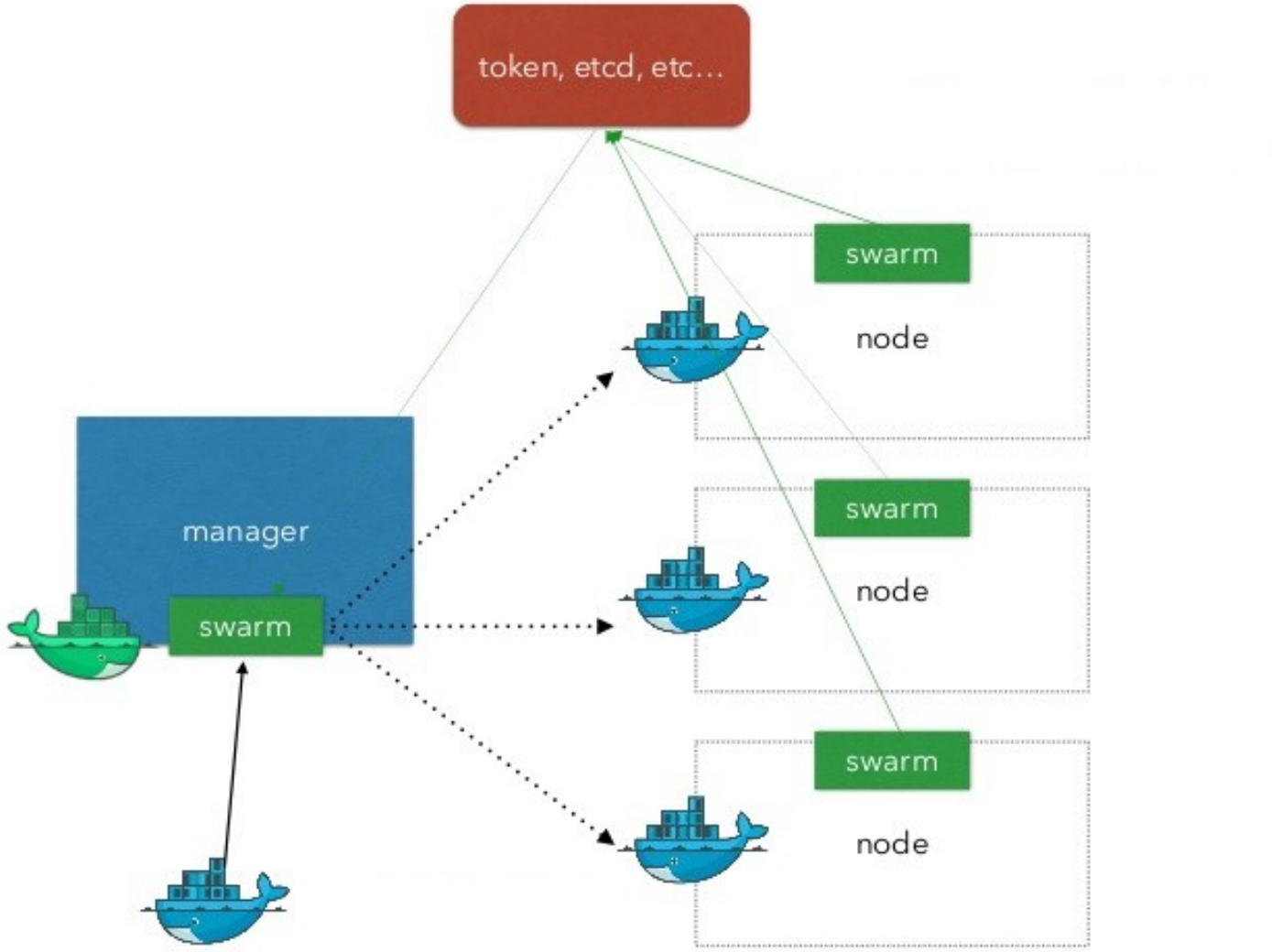
- Puis utilisation normale du client docker et docker-compose

```
eval "$(docker-machine env host2)"
docker-compose up -d
```

Swarm

- Sur un cluster d’hôtes Docker permet de lancer des conteneurs sans avoir à préciser quel hôte utiliser
- Docker Swarm consolide un ensemble d’hôtes Docker en un hôte virtuel, et permet d’utiliser le client Docker ou Compose comme si vous n’aviez qu’un hôte
- Composé de trois éléments :
  - Un maître
  - Des agents
  - Un service de découverte

Swarm


token, etcd, etc…
swarm
node
swarm
node
swarm
node
manager
swarm

Swarm

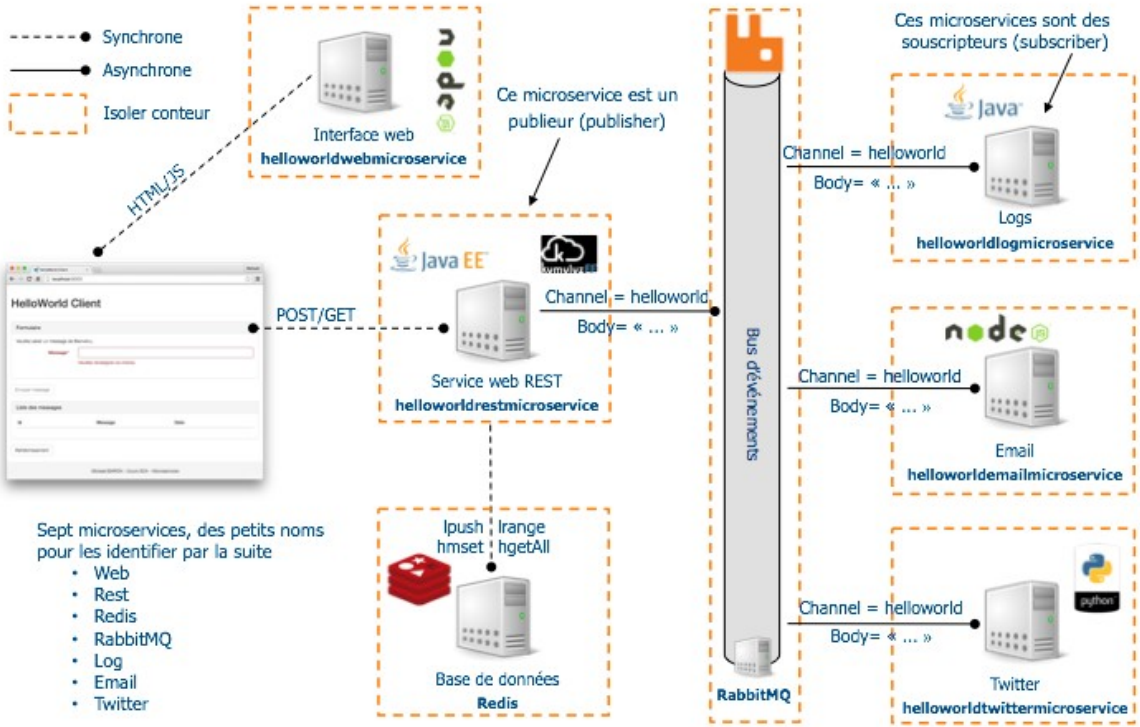
- Exercice :

Vérifiez la version de docker installée, et lancez un cluster avec 2 MariaDB + 2 php et un frontal Nginx sur 2 hôtes.

Montez à 5 php pour supportez un pic de charge puis descendez à 2

Bonus

- Déployez votre application phpinfo dans le cloud google (ou amazon) à l'aide de conteneur(s)
- Déployez un Helloworld avec cette architecture :



Bonus

- Aide :

http://mbaron.developpez.com/tutoriels/microservices/developpement-application-docker/