Machines Virtuelles

Guillaume Urvoy-Keller

November 23, 2022

Rôle de l'hyperviseur

- Offrir à l'OS invité (de la VM) un environnement virtuel tel que l'OS invité s'exécute **sans** modification.
- Assurer une correspondance efficace entre CPU/RAM/périphériques virtuel(le)(s) et physique(s)

Virtualisation de la CPU

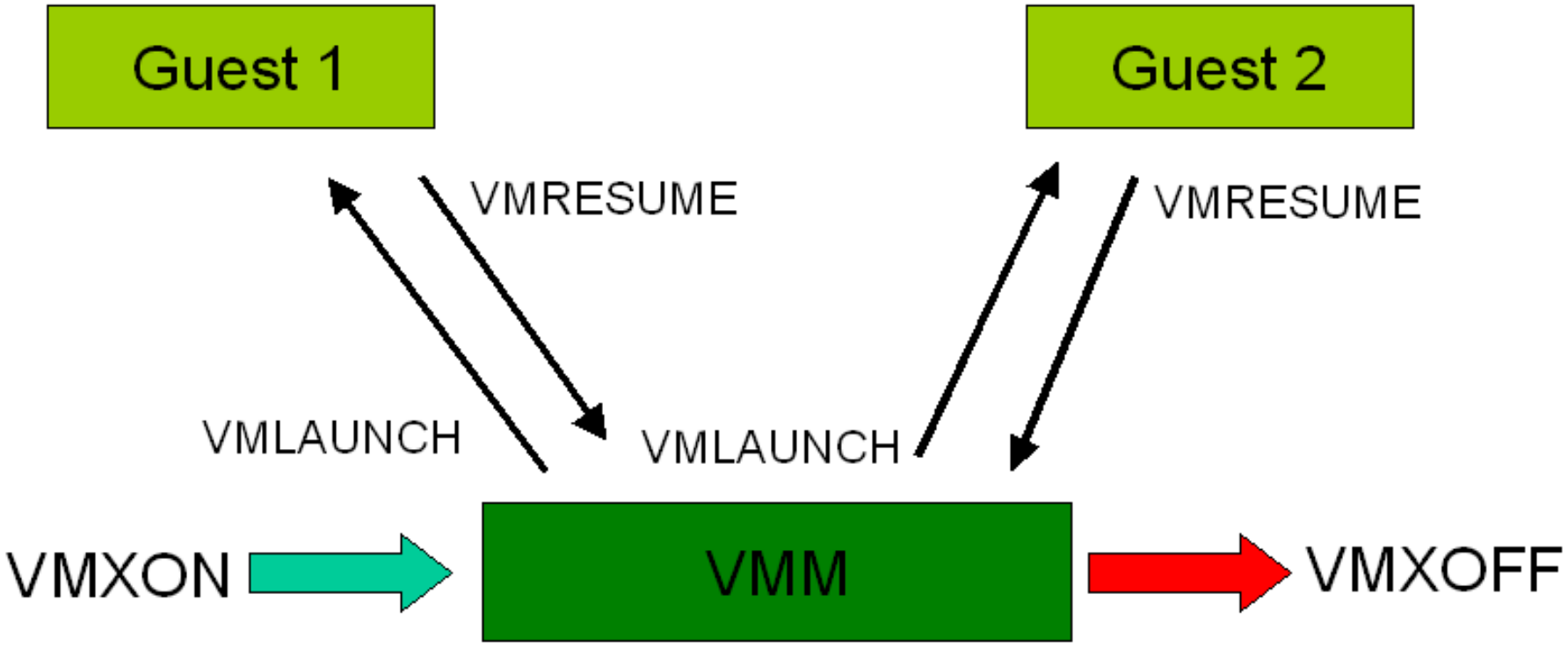
- Architecture x86 commune à Intel et AMD
  - On parle d'ISA: Instruction Set Architectures
  - ISA: Jeux d'instruction et architecture des registres
  - Instructions en mode noyau (accès RAM, périphériques) pour l'OS hôte (ou l'hyperiseur) seulement
  - Instructions en mode utilisateur: opérations sur les données du programme
- Virtualisation bas niveau:
  Techno Intel Virtualization Technology (Intel VT) et AMD Virtualization (AMD-V) introduites en 2006... puis enrichies
- Intel-VT et AMD-V aident l'hyperviseur dans sa gestion des VMs

Virtualisation processeur

- Le processeur permet un multiplexage temporels des processus
  $\Rightarrow$ un seul processus à la fois : l'hyperviseur ou l'OS invité ou le programme invité
- Question : comment l'OS invité récupère la main sur le programme invité? Et comment l'hyperviseur récupère la main sur l'OS invité
- Lorsqu'il y a un système non virtualisé - OS+programmes - l'OS modifie le timer de temps (dans un registre système) qui repasse le processeur en mode noyau
  $\Rightarrow$ Conflit : l'OS invité va l'utiliser pour ses programmes et l'hyperviseur pour ses OS invités.

Activation du mode

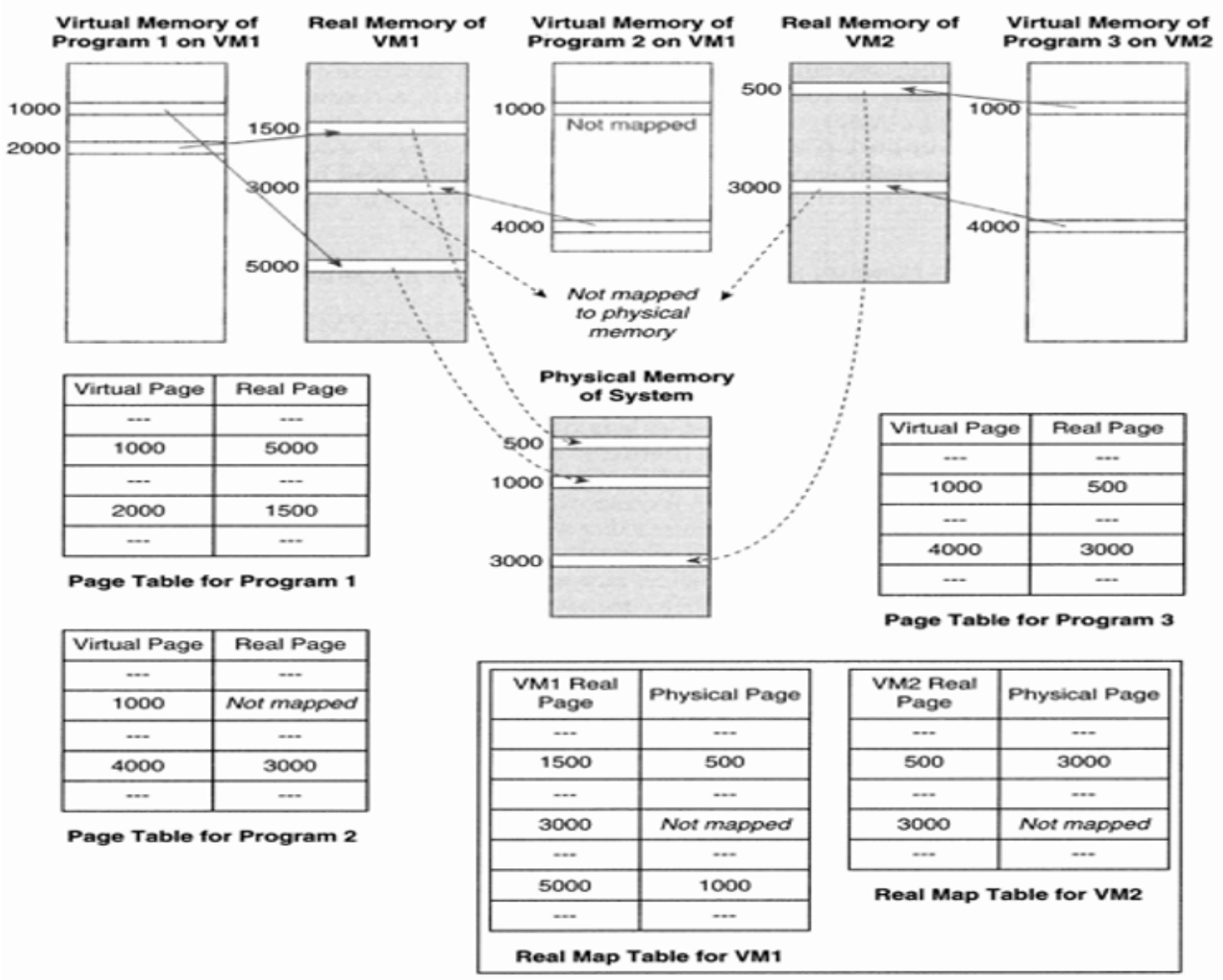
- Intel-VT et AMD-v permettent d'avoir deux timers de temps, un pour la VM et un pour l'hyperviseur.
- Pour cela, instructions spécifiques: VMXON/OFF: on va activer des VMs et VM launch/resume pour donner la mains aux VMs



Virtualisation mémoire

- Cas non virtualisé:
  - Il existe des pages virtuelles (de 1 à 2^{64}) pour chaque programme et des pages réelles vues par l'OS
  - Une table de correspondance est maintenue en RAM avec un cache maintenu par le processeur (TLB)
- Cas virtualisé $\rightarrow$ 2 niveaux de traductions :
  - Programme invité $\rightarrow$ OS invité
  - OS invité $\rightarrow$ hyperviseur
- Intel offre l'extended page tables (EPT), connue sous le nom *SLAT, Second-Level Address Translation* chez AMD
- Chez Intel, depuis le Core i3

Virtualisation mémoire



Virtualisation E/S

- E/S lentes. Il existe des techniques typiques pour améliorer les performances.
  - déportage (*offload*)du calcul de la somme de contrôle sur la carte réseau
  - Accès direct mémoire (DMA) : OS indique au contrôleur du périphérique une zone mémoire physique où écrire ses données.

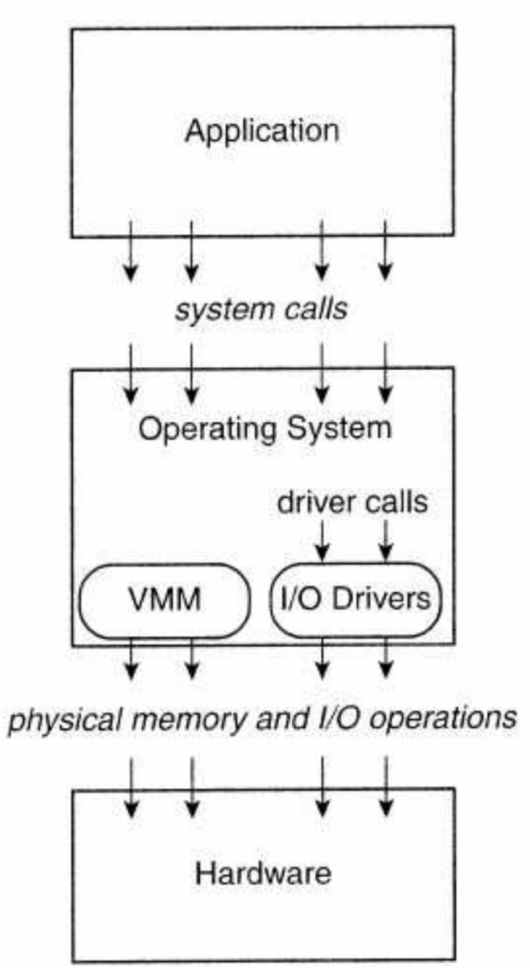
Virtualisation E/S

- Une des tâches les plus complexes du fait de la prolifération des périphériques
- Technique générale : offrir une interface virtualisée à l'OS invité

Typologie des périphériques

- Dédiés. Ex : écran, clavier.
- Partitionnés. Ex : disque
- Partagés. Ex: carte réseau
- Spoolés. Ex : imprimante. Partagé, mais à un niveau de granularité élevé (fichiers dans spooler). En pratique, imprimantes souvent accédées par le réseau $\rightarrow$ retour au cas précédent
- Inexistants. Ex : switch virtuel qui permet la communication inter-VM.

A quel niveau intercepter les E/S?

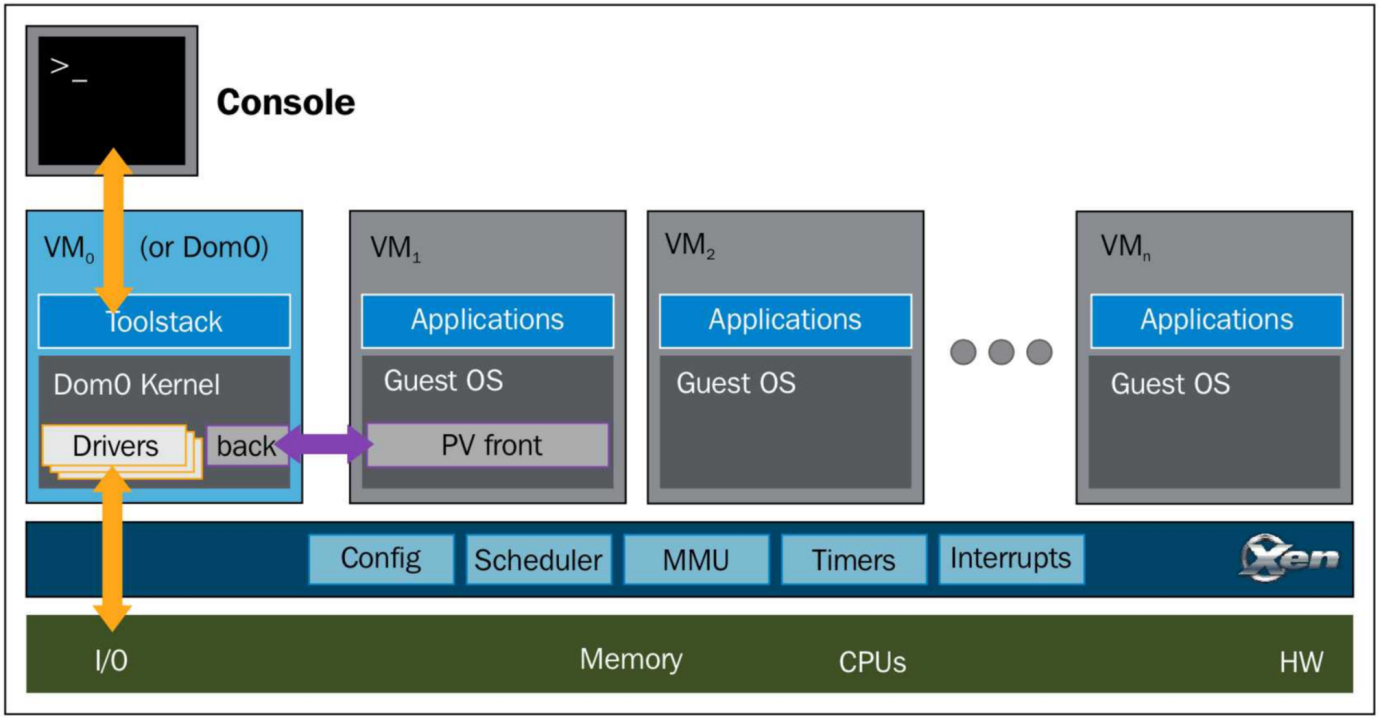


A quel niveau intercepter les E/S?

- Appels système? Compliqué
- Instruction d'E/S du processeur? Compliqué mais des techno Intel le permettent
- Driver? Souvent utilisé, ex : Virtualbox additions invité, VMware Tools, Xen.
  - On offfe la VM un driver spécial à la VM qui permet une discussion directe avec l'hyperviseur.

Exemple de Xen

- Driver a un *front-end* dans la VM et un *back-end* dans Dom0
  Dom0 : un VM spéciale, qui peut parler aux périphériques physiques
- Dom0 est séparée de l'hyperviseur pour ne pas exposer l'hyperviseur aux bugs (fréquents) dans les drivers.



Source: http://www.xenproject.org/

Support processeur

Idée : donner un accès direct à l'OS invité sans médiation de l'hyperviseur

- Intel® Virtualization Technology for Directed I/O (VT-d)
- Virtual Machine Device Queues (VMDQ),
- Single Root I/O Virtualization (SR-IOV, a PCI-SIG standard)
- Intel® Data Direct I/O Technology (Intel® DDIO) enhancements

Exemple DMA

- Système non vitualisé OS invité "voudrait" utiliser du DMA pour ses programmes
- Problème: OS invité ne voit pas les pages physiques (seul l'hyperviseur les voit)
- Utilisation d'une I/O MMU (memory management unit - sur le processeur) $\rightarrow$ qui fait la correspondance entre addresse physique de l'hôte et adresses physiques (réelles) de l'hyperviseur.

Containers

Guillaume Urvoy-Keller

November 26, 2019

Agenda

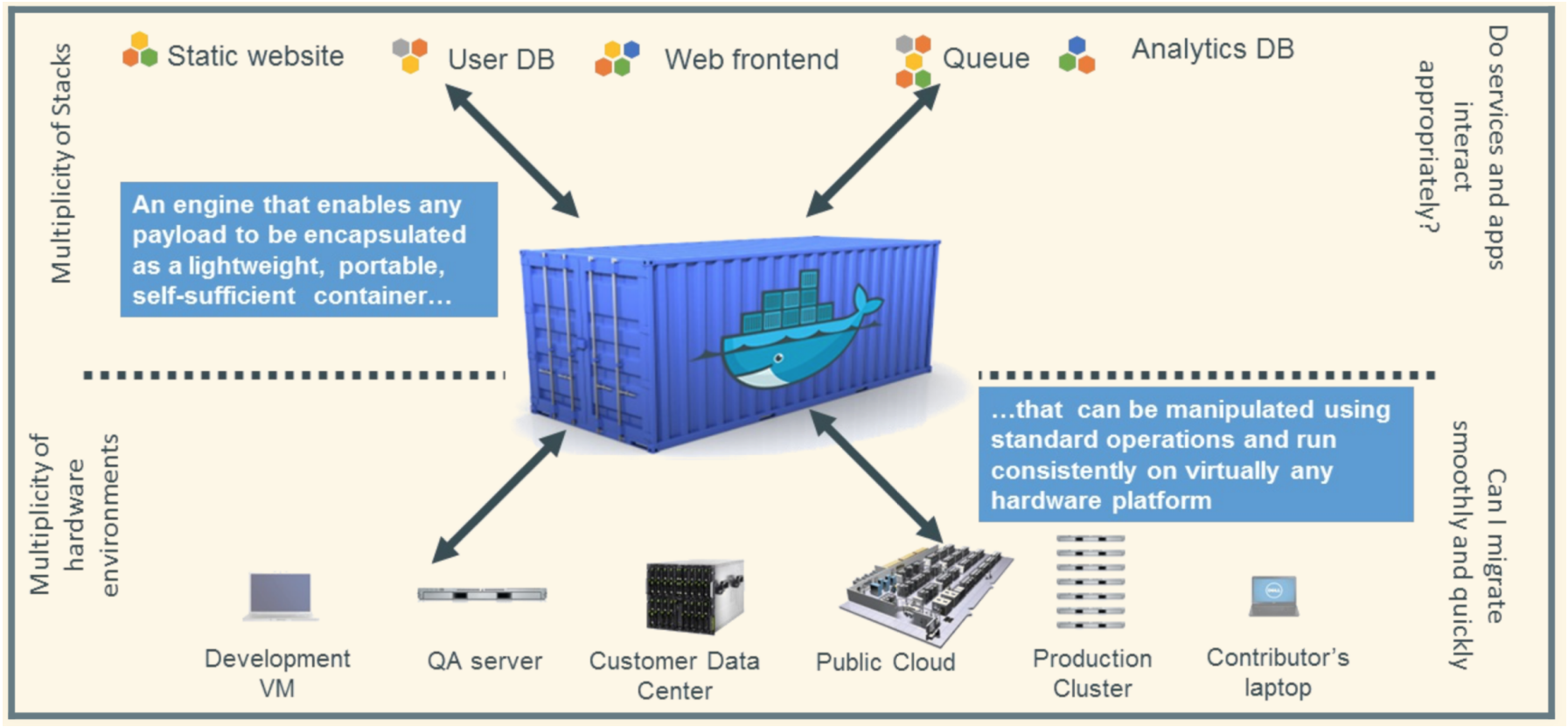
Source documents

- Remerciement à Christian Benedetti, administrateur système, Université Côte d'Azur pour son cours et ses articles dans Linux Magazine sur les containers (notamment)

Les promesses des containers

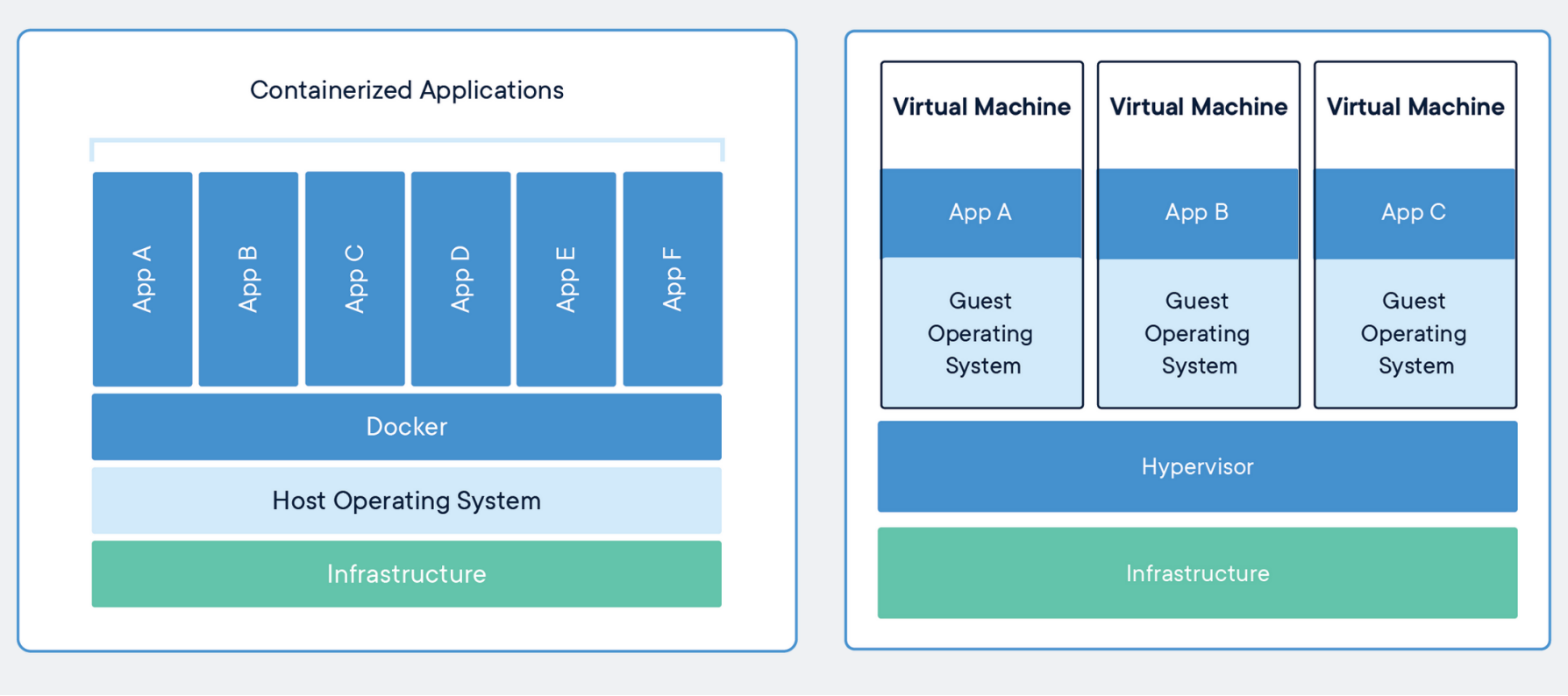
- Les installations (librairies) des différentes applications ne créent pas des problèmes de configuration mutuels
- Portabilité du container: déploiement possible sur des architectures variées.

Les promesses des containers



Source : docker.com

VMs vs. container



Source : docker.com

VMs vs. container

VM :

- +
  - OS complet
  - Isolation "parfaite" entre VM
- -
  - Usage disque important
  - Impact sur les performances

Container :

- +
  - Faible espace disque utilisé
  - Impact quasi nul sur les performances CPU, réseau et I/O
- -
  - Fortement lié au kernel hôte
  - Sécurité

Container Linux

(Groupe de) processus isolé(s) des mécanismes noyaux.

- Namespaces
- Cgroups

Namespaces

- Apparue dans le noyau 2.4.19 en 2002, réellement utiles en 2013 dans le noyau 3.8
- Limite ce qu'un processus peut voir du système:
- 6 namespaces : pid, net, mnt, uts, ipc, user
- Chaque processus est dans un namespace de chaque type
- **Les namespaces sont natifs en Linux**: même si il n'y a pas de container créé ou Docker installé, chaque processus créé est associé à 6 namespaces (namespaces par défaut)

Namespace Réseau (NET)

Le processus ne voit que la pile réseau du namespace dont il fait partie:

- Ses interfaces (eth0, l0, différentes de l'hôte)
- Sa table de routage séparée.
- Ses règles iptables (firewall, NAT, etc)
- Ses sockets

Namespaces

UTS

détermine le nom de l'hôte (get/set hostname)

IPC (Inter-Process Communication

Permettent à un groupe de processus d'un même namespace d'avoir leurs propres

sémaphores, files de messages et mémoire partagée

... sans risque de conflit avec d'autres groupes d'autres namespaces

USER

mappe uid/gid vers différents utilisateurs de l'hôte

uid 0 $\Rightarrow$ 9999 du container C1 correspond à uid 10000$\Rightarrow$ 119999 sur l'hôte

uid 0 $\Rightarrow$ 9999 du container C2 correspond à uid 12000$\Rightarrow$ 139999 sur l'hôte

Namespaces

MOUNT

Un namespace:

- dispose de son propre rootfs (conceptuellement proche d'un chroot)
- peut disposer de son propre /proc, /sys
- peut aussi avoir ses montages "privés"
- son /tmp (par utilisateur, par service)

PID

- Un processus ne voit que les processus de son namespace PID
- Chaque namespace pid a sa propre numérotation, débutant à 1
- Si le PID 1 disparait, le namespace est détruit.
- Un processus dans un namespace PID est aussi visible dans le namespace par défaut: il a un numéro dans les 2 (pas le même).

Manipulation Namespaces

Créés à l'aide des commandes clone() ou unshare()

Matérialisés par des pseudo-files dans /proc/$PID/ns fichiers dans /proc/{pid}/ns

Exemple pour le processus ayant le PID 16 :

```
root@debian:/proc/16/ns# ls -al
total 0
dr-x--x--x 2 root root 0 Nov 25 15:01 .
dr-xr-xr-x 9 root root 0 Nov 14 12:18 ..
lrwxrwxrwx 1 root root 0 Nov 25 15:01 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov 25 15:01 uts -> 'uts:[4026531838]'
```

Control groups

- Fonctionnalité du noyau, apparue en 2008 (noyau 2.6.24)
- Contrôle les ressources d'un processus en terme de CPU, mémoire, réseau, I/O

Namespace vs. cgroups

Namepsaces permettent une séparation logique des containers alors que les cgroups permettent de spécifier quelles resources physiques ils peuvent consommer

Cgroups

- Une hiérarchie par resource (ou groupe de resources): hiérarchie CPU, Mémoire
- Les groupes sont matérialisés par des pseudos systèmes de fichiers: généralement montés dans /sys/fs/cgroup
- Pid 1 est placé à la racine de chaque hiérarchie
- Les nouveaux processus sont démarrés dans le groupe de leur parent $\Rightarrow$ hiérarchie

Pour placer un processus dans un cgroup: On écrit le numéro du processus dans un fichier spécial

Docker - Les bases

Important

- Docker utilise les 6 namespaces et les cgroups pour construire des containers
- Docker est un moteur de gestion des containers.

Les bases

Démarre un container basé sur l'image debian et lance un processus, /bin/echo

```
$ docker run debian /bin/echo "Salut"
Salut
```

Démarre un terminal en interactif relié à /bin/bash (dans l'image debian)

```
$ docker run -it debian /bin/bash
root@2c666d3ae783:/# ps -a
PID TTY TIME CMD
6 ? 00:00:00 ps
```

Executer une commande dans le namespace du container

```
$ docker exec -it happy_mietner /bin/bash
root@2c666d3ae783:/# exit
```

Les bases

Se détacher d'un container :

```
root@2c666d3ae783:/# ^P^Q
```

Se rattacher à un container existant :

```
$ docker attach happy_mietner
root@2c666d3ae783:/#
```

Gestion des containers

Lister les containers avec ps

```
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
 2c666d3ae783 debian "/bin/bash" 44 seconds ago Up 42 seconds
```

Lister les processus d'un container

```
$ docker top happy_mietner
UID PID PPID C STIME TTY TIM
root 23338 867 0 15:06 pts/15 00:00
```

Voir ce qui se passe dans un namespace depuis l'extérieur : le log

```
$ docker logs hungry_visvesvaraya Salut
Salut
```

Gestion des containers

Cycle de vie d'un container :

- Démarré
- Arrêté (par exemple suite à un exit) $\Leftrightarrow$ équivalent d'une VM suspendue
- Redémarrage ou Destruction

Exemple

On démarre un container, puis on s'en détache, puis on s'y rattache puis on le suspend avec un exit.

A ce moment, le container n'est plus visible avec ps mais avec ps -a

Puis on le démarre avec un start

```
sh−3.2# docker run −it ubuntu /bin/bash
root@8cbe7248c918:/# ^P^Q
sh−3.2# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
8cbe7248c918 ubuntu "/bin/bash" 23 seconds ago Up 22 seconds affectionate_dhawan
sh−3.2# docker attach 8cbe7248c918
root@8cbe7248c918:/# exit
sh−3.2# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
sh−3.2# docker ps −a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
8cbe7248c918 ubuntu "/bin/bash" 44 seconds ago Exited (0) 6 seconds ago
        affectionate_dhawan
sh−3.2# docker start −i 8cbe7248c918
root@8cbe7248c918:/#
```

Images

- Ensemble de fichiers à partir desquels les containers sont construits
- Image composée de couches
- Des images peuvent partager des couches pour optimiser

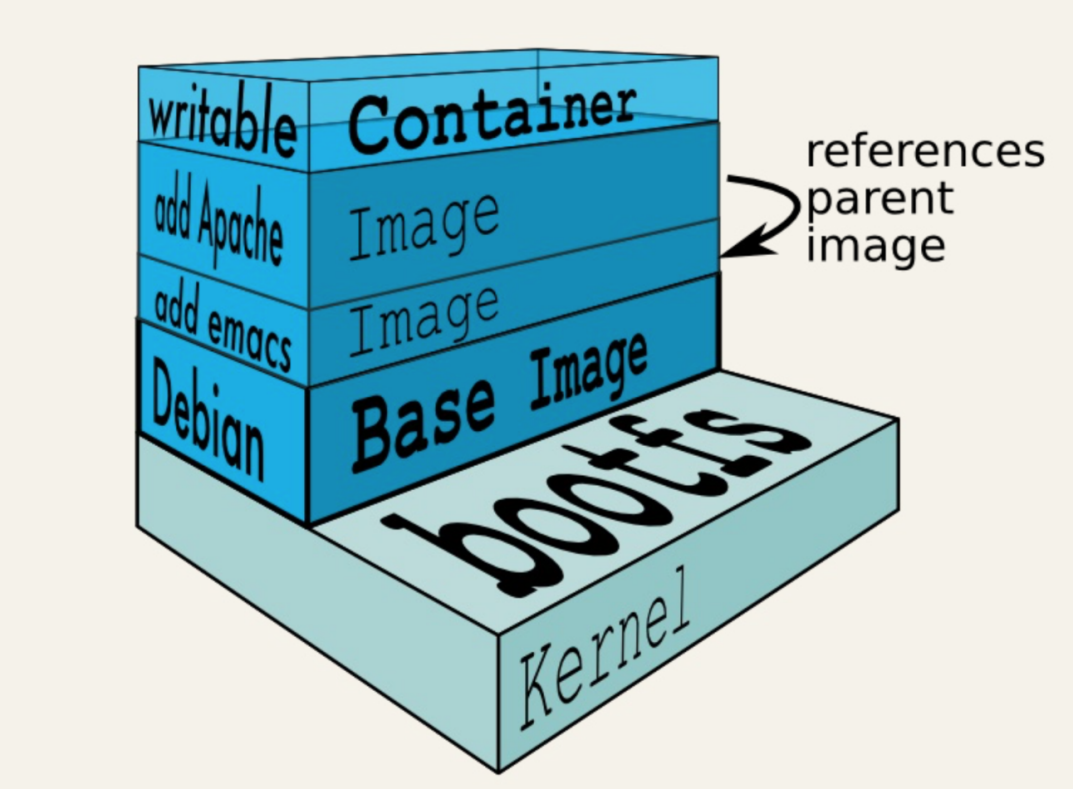


Image - Exemple

```
sh−3.2# docker pull ubuntu # telechargement ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
sh−3.2# docker pull nginx # telechargement nginx
Using default tag: latest
latest: Pulling from library/nginx
000eee12ec04: Pull complete # couche 1
eb22865337de: Pull complete # couche 2
bee5d581ef8b: Pull complete # couche 3
Digest: sha256:50cf965a6e08ec5784009d0fccb380fc479826b6e0e65684d9879170a9df8566
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
sh−3.2# docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
nginx latest 231d40e811cd 2 days ago 126MB
ubuntu latest 775349758637 3 weeks ago 64.2MB
```

Images

- Les images sont téléchargées depuis le hub docker
- Possibilité d'avoir un hub privé
- Nom de l'image :
  - un nom simple, ex: ubuntu, nginx → images officielles
  - un nom du type benedetti/nginx: image d'un utilisateur
  - tag: numéro de version

```
$ docker tag debian benedetti/debian:8.6 docker images | grep debian
$ docker images | grep debian
debian latest 140f9bdfeb97 4 days ago 123 MB
benedetti/debian 8.6 93a2e30f1000 4 days ago 123 MB
debian 8.5 93a2e30f1000 3 months ago 125.1 MB
debian 8.4 f854eed3f31f 5 months ago 125 MB
debian 8.2 32f2a4cccab8 8 months ago 125 MB
```

Création Images

A partir d'un fichier Dockerfile :

```
FROM debian # depuis l'image debian
RUN apt-get update # on execute des commandes
RUN apt-get install -y nginx
CMD nginx -v # commande par defaut qui sera execute
```

Construction de l'image avec le nom benedetti/nginx et le tag 0.2:

```
$ docker build -t benedetti/nginx:0.2 .
Sending build context to Docker daemon 2.048 kB Step 1 : FROM debian
---> 93a2e30f1000
Step 2 : RUN apt-get update
---> Running in 20f50a8284f5
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB] ...
Processing triggers for sgml-base (1.26+nmu4) ... ---> 95f9e15fa8d2
Removing intermediate container e99f0c6f5bd2 Successfully built 95f9e15fa8d2
```

Réseaux et Volumes

Gestion Réseaux

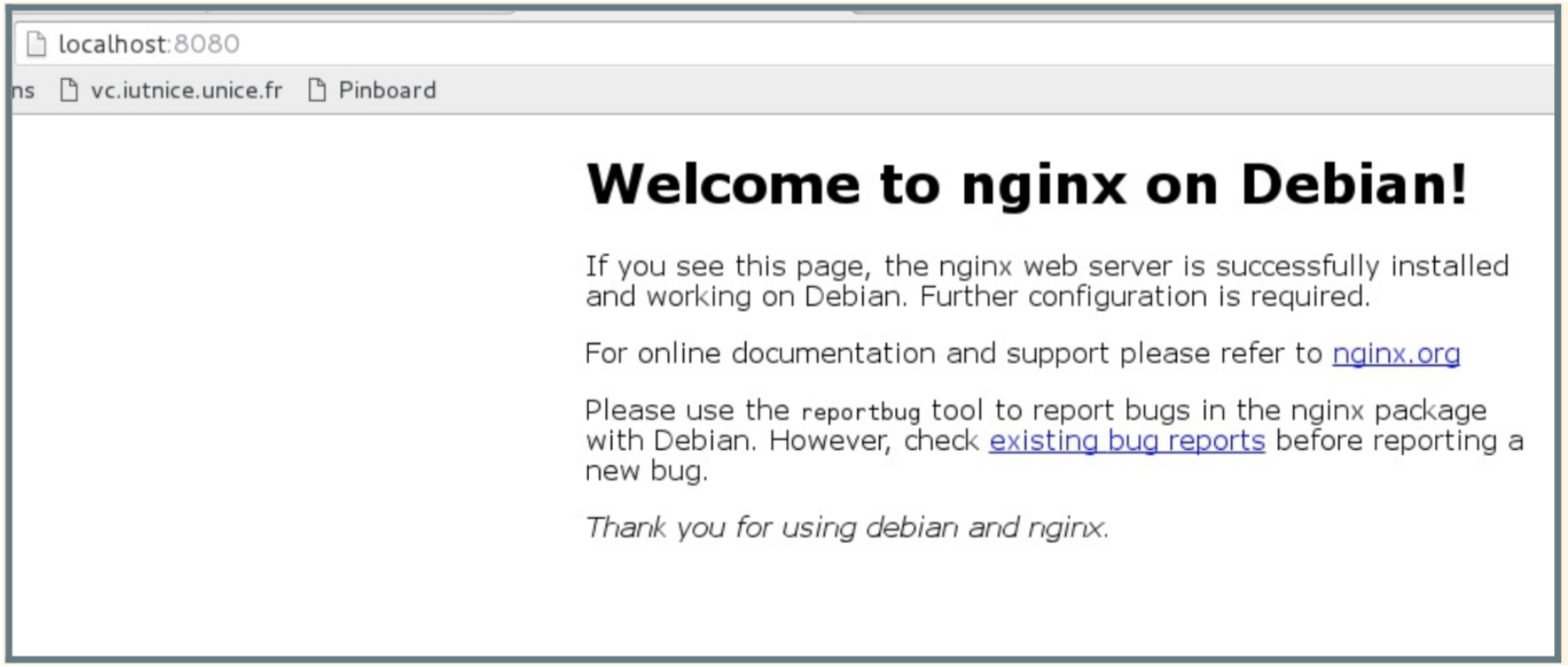
- Tous les ports sont privés par défaut
- Il faut les rendre explicitement accessibles depuis l'extérieur → accessibles comme un port de la machine hôte
- Exemple : le port 80 du container va correspondre au port 8080 sur l'hôte

```
$ docker run -d -p 8080:80 nginx
```

- L'adresse IP du container n'est **jamais exposé** à l'extérieur: on expose une application seulement!

Réseaux

```
$ docker run -d -p 8080:80 nginx
```



Volumes

On peut monter un volume de l'hôte dans le container, par exemple ici, le répertoire courant (pwd) depuis lequel on lance le container, qui est monté dans /usr/share/nginx/html.

```
$ docker run -d -v $(pwd):/usr/share/nginx/html -P nginx
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
eca9f876dc7f nginx "nginx -g 'daemon of..." 5 seconds ago Up 3 seconds 0.0.0.0:32769->80/
        tcp confident_hellman
$ echo "Bienvenue sur mon image Nginx" > index.html
$ curl http://localhost:32769
 Bienvenue sur mon image Nginx
```

Volumes

Création d'un volume nommé

```
$ docker volume create −−name=logs logs
$ docker run −P −v logs:/var/log/nginx −d benedetti/nginx:0.7 $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
eca9f876dc7f nginx "nginx −g 'daemon of..." 5 seconds ago Up 3 seconds rapid_joe
$ docker run −it −−volumes−from rapid_joe debian ls /var/log/nginx access.log error.log
```

Les volumes sont persistants

```
$ docker rm −f rapid_joe
$ docker volume ls DRIVER
local local
VOLUME NAME 57a0848c5e5f2924be84e830757922c7b5b856aa5bac12e494da495 logs
```

Docker Swarm

Docker Swarm

Problème

Vous avez plusieurs hôtes Docker.
Vous désirez les utiliser sous forme de cluster, et répartir de manière transparente l'exécution de conteneur.

Solution

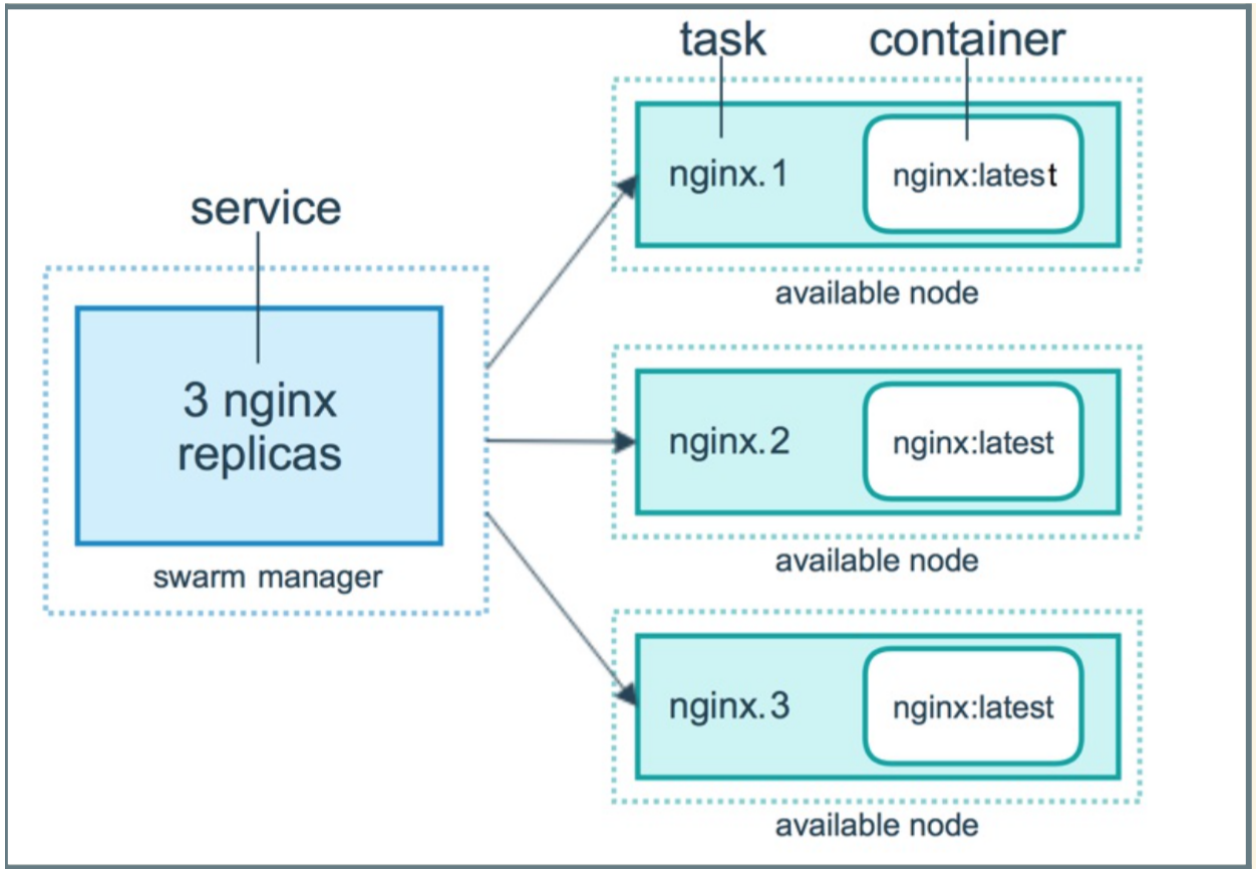
Docker Engine 1.12 inclut le mode swarm pour nativement gérer un cluster de Docker Engines qu'on nomme un swarm (essaim).
CLI Docker utilisée pour créer un swarm, déployer des service d'application sur un swarm

Docker Swarm

- Un swarm consiste en un ou plusieurs **managers** qui permettent de déployer les containers
- Au niveau swarm, on parle d'un **service** et non d'un container
- Un service contient un ou plusieurs containers
- Container exécutés sur les noeuds
- **Modes d'exécution:**
  - Répliqué : un container du service par noeud
  - Global: les N containers du services sont répartis sur le cluster

Docker Swarm

Exemple : service en mode répliqué avec container nginx



Création Swarm

Sur le manager dont l'IP est 172.31.4.182:

```
$ docker swarm init --advertise-addr <MANAGER-IP> Swarm initialized: current node (8jud)
        is now a manager. To add a worker to this swarm, run the following command:
docker swarm join --token SWMTKN-1-59fl4ak4nq4eprhrola2l87... 172.31.4.182:2377
```

Sur les noeuds :

```
docker swarm join --token SWMTKN-1-59fl4ak4nq4eprhrola2l87... 172.31.4.182:2377
```

On fait un ls niveau swarm et on trouve 2 noeuds actifs

```
docker node ls
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS
8jud...ox4b * ip-172-31-4-182 Ready Active Leader
ehb0...4fvx ip-172-31-4-180 Ready Active
```

Lancement tâche sur Swarm

Sur le manager, on lance le service helloworld :

```
$ docker service create --replicas 1 --name helloworld alpine ping docker.com
9uk4639qpg7npwf3fn2aasksr
```

On fait un ls niveau service (le service n'a qu'un replica ici):

```
$ docker service ls
ID NAME SCALE IMAGE COMMAND
9uk4639qpg7n helloworld 1/1 alpine ping docker.com
```