Docker dans le contexte DevOps

Ce document est une introduction au système de conteneurisation Docker.

Après une présentation du concept, nous découvrons les commandes de base permettant de gérer des applications isolées dans leurs contextes.

Bonne lecture…

Pierre ROYER Manager | Architecte | Formateur #numérique

I. INDEX

II. PRÉAMBULE

A. Ce document

Informations

Nom du document	DevOps.Docker.docx	Référence	DEVOPS-DOCKER
Version	2020.02.02	Pages	59
Date de création	07/03/2017	Dernière modification :	19/02/2023
Auteur :	Pierre ROYER Tél : (+33) 614 672 909 https://www.linkedin.com/in/pierreau	Contributeur(s) :	
Mode de diffusion	☐ confidentiel ☐ restreint ☒ interne ☐ libre	Liste de diffusion	https://pierreau.fr
Annexes :			

B. Conventions

Les syntaxes utilisées dans ce document :

[root@Rocky ~]# représente un prompt bash en root sur un serveur Rocky Linux
root@Debian:~ # représente un prompt bash en root sur un serveur Debian
[pierreau@Rocky ~]$ désigne un compte utilisateur local

Le contenu d'un fichier est encadré, les commandes sont en gras :

```
[Rocky@localhost ~]# vi /etc/ssh/sshd_config
```

```
PermitRootLogin yes
```

Les caractères en italique sont des exemples de paramètres :

```
192.168.100.100 ServeurA
192.168.100.101 ServeurB
```

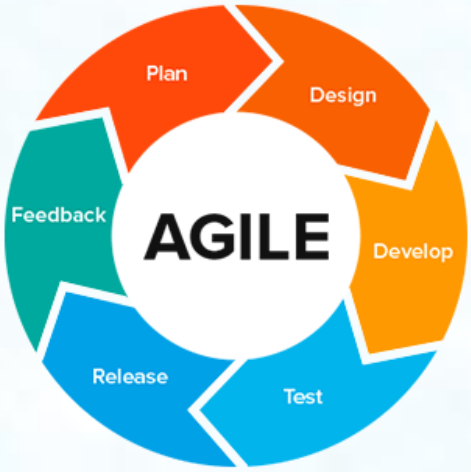
Information utile	Attention particulière	Risque important
		

III. Introduction

A. Contexte Agile

Les méthodes agiles sont des groupes de pratiques de pilotage et de réalisation de projets. Elles ont pour origine le manifeste Agile rédigé en 2001, qui consacre le terme d'« agile » pour référencer de multiples méthodes existantes.

Les méthodes agiles impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes. Elles reposent sur un cycle de développement itératif, incrémental et adaptatif.



Les principes :

1 : les individus et leurs interactions, plus que les processus et les outils
2 : des logiciels opérationnels, plus qu'une documentation exhaustive
3 : la collaboration avec les clients, plus que la négociation contractuelle
4 : l'adaptation au changement, plus que le suivi d'un plan.

B. DevOps

Le client est devenu le point central dans les contextes (méthodologies) Agiles, et les il est de bon ton de pouvoir s'adapter à ses demandes rapidement.

Avec les évolutions fulgurantes de l'informatique, les métiers se sont largement multipliés et spécialisés. Les technologies sont devenues complexes, les outils de développement multiples, et les contraintes métier (et business) sont pour leur part sujettes à des adaptations et mises à jour de plus en plus imminentes.

Afin de simplifier la flexibilité demandée par les métiers (dev), et la mise en place des évolutions technique (ops), le DevOps automatise les tâches intermédiaires via des outils (industrialisation), et en réorganisant les process de déploiement (méthodologies).

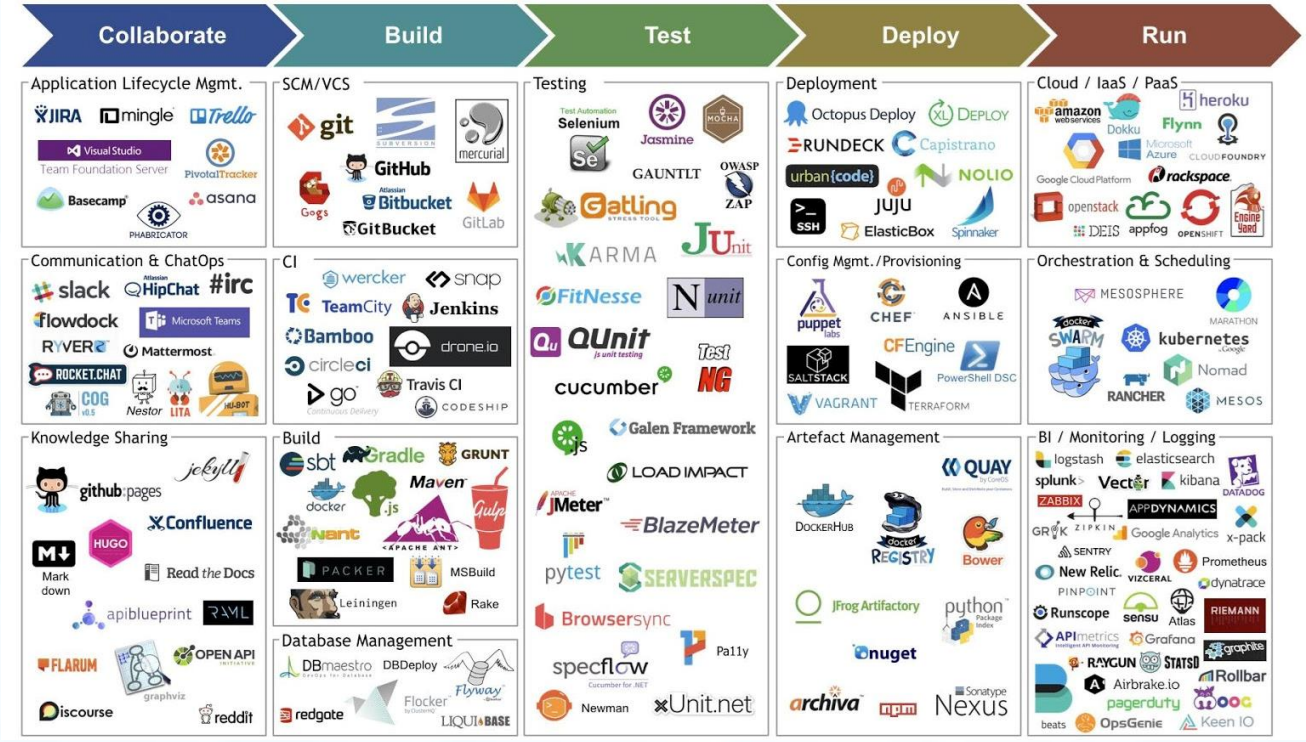
Les solutions techniques permettent une extension des ressources matérielles (scalabilité), gérées par les exploitants, et doivent aussi répondre à une flexibilité (ou élasticité) vers le bas, et/ou vers le haut.

Le cycle DevOps

Patrick Debois est le fondateur du devOps

Le DevOps est un mouvement en ingénierie informatique et une pratique technique visant à l'unification du développement logiciel (dev) et de l'administration des infrastructures informatiques (ops), notamment l'administration système.

Docker fait partie de cette gamme d'outils disponible, qui permettent de raccourcir le temps de mise en production des nouveaux socles, avec des pipelines CI / CD (Continuous Integration / Continuous Delivery) :



Dans ce contexte complexe, il devient parfois difficile d'implémenter rapidement, de manière fiable et sécurisée les infrastructures (serveurs, réseaux, bases de données, sauvegardes...), et les outils DevOps.

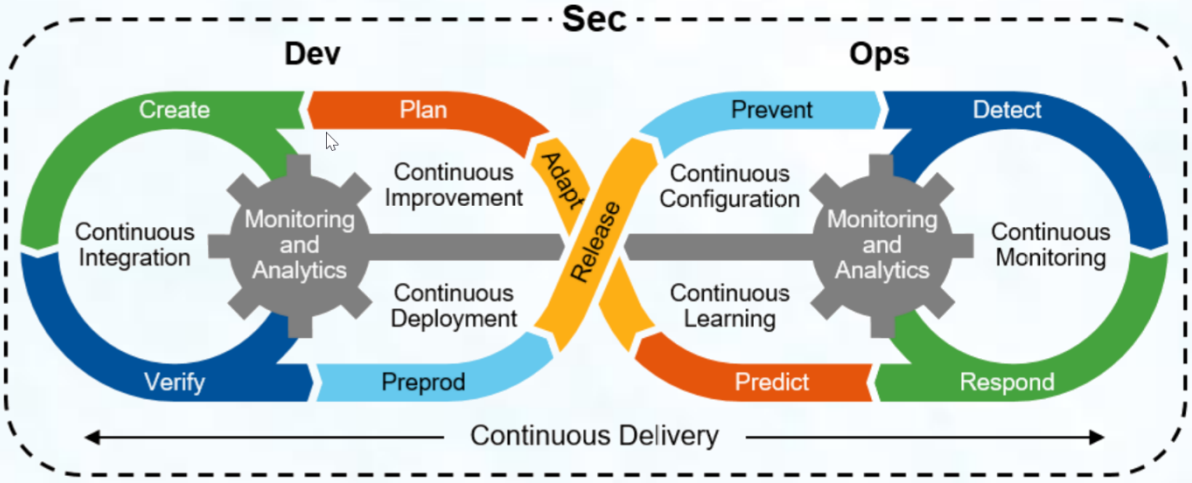
C. La sécurité avec DevSecOps

Les processus liés à la sécurité sont souvent négligés avec l'Agilité (manque de traçabilité), et dans le DevOps (déploiements très rapides). Ils sont aussi souvent isolés et confiés à une équipe spécifique.

À l'ère de la livraison et de l'intégration continues, le processus de développement est un défi à ne surtout pas sous-estimer. C'est pourquoi les entreprises vont de plus en plus souvent au-delà de l'approche DevOps qui lie étroitement le développement et l'exploitation dès le début du processus

pour y inclure la composante sécurité, d'où l'abréviation DevSecOps. La démarche DevSecOps tient compte à la fois des exigences en termes de vitesse de développement et de sécurité.

L'approche DevSecOps implique de réfléchir à la sécurité de l'application et de l'infrastructure dès le départ : principe de « security by default ». Il convient également d'automatiser certaines passerelles de sécurité afin d'éviter tout ralentissement des workflows DevOps.



Cette approche est notamment indispensable, pour répondre par exemple aux contraintes du RGPD.

La sécurité n'est pas forcément considérée comme une priorité dans le développement d'applications traditionnel. Docker dispose de solutions sécurisées, dans la mesure où les interfaces avec la chaîne d'intégration (Git ? Microsoft Windows ?) sont elles-mêmes sécurisées…

IV. LA VIRTUALISATION

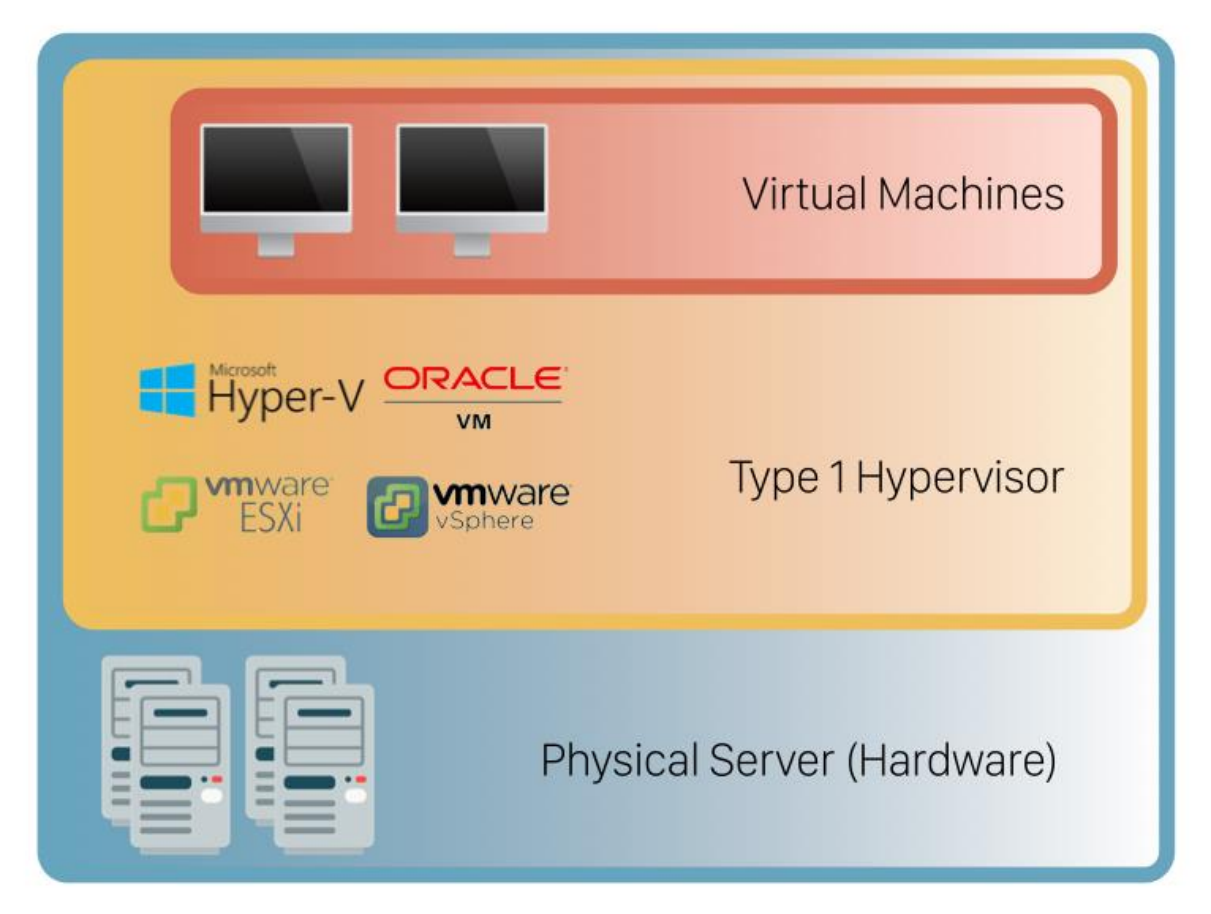
La virtualisation de serveurs est un concept qui existe depuis 1972 : IBM le pratiquait depuis 1972 ; avec des mainframes VM/370.

Les machines virtuelles partagent un même serveur physique, alors que les conteneurs partagent le même système d'exploitation.

La virtualisation sur les ordinateurs actuels n'est possible qu'avec l'activation sur les processeurs (UEFI) des technologies telles que Intel® VT, ,

A. Hyperviseur de type 1 (paravirtualisation)

En paravirtualisation, l'hyperviseur capture les appels système de l'invité et les transmet directement au matériel. L'invité exploite ainsi les ressources matérielles partagées par l'ordinateur hôte. La paravirtualisation offre des performances optimales. Cependant, les VMs doivent supporter le processeur de l'hôte : on ne peut pas installer un OS conçu pour Intel/AMD sur ARM64.

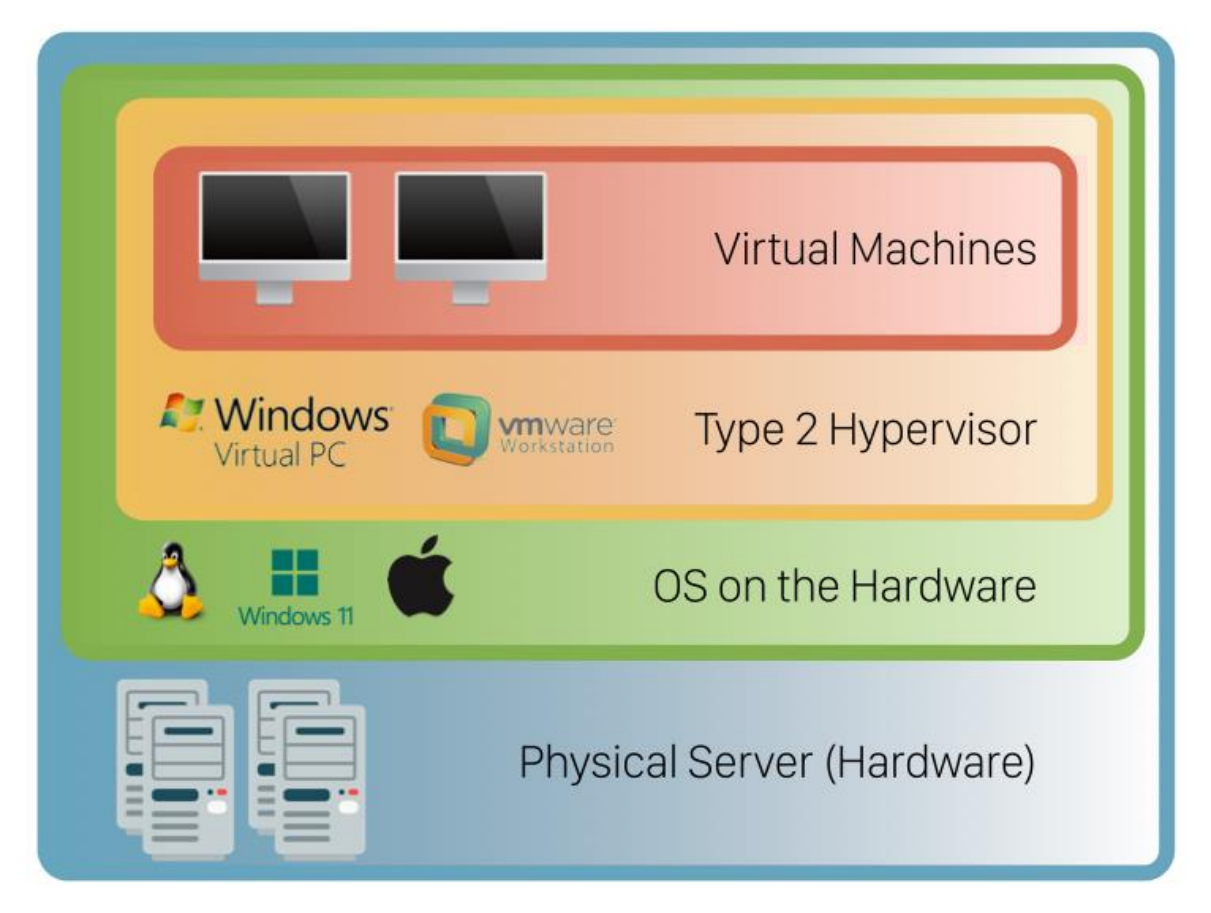


Source : https://phoenixnap.mx/

Les solutions présentes sur le marché sont Proxmox, VMware vSphere (ESXi / ESX), KVM, Citrix Xen Server, Microsoft Hyper-V Server, Parallels Server Bare MetalOracle VM.

A. Hyperviseur de type 2 (émulateurs)

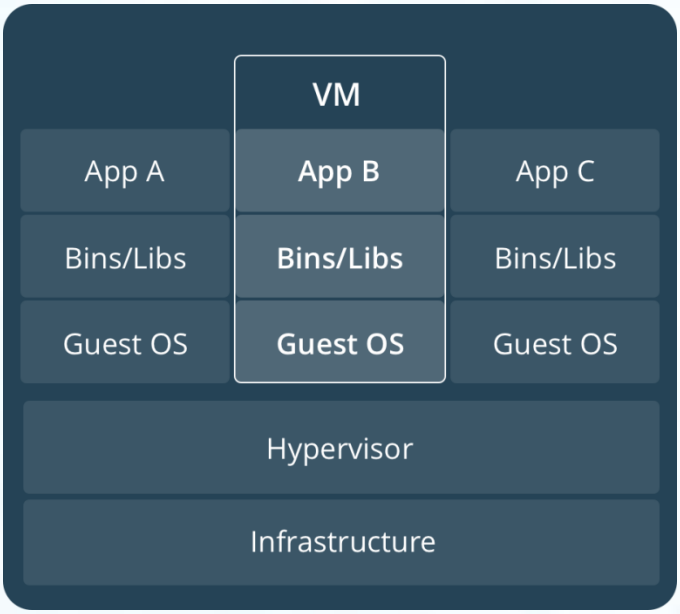
L'hyperviseur (hôte) crée un environnement virtuel simulé complet. Le système d'exploitation invité n'aura accès qu'à des ressources simulées (processeur, disque, mémoire, carte graphique, interfaces réseau...) et non aux ressources matérielles réelles. Les performances en émulation sont dégradées du fait de cette simulation.



Source : https://phoenixnap.mx/

Les solutions récurrentes sur le marché sont les logiciels VMware Workstation / Player, QEMU, Microsoft Microsoft Virtual Server, Parallels Desktop / Server, Oracle VM VirtualBox…

B. Virtualisation systèmes



Source : www.docker.com

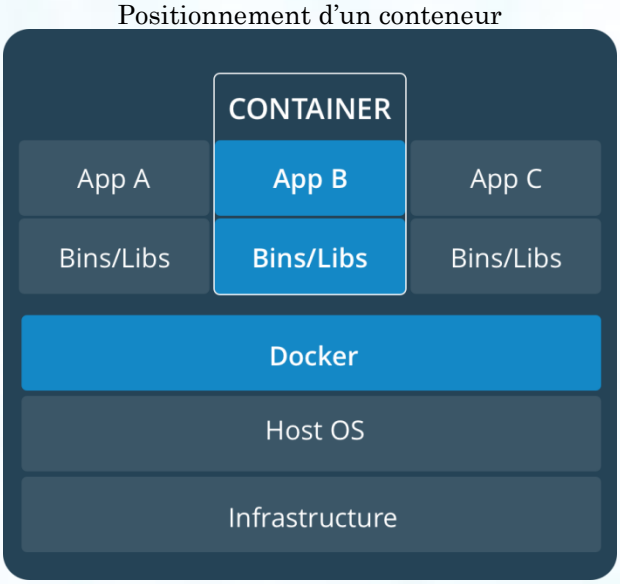
C. Virtualisation applicative

Contrairement aux machines virtuelles, les conteneurs Docker n'incluent pas de système d'exploitation, mais permettent d'isoler des applications dans des contextes.

Ce principe d'isolation existe depuis plusieurs années, et a été implémenté entre autres sur :

- Unix FreeBSD avec « BSD Jail » dès 1999
- Sun Solaris 10 (Oracle) en 2005. Les conteneurs Solaris sont appelés « zones »
- Linux en 2008 : LXC (LinuX Containers)

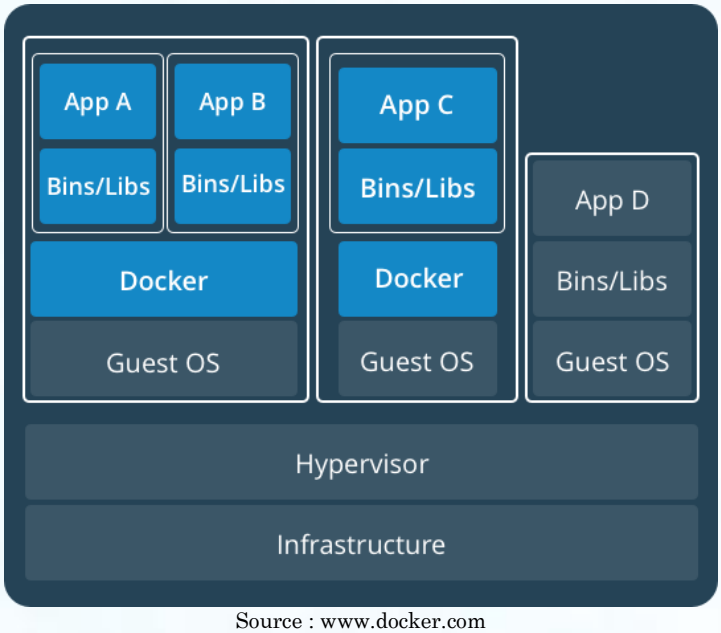
Positionnement d'un conteneur



Source : www.docker.com

D. Virtualisation virtuelle

La virtualisation système et applicative sont deux concepts différents, mais qui peuvent être complémentaires. Il est ainsi possible de contenir des conteneurs dans un système virtualisé :



Source : www.docker.com

Si l'on pousse le raisonnement, il est tout à fait envisageable (mais pas conseillé !) d'exécuter Docker… dans un autre conteneur Docker, lancé au sein d'une machine virtuelle qui est hébergée sur un serveur physique…

E. LinuX Containers

Il s'agit d'un système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation. Il est utilisé pour faire fonctionner des environnements Linux isolés les uns des autres dans des conteneurs, partageant le même noyau.

Le conteneur apporte une virtualisation de l'environnement d'exécution (processeur, mémoire vive, réseau, système de fichier…) et non pas de la machine

Installation :

```
[root@Rocky ~]# dnf install -y lxc lxc-templates wget
[root@Rocky ~]# systemctl start lxc
[root@Rocky ~]# lxd init
```

Quelques commandes :

- lxc-checkconfig
- lxc storage list
- lxc network ls
- lxc image list
- lxc launch images:ubuntu/jammy/amd64 my-ubuntu
- lxc-create
- lxc-start -n *container*
- lxc-attach
- lxc-console -n *container*
- lxc-info
- lxc list
- lxc-stop -n *container*
- lxc-destroy -n *container*

F. Présentation de Docker

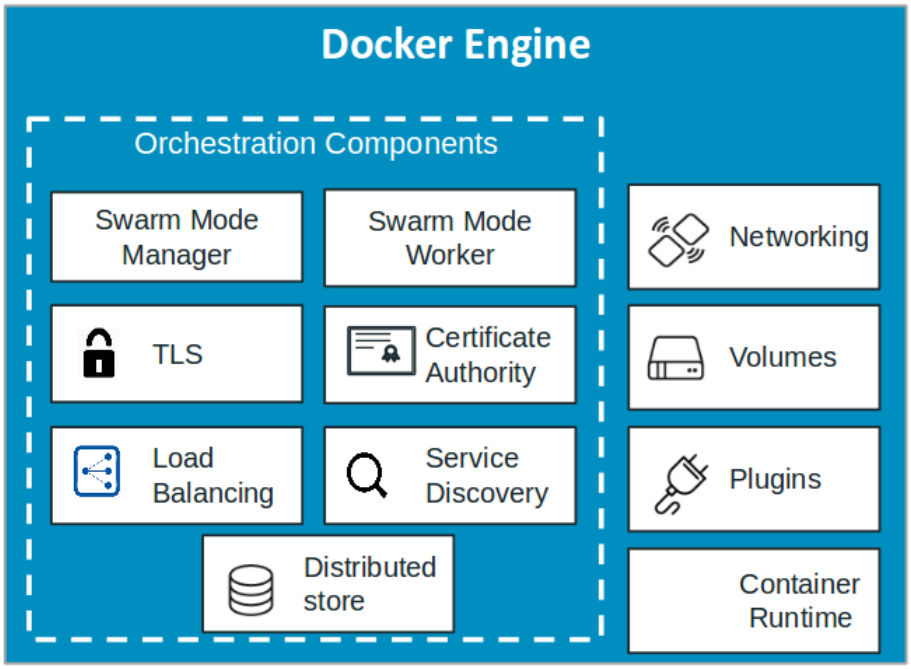
Docker est un logiciel sous licence Apache 2.0, qui a été distribué en tant que projet open source à partir de mars 2013.

Docker permet d'automatiser le déploiement d'applications dans des conteneurs logiciels. Il peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.

Docker utilise des fonctionnalités natives de Linux comme :

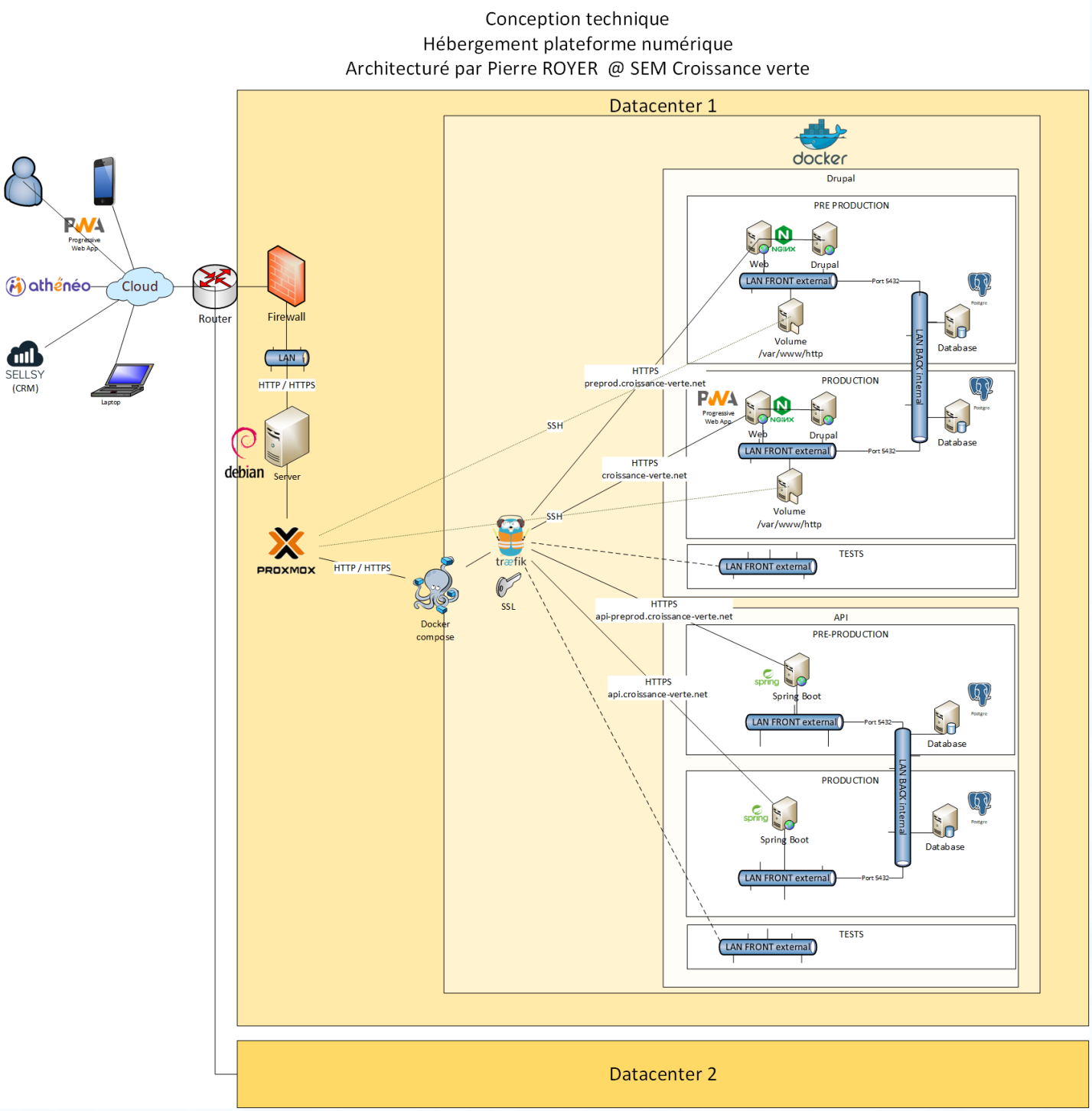
- son noyau
- ses Cgroups (control groups) : fonctionnalités pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.)
- une surcouche de LXC (LinuX Containers).

Outre le fait d'isoler un système de fichiers et des réseaux, Docker Engine intègre des outils d'orchestration, que nous aborderons dans le chapitre dédié à Swarm :



G. Exemple d'implémentation

Le schéma suivant représente un exemple d'implémentation de conteneurs dans plusieurs environnements, accessibles via un reverse-proxy Traefik. L'ensemble est hébergé dans une machine virtuelle.



Docker répond aux principes d'urbanisation, avec un ensemble de **services faiblement couplés** (→ simplifier la gestion des dépendances) : une application doit appartenir à un seul bloc, afin de limiter les impacts lorsqu'on remplace celle-ci. Le site official de Docker préconise : " It is generally recommended that you separate areas of concern by using one service per container"

MAIS ALLONS DE CE PAS AU CŒUR DU SUJET…

V. INSTALLATION

Docker est distribué sous deux licences :

- Docker EE (Enterprise Edition) qui offre un support payant. Trois versions existent : Basic, Standard et Advanced.
- Docker CE (Community Edition), sans support.

Ce document se base sur Docker CE, et un système d'exploitation Rocky Linux 9.0

A. Procédure

Paramétrer une IP fixe et une passerelle : **/etc/sysconfig/network-scripts/ifcfg-ens*33***
Faire correspondre le nom et l'adresse de l'interface réseau : **/etc/hosts**
Déclarer les DNS : **/etc/resolv.conf**
Nommer le serveur :

```
[root@Rocky ~]# hostnamectl set-hostname 'Docker'
```

Prévoir un espace disponible suffisant (plusieurs Go) sur le point de montage **/var/lib/docker/**
Les conteneurs seront stockés à cet endroit.

1. Cgroup

Les ressources des conteneurs doivent absolument être contrôlées via les cgroups, afin de ne pas permettre des débordements systèmes du type « Out Of Memory Exception » :

- Version 2 : `/sys/fs/cgroup/cgroup.controllers`
- Version 1 : `/proc/cgroups`

Les cgroup à activer d'office : cpu, cpuset, memory

2. Installation via un script

```
root@zabbix:~# apt install curl
root@zabbix:~# curl -fsSl https://get.docker.com | sh
```

3. Installation manuelle

Paramétrage des dépôts de docker :

```
[root@Rocky ~]# dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo
[root@Rocky ~]# dnf repolist -v
```

Recherche de la dernière version :

```
[root@Rocky ~]# dnf list docker-ce
Docker CE Stable - x86_64
Available Packages
docker-ce.x86_64              3:20.10.12-3.el8          docker-ce-stable
```

Installation :

Selon les distributions Linux, il se peut que la version de la dépendance par défaut du package **containerd.io** ne convient pas : il faut utiliser l'option **--nobest** ou **--allowerasing**

```
[root@Rocky ~]# dnf install docker
```
ou
```
[root@Rocky ~]# dnf install docker-ce --nobest
[root@Rocky ~]# dnf install docker-ce --allowerasing
```

Créer un compte de service pour docker :

```
[root@Rocky ~]# useradd -m docker
```

Ajouter ce compte au groupe docker :

```
[root@Rocky ~]# usermod -aG docker docker
[root@Rocky ~]# id docker

[root@Rocky ~]# systemctl start docker
[root@Rocky ~]# systemctl enable docker
```

Utiliser le compté créé pour gérer docker :
```
[root@Rocky ~]# su - docker
```

B. Post-installation

Vérifications de l'installation :

```
[docker@Rocky ~]$  docker version
[docker@Rocky ~]$  docker info
```

Vérification des interfaces et réseaux virtuels :

Lors de l'installation, une interface virtuelle est créée :

```
[docker@Rocky ~]$  ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
```

Docker met à disposition des réseaux virtuels, visibles via la commande suivante :

```
[docker@Rocky ~]$  docker network ls
NETWORK ID          NAME                DRIVER
7fca4eb8c647        bridge              bridge
9f904ee27bf5        none                null
cf03ee007fb4        host                host
```

Le réseau "bridge" utilise l'interface réseau « docker0 », qui est définie dans la machine hôte.

Test de l'installation :

```
[docker@Rocky ~]$  docker run hello-world
```

Configurations stockage / IPV6…

```
[docker@Rocky ~]$  vi /etc/docker/daemon.json
```

```
{
    "storage-driver": "vfs"
}
#{
#    "storage-driver": "devicemapper"
#}
{
  "ipv6": true,
  "fixed-cidr-v6": "2a01:e0a:172:3910::/64"
}
```

```
[docker@Rocky ~]$ systemctl reload docker
```

Alias

Afin de faciliter la saisie des commandes Docker, je recommande l'utilisation d'alias personnalisés Unix :

```
[docker@Rocky ~]$ vi ~/.bashrc
```

```
alias dat='docker attach'
alias dex='docker exec -ti'
alias dim='docker images'
alias dkl='docker kill'
alias dps='docker ps -a'
alias drm='docker rm'
alias drmi='docker rmi'
alias dsta='docker start'
alias dsto='docker stop'
alias dcps='docker-compose ps'
alias dsdf='docker system df -v'
etc...
```

VI. PREMIERS PAS

A. Apperçu des commandes

```
[docker@Rocky ~]$  docker --help

Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.8.0-docker)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  scan*       Docker Scan (Docker Inc., v0.17.0)
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams 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 to files or directories 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 Docker objects
  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
  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 one or more containers
  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
  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
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  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
  wait        Block until one or more containers stop, then print their exit codes
```

B. Installation d'un conteneur

1. Récupération de l'image

Les images de conteneurs peuvent contenir aussi bien des applications, que d'autres distributions Linux. Les exemples suivants vont se baser sur une image Debian, ayant une réputation d'au moins 10 étoiles :

```
[docker@Rocky ~]$  docker search --filter stars=10 debian
[docker@Rocky ~]$  docker pull debian
```

Quelques autres images :

```
[docker@Rocky ~]$  docker pull ubuntu:latest
[docker@Rocky ~]$  docker pull bitnami/minideb
[docker@Rocky ~]$  docker pull alpine
[docker@Rocky ~]$  docker pull docker pull rockylinux:9.0-minimal
```

Liste des images locales :

```
[docker@Rocky ~]$  docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest              978d85d02b87        2 weeks ago         123 MB
```

2. Création d'un conteneur

Créer un conteneur à partir de l'image Debian, nommé « DebCont0 », accessible avec un terminal (-t) interactif (-i)

```
[docker@Rocky ~]$  docker create --name DebCont0 debian
```

Vérification, en visualisant la liste des conteneurs :

```
[docker@Rocky ~]$  docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED             STATUS          PORTS       NAMES
2e2d14b89c2b   debian    "/bin/bash"   4 seconds ago       Created                     DebCont0
```

3. Démarrage de conteneurs

Démarrer un conteneur nommé « ContDeb1 » en tache de fond (option -d), depuis l'image « debian » avec un terminal en mode interactif :

```
[docker@Rocky ~]$  docker run -dti --name ContDeb1 debian
```

L'option **--rm** permet de détruire le conteneur une fois utilisé et stoppé :

```
[docker@Rocky ~]$  docker run --rm --name ContDeb2 debian
```

Exécution d'une commande dans le conteneur Debian déjà démarré :

```
[docker@Rocky ~]$  docker exec -ti ContDeb1 cat /etc/issue
Debian GNU/Linux 11 \n \l
```

Démarrer un conteneur et accéder à son terminal

```
[docker@Rocky ~]$  docker run -ti debian
root@8fcceab47ae2:/# cat /etc/issue
Debian GNU/Linux 11 \n \l
```

Accéder à un conteneur démarré

```
[docker@Rocky ~]$  docker attach ContDeb1
```

Quitter un conteneur

```
root@8fcceab47ae2:/# <CTRL>p <CTRL>q
```

Renommer un conteneur :

```
[docker@Rocky ~]$  docker rename DebCont0 ContDeb0
```

4. Arrêt / suppression de conteneurs / images

Accéder au shell d'un conteneur déjà démarré

```
[docker@Rocky ~]$  docker exec -ti ContDeb1 /bin/bash
```

Arrêt normal d'un conteneur

```
[docker@Rocky ~]$  docker stop ContDeb1
```

Démarrer un conteneur déjà existant en attachant les signaux STDIN/OUT ERR, en mode interactif :

```
[docker@Rocky ~]$  docker start -a -i ContDeb1
```

Arrêt de tous les conteneurs :

```
[docker@Rocky ~]$  docker stop $(docker ps -a -q)
```

Arrêt forcé d'un conteneur

```
[docker@Rocky ~]$  docker kill ContDeb1
```

Effacer un conteneur

```
[docker@Rocky ~]$  docker rm ContDeb1
```

Effacement forcé d'un conteneur (même démarré)

```
[docker@Rocky ~]$  docker rm -f ContDeb1
```

Suppression de tous les conteneurs

```
[docker@Rocky ~]$  docker rm $(docker ps -a -q)
```

Effacer une image

```
[docker@Rocky ~]$  docker rmi -f debian
```

Suppression de toutes les images

```
[docker@Rocky ~]$  docker rmi $(docker images -q)
```

Arrêts / suppressions selon des critères nominatifs :

```
[docker@Rocky ~]$  docker ps -a | grep "\-Pierre" | awk '{print $1}' | xargs docker stop
[docker@Rocky ~]$  docker ps -a | grep "\-Pierre" | awk '{print $1}' | xargs docker rm
```

5. Copie de fichiers

La copie de fichiers locaux dans un conteneur nommé « apache » s'effectue de cette manière :

```
[docker@Rocky ~]$  docker cp index.html apache:/tmp/
```

Mise à jour d'un répertoire (spécifier le point à ma source) :

```
[docker@Rocky ~]$  $ docker cp html/. apache:/tmp/
```

6. Paramètres de démarrage

Création d'un conteneur nginx, connu par Docker sous le nom de ContNgx1, avec un hostname (à l'intérieur du conteneur), limité à 256 Mo de RAM + 256 Mo de swap, et mappage du port externe 8080 vers le port interne 80

```
[docker@Rocky ~]$  docker run -dti --name ContNgx1 --hostname nginx -m 256m --memory-swap 512m -p 8080:80 nginx
```

nginx répond bien à partir du serveur hôte (http://localhost:8080/)

Sur l'ensemble des ports, nous avons ceux numérotés de :

- 1 à 1023, qui sont les ports systèmes, utilisés par des processus qui fournissent les services de réseau les plus répandus.
- 1024 à 49151 : les « Ports enregistrés », qui sont attribués par l'IANA (Internet Assigned Numbers Authority), à l'usage d'application ou matériels spécifiques.
- **49152 à 65536** : ports dynamiques, privés ou éphémères.

Soit on mappe les ports standards 80:80 sans changements (voire 8080:80), soit il faut exposer les ports non système / enregistrés (dernière plage) afin de ne pas empiéter sur des ports applicatifs déjà utilisés ou a venir.

C. Modification des caractéristiques

Nous pouvons modifier quelques paramètres d'un conteneur, à la volée, une fois démarré :

```
[docker@Rocky ~]$  docker update --memory-reservation 16m --memory 32m --memory-swap 64m ContNgx1
[docker@Rocky ~]$  docker update --cpuset-cpus 3 ContNgx1
[docker@Rocky ~]$  docker update --cpuset-cpus 0,2 ContNgx1
[docker@Rocky ~]$  docker update --cpuset-cpus 2-3 ContNgx1
[docker@Rocky ~]$  docker update --cpus 1.5 ContNgx1
[docker@Rocky ~]$  docker update --cpus 0.01 ContNgx1
…
```

`memory-reservation` est la limite « soft » de la mémoire vive allouée
`memory` est la limite « hard » de la RAM allouée (minimum 6 megabytes)
`cpuset-cpus` désigne spécifiquement des vCUPs prioritaires
`cpus` permet d'allouer une fraction sur l'ensemble des processeurs.

- `memory-reservation` doit être inférieur à `memory` qui doit être inférieur ou égal à `memory-swap`
- Si `memory` et `memory-swap` sont de même taille, le conteneur n'aura pas de swap disponible.
- Si `memory-swap` est indéfini ou égal à zéro, son allocation sera de la même taille que `memory` ; autrement dit, il pourait y avoir des échanges avec le stockage physique, et donc des ralentissments.

Vérifications :

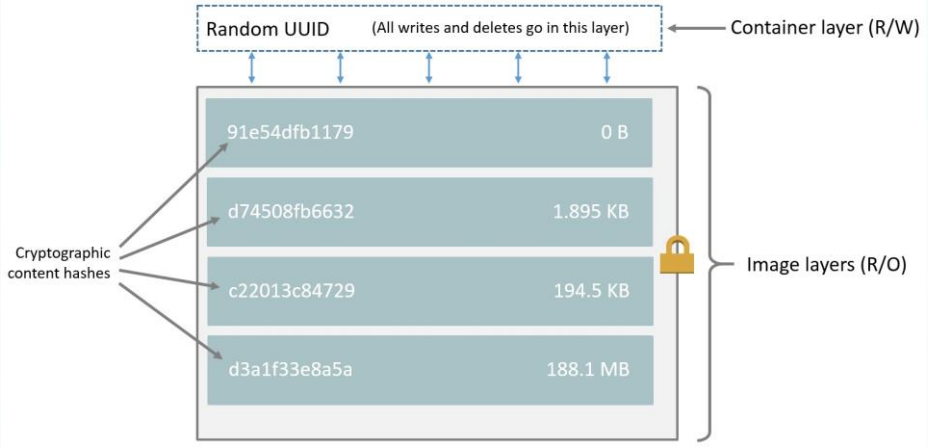
```
[docker@Rocky ~]$  docker stats ContNgx1
[docker@Rocky ~]$  docker inspect ContNgx1 |grep Hostname\"
[docker@Rocky ~]$  docker inspect ContNgx1 |grep \"Memory\"
[docker@Rocky ~]$  docker inspect ContNgx1 |grep MemorySwap\"
```

En fonction des kernels, certaines restrictions de la mémoire ne sont pas prises en charge…

VII. Images & Layers & Containers

A. Modèle en couches

Un conteneur Docker est en réalité une couche (layer) contenant un système de fichiers modifiable, s'appuyant sur plusieurs autres couches en lecture seule, qui se superposent.

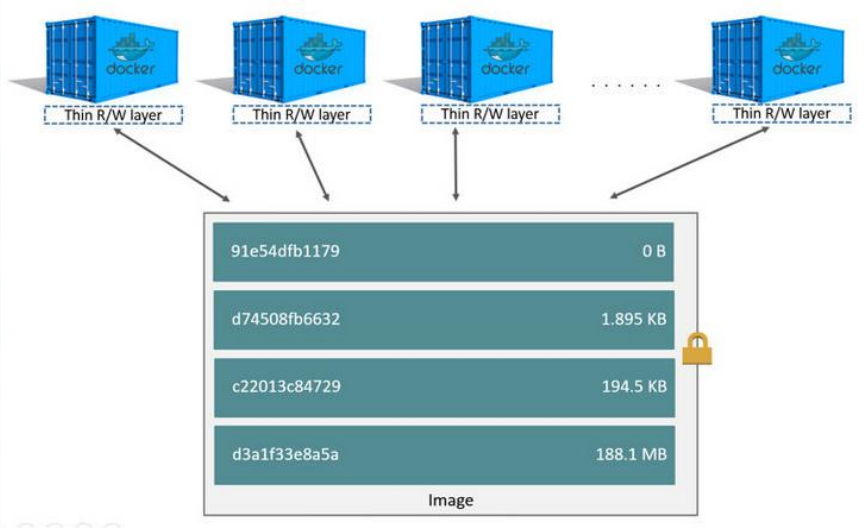


Source : www.docker.com

La liste de l'ensemble des couches formant une image est affichée via

```
[docker@Rocky ~]$  docker history <image>
```

A chaque création d'un nouveau conteneur basé sur une même instance d'image, seule la dernière couche en lecture/écriture est créée, ce qui optimise ainsi les ressources en stockage, et la rapidité d'instanciation d'un nouveau conteneur :



Source : www.docker.com

B. Pilotes et systèmes de fichiers

La gestion de ces couches est assurée par un driver Docker, qui est associé à un type de filesystem.

Il existe plusieurs types de drivers, qui sont pris en charge pas différents systèmes hôtes. Parmi les plus fréquents :

Driver	(sup)porté par
overlay / overlay2 (Linux kernel >= 4.0) / AUFS	ext4 / xfs

Btrfs	Btrfs
devicemapper	LVM
zfs	zfs

Le choix de ces pilotes peut être spécifié soit :

1/ dans la variable DOCKER_OPTS définie dans le fichier /etc/default/docker

2/ avec la commande :

```
[docker@Rocky ~]$  dockerd --storage-driver=devicemapper &
```

C. Conversion d'un conteneur en image

Imaginons que :

- l'on récupère une image Debian
- on créé un conteneur nommé « debian_nginx.updated »
- on le mette à jour, en installant Nginx
- on veuille créer une image nommée « debian/nginx:updated ».

La séquence sera la suivante :

```
[docker@Rocky ~]$  docker pull debian:latest
[docker@Rocky ~]$  docker run -dti --name debian_nginx.updated debian:latest
[docker@Rocky ~]$  docker exec -ti debian_nginx.updated apt-get –y update
[docker@Rocky ~]$  docker exec -ti debian_nginx.updated apt-get install -y nginx
```

On contrôle l'existence d'un différentiel entre l'image d'origine et le conteneur :

```
[docker@Rocky ~]$  docker diff debian_nginx.updated |head -10
...
[docker@Rocky ~]$  docker commit debian_nginx.updated  debian/nginx:updated
```

On contrôle la création de l'image :

```
[docker@Rocky ~]$  docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian/nginx        updated             f3234c725502        6 seconds ago       195 MB
debian              latest              8cedef9d7368        6 weeks ago         123 MB
```

Il est possible de renommer une image :

```
[docker@Rocky ~]$  docker tag debian/nginx:updated debian/nginx:latest
```

D. Options de conversion

Il est possible d'intégrer des options de création d'image :

```
[docker@Rocky ~]$  docker commit -c "EXPOSE 80" debian_nginx.updated  debian/nginx:updated
```

…lors du lancement du conteneur construit à partir de cette image, le port 80 sera exposé.

E. Archivage d'une image

Les images peuvent être archivées (et compressées) :

```
[docker@Rocky ~]$  docker save debian/nginx:updated | gzip > debian_nginx:updated.tar.gz
```

…puis restaurées :

```
[docker@Rocky ~]$  docker load < debian_nginx:updated.tar.gz
```

F. Gestion des images

Supprimer une image spécifique :

```
[docker@Rocky ~]$  docker image rm debian/nginx:updated
```

Suppression des images non utilisées :

```
[docker@Rocky ~]$  docker image prune -a
```

G. Utilisation de dépôt

Nous pouvons créer un compte gratuit sur https://hub.docker.com

Connexion à notre compte :

```
[docker@Rocky ~]$  docker login -u="pierrau" -p="******"
```

Sauvegarde et restauration d'une image :

```
[docker@Rocky ~]$  docker tag pierreau.glpi pierrau/glpi
[docker@Rocky ~]$  docker push pierrau/glpi
[docker@Rocky ~]$  docker pull pierrau/glpi
```

PS : *pierrau* est à remplacer par le login du compte docker hub.

Déconnexion à notre compte :

```
[docker@Rocky ~]$  docker logout
```

D'autres gestionnaires de dépôts existent, tel que Harbor : https://goharbor.io/

VIII. DOCKERFILE

Les Dockerfiles sont des scripts permettant d'automatiser et d'industrialiser la création d'images personnalisées.

A. Les principales commandes

Exemple de script
[docker@Rocky ~]$ **vi Dockerfile**

```
# Only the instructions RUN, COPY, ADD create layers

# Image de référence
 FROM ubuntu:latest

# Données méta
 LABEL maintainer="pierre ROYER" \
       URL="https://pierreau.fr/pro/" \
       version="1.0" \
       environment="production" \
       released="November 28, 2016"

 MAINTAINER admin@pierreau.fr

# Commandes exécuteés pendant la création de l'image
  RUN apt update && apt upgrade -y
  RUN apt install apache2 \
                  wget \
                  curl \
                  openssh-server \
                  vim \
                  tzdata -y

# Fuseau horaire
  ARG DEBIAN_FRONTEND=noninteractive
  ENV TZ=Europe/Paris
  RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone
  RUN dpkg-reconfigure --frontend noninteractive tzdata

# Spécifier de répertoire courant
 WORKDIR /var/www/html/

# Configuration Apache
 RUN  echo 'ServerName apache.pierrau.fr:80' >> /etc/apache2/apache2.conf

# Copie de fichiers du host vers le conteneur
# COPY index.html /var/www/html/
# RUN  cd /var/www/html/
#  RUN  mv index.html index.html.old
#  RUN  wget https://pierreau.fr/_Share/index.html

# Permet l'ajout d'archives ou d'URL
 ADD https://pierreau.fr/index.php /var/www/html

# Points de montage créés dans le conteneur
 VOLUME   ["/var/www/html", "/etc/apache2", "/var/log/apache2/"]

 RUN useradd -m -s /bin/bash -p sshuser sshuser

# Vérifie l'état de santé (pour un serveur web)
 HEALTHCHECK --interval=1m --timeout=10s --retries=5 CMD curl -f
http://localhost/ || exit 1

# Commande ENTRYPOINT (non surchargeable par docker run) lancée au
démarrage du conteneur
# COPY  docker-entrypoint.sh /usr/sbin/
# RUN   chmod +x /usr/sbin/docker-entrypoint.sh
```

```
# ENTRYPOINT ["/usr/sbin/docker-entrypoint.sh"]

# Commande (surchargeable par docker run) lancée au démarrage du conteneur
 CMD ["apachectl", "-D", "FOREGROUND"]

# Affectation d'une valeur à une variable d'environnement (prompt)
 ENV   PS1="\u@\h\w \s-\v\$ "

# Ports réseau ouverts à l'extérieur du conteneur
 EXPOSE 22/tcp 123/udp
```

ENTRYPOINT :

```
[docker@Rocky ~]$  vi docker-entrypoint.sh
```

```
#!/bin/bash
set -e

service ssh restart && bash
# /usr/sbin/zabbix_agentd -f

exec "$@"
```

Génération de la paire de clés SSH ; transfert de la clé publique dans le conteneur :

```
[docker@Rocky ~]$  ssh-keygen -b 2048 -t rsa
[docker@Rocky ~]$  ssh sshuser@172.17.0.3
```

Exemple de fichier HTML :

```
[docker@Rocky ~]$  vi index.html
```

```
<!DOCTYPE html>
<html lang="fr">
      <head>
            <title>Cloud de notre formation priv&eacutee</title>
            <style>
                  body { background-color: darkgrey; color: black;}
                  h1 { color: LightGreen }
            </style>
      </head>
      <body>
            <h1>Le site Internet de notre formation</h1>
            <p>Notre serveur web est fonctionnel !</p>
      </body>
</html>
```

En considérant que le fichier Dockerfile est dans le répertoire courant, la création de l'image nommée « pierreau/ubuntu.apache » via ce fichier s'effectue avec la commande

```
[docker@Rocky ~]$  docker build -t 'pierreau/ubuntu.apache' .
```

On peut spécifier le chemin du dossier d'un autre dockerfile de cette manière :

```
[docker@Rocky ~]$  docker build -f /[path]/Dockerfile .
```

L'image créée sera constituée de plusieurs couches (layers), visibles avec la commande

```
[docker@Rocky ~]$  docker history pierreau/ubuntu.apache
```

```
[docker@Rocky ~]$  docker run -dti --name ubuntu.apache -p 60080:80 -m 16M pierreau/ubuntu.apache
```

B. Git

Si nous utilisons Git, nous pouvons ignorer certains fichiers dans le contexte du « build » :

```
[docker@Rocky ~]$  vi .gitignore
```

```
node_modules
.git
```

C. supervisord

Dans le cas de gestion de services multiples dans un conteneur, une solution existe :

Run multiple services in a container | Docker Documentation

IX. VOLUMES

Un volume permet à des conteneurs de stocker des informations persistantes. Il peut être partagé entre plusieurs containers.

A. Création d'un volume

Création d'un volume :

```
[docker@Rocky ~]$  docker volume create DataVolume1
```

Vérifications :

```
[docker@Rocky ~]$  docker volume ls
DRIVER              VOLUME NAME
local               DataVolume1

[docker@Rocky ~]$  docker volume inspect DataVolume1
```

Partage de ce volume dans deux dossiers « ExtVolume » montés dans 2 conteneurs « DebCont1 » (effacé automatiquement lors de son arrêt), et « DebCont2 » :

```
[docker@Rocky  ~]$     docker  run  -ti  --name  DebCont1  --hostname  cont1  -v
DataVolume1:/ExtVolume debian
root@cont1:/# ls ExtVolume
root@cont1:/# touch ExtVolume/test
root@cont1:/# ls ExtVolume
test
<CTRL P> <CTRL Q>
[docker@Rocky ~]$  docker stop DebCont1

[docker@Rocky  ~]$     docker  run  -ti  --name  DebCont2  --hostname  cont2  -v
DataVolume1:/ExtVolume debian
root@cont2:/# ls ExtVolume
test
```

B. Création à la volée

Il est possible de créer un volume persistant dès la création ou le démarrage d'un conteneur :

```
[docker@Rocky ~]$  docker create --name DebCont0 -t -i -v DataVolume1:/ExtVolume
debian

[docker@Rocky ~]$  docker run -ti --name DebCont1 -v DataVolume1:/ExtVolume debian
```

C. Droits sur les conteneurs

La lecture seule sur un volume est assurée par le paramètre read-only **:ro**

```
[docker@Rocky ~]$  docker run -ti -name DebCont1 -v DataVolume1:/ExtVolume:ro debian
```

D. Gestion des volumes

Supprimer un volume :

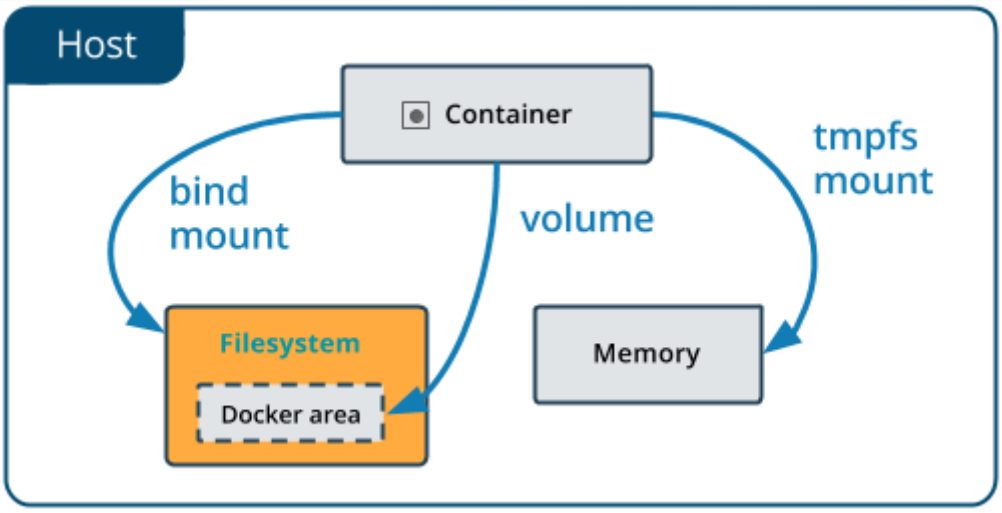
```
[docker@Rocky ~]$  docker volume rm DataVolume1
```

Suppression des volumes non utilisés par des conteneurs :

```
[docker@Rocky ~]$  docker volume prune
```

E. Bind et tmpfs

D'autres types de volumes existent : ceux montés dans un répertoire de l'hôte (bind), et ceux qui sont volatiles car situés dans la ram (tmpfs)



Source : https://docs.docker.com/storage/tmpfs/

1. Montage d'un répertoire sur l'hôte

```
[docker@Rocky ~]$  mkdir ~/web-directory
[docker@Rocky ~]$  touch ~/web-directory/index.html

[docker@Rocky ~]$  docker run -dti --name web-server2 -v /home/docker/web-directory/HTML:/var/www/html -p 60081:80 httpd:latest

[docker@Rocky ~]$  docker exec web-server ls /var/www/html/
index.html
```

Afin de spécifier un répertoire relatif en non abolu, la syntaxe est la suivante :

```
[docker@Rocky ~]$ docker run -dti --name web-server2 -v $(pwd)/HTML:/var/www/html -p 60081:80 httpd:latest
```

2. Montage bind d'un fichier sur l'hôte

```
[docker@Rocky ~]$  mkdir ~/log
[docker@Rocky ~]$  touch ~/log/access.log

[docker@Rocky ~]$ docker run -dti --name web-server2 --mount type=bind,source="$(pwd)"/log/access.log,target=/var/log/apache2/access.log -p 60081:80 httpd:latest
```

3. Montage bind avec docker-compose

```
---
version: '3.7'
volumes:
  site:
```

```
    name: moodle-web
services:
  moodle
    image: httpd:latest
    volumes:
      - type: bind
        source: ./moodle
        target: /var/www/html/moodle
```

4. Montage d'un volume dans la mémoire vive (taille en octects)

```
[docker@Rocky ~]$ docker run -ti --name volram -m 1000m --mount type=tmpfs,destination=/tmpfs,tmpfs-size=1000000000 ubuntu:latest
[docker@Rocky ~]$ docker exec -ti volram

root@7d85f516ed88:/# cp -r /* /tmpfs/
...
```

Pour un volume en RAM, je conseille très vivement de limiter l'utilisation de la mémoire du conteneur (`--memory`).

Ce type de volume ne peut pas être partagé avec d'autres conteneurs.

X. PORTAINER

OK, maintenant que l'on maitrise Dockerfile et les volumes, il est temps de mettre en action des interfectes web (dans un conteneur) pour la supervisison de nos ressources.

http://portainer.io/

On créé un volume persistant :

```
[docker@Rocky ~]$ docker volume create portainer_data
```

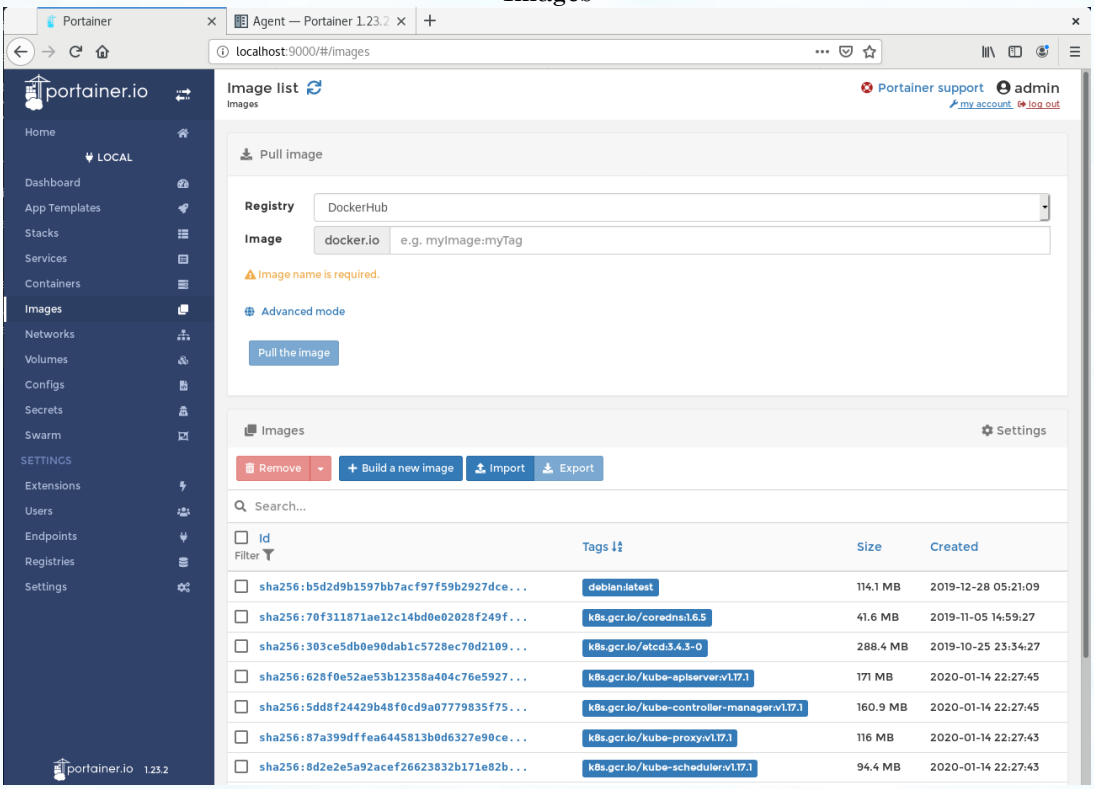
On installe un nouveau conteneur (portainer) qui va stocker ses informations dans le volume créé :

```
[docker@Rocky ~]$ docker run -d --name=portainer -m 32M -p 8000:8000 -p 9000:9000 --cpuset-cpus="0" --restart=on-failure -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

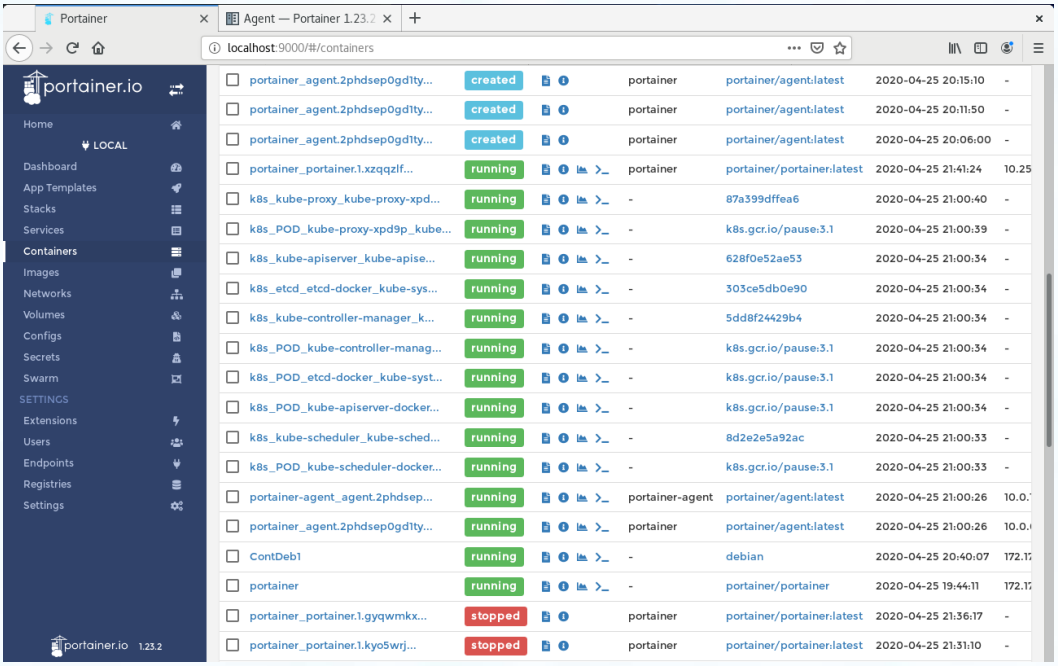
L'accès s'effectue à l'URL http://*localhost*:9000/

Interace simple et fonctionnelle, permet de gérer les images, conteneurs, réseaux, volumes, services, stacks… :

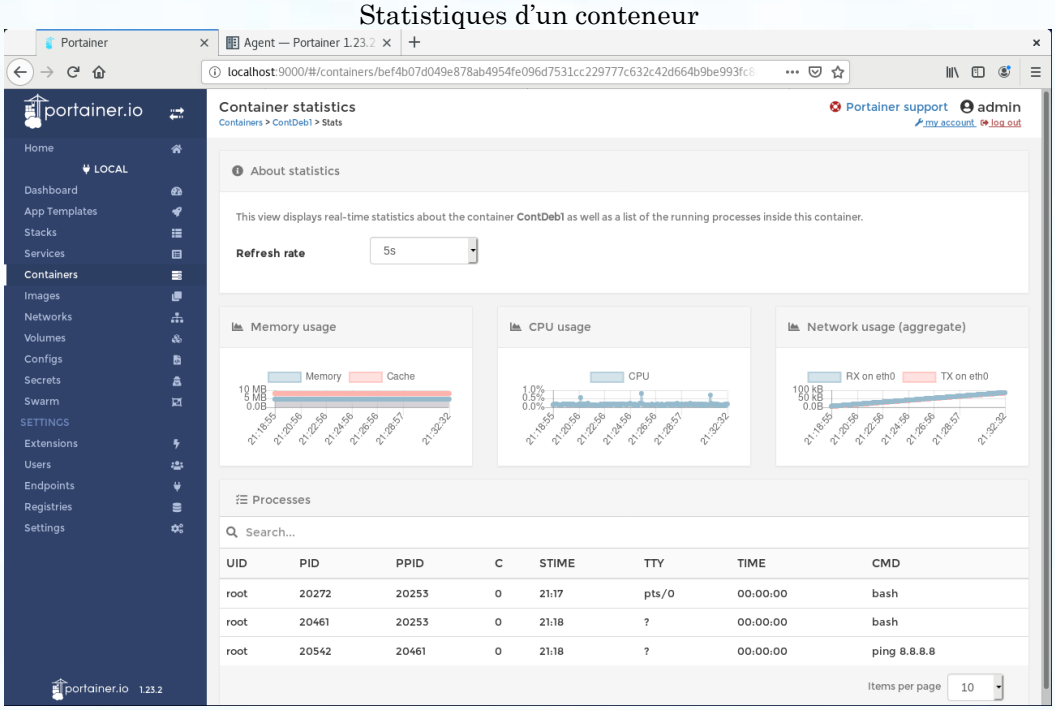
Images



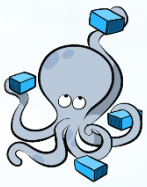
Conteneurs



Statistiques d'un conteneur



XI. DOCKER-COMPOSE



Docker Compose est un outil pour définir et exécuter des applications Docker multi-conteneurs interdépendants. Il utilise :

- La définition de l'environnement via un Dockerfile
- La définition des services contenus dans un fichier « docker-compose.yml »
- La commande « docker-compose up » pour démarrer l'environnement isolé

A. Installation

Identifier la dernière version en date sur l'URL https://github.com/docker/compose/releases/

Définir la variable d'environnement en fonction de la dernière version :

```
[docker@Rocky ~]$  dockerComposeVersion="v2.11.2"
```

Installer Docker Compose :

```
[docker@Rocky ~]$  dnf install curl
[docker@Rocky ~]$  curl -L
https://github.com/docker/compose/releases/download/$dockerComposeVersion/d
ocker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

Définir les droits d'execution :

```
[docker@Rocky ~]$  chmod +x /usr/local/bin/docker-compose
```

B. docker-compose.yml

La syntaxe d'un fichier docker-compose doit respecter les spécifications YALM :

- Le fichier débute par trois tirets
- L'indentation est homogène : ici 2 espaces
- Pas de tabluation à la place des espaces.

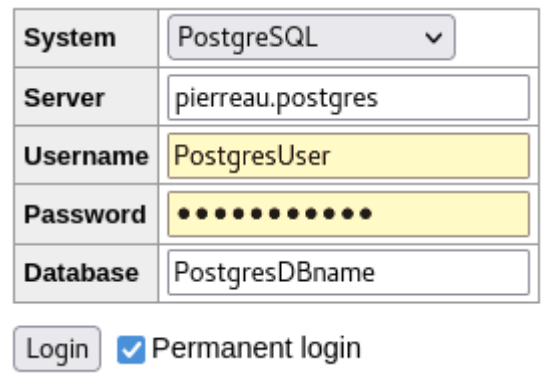
Exemple de fichier au standard YAML :

```
[docker@Rocky ~]$  vi docker-compose.yml
```

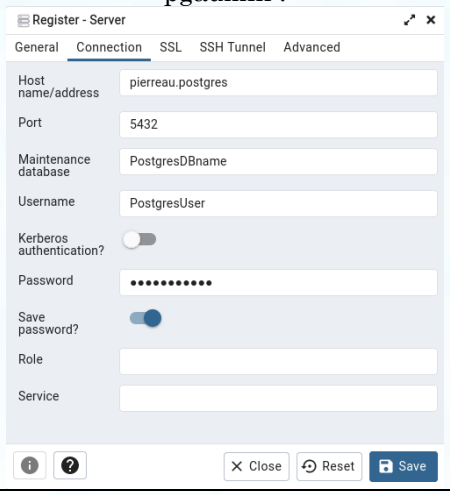
```
---
version: '3.7'

volumes:
  Postgres-Data:
    name: Postgres-data
    driver: local

services:
  postgres:
    image: postgres:latest
    container_name: pierreau.postgres
    hostname: pgadmin
    restart: always
    ports:
```

```
      - "5432:5432"
    environment:
      POSTGRES_USER: PostgresUser
      POSTGRES_PASSWORD: PostgresPWD
      POSTGRES_DB: PostgresDBname
    volumes:
      - Postgres-Data:/var/lib/postgresql/data

  pgadmin:
    image: dpage/pgadmin4
    container_name: pierreau.pgadmin
    depends_on:
      - postgres
    restart: on-failure
    ports:
      - "55320:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: user@pierreau.fr
      PGADMIN_DEFAULT_PASSWORD: SuperSecret

  adminer:
    image: adminer:latest
    container_name: pierreau.adminer
    depends_on:
      - postgres
    restart: on-failure
    environment:
      ADMINER_DEFAULT_DB_DRIVER: postgresql
      ADMINER_DEFAULT_SERVER: pierreau.postgres
      ADMINER_DEFAULT_DB_NAME: PostgresDBname
      ADMINER_PLUGINS: dump-zip dump-bz2 dump-date
      ADMINER_DESIGN: galkaev # nette
    ports:
      - "55321:8080"
```

Adminer :



pgadmin :



Visualisation des paramètres :

```
[docker@Rocky ~]$  docker-compose config
```

Installation de la pile :

```
[docker@Rocky  ~]$    docker-compose  -p  "mon-application"  -f  ./docker-compose.yml up -d
```

Liste des conteneurs :

```
[docker@Rocky ~]$  docker-compose -f ./docker-compose.yml ps
```

Liste des images utilisées :

```
[docker@Rocky ~]$ docker-compose images
```

Utiliser le nom de la pile, si elle à été définie précédemment :

```
[docker@Rocky ~]$  docker-compose -p "mon-application" images
[docker@Rocky ~]$  docker-compose -p "mon-application" ps
```

C. La sécurité

Plusieurs mécanismes co-existent : fichiers de variables (cf. chapitre suivant), réseaux internes (cf. chapitre à suivre…). Les bonnes pratiques (recommandées par la CNIL entres-autre) passent pas la limitation des ressources d'un conteneur. A la fois pour protéger les autres conteneurs, et pour limiter les catastrophes sur l'hôte.

```
[docker@Rocky ~]$  vi docker-compose.yml
```

```
---
version: '3.9'

services:
  postgres:
    image: postgres:latest
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: PostgresUser
      POSTGRES_PASSWORD: PostgresPWD
      POSTGRES_DB: PostgresDBname

    deploy:
      resources:
        limits:
          cpus: '0.3'
          memory: 512M
        reservations:
          cpus: '0.2'
          memory: 256M
```

Il est vital de prendre en considération ces limitations en production.

D. docker-compose + Dockerfile

```
[docker@Rocky ~]$  vi Dockerfile
```

```
# Image de référence
 FROM nginx:alpine

# Copie de fichiers du host vers le conteneur
 COPY nginx.conf /etc/nginx/nginx.conf
```

```
[docker@Rocky ~]$ vi docker-compose.yml
```

```
---
version: '3.7'

services:
  monservice:
```

```
    container_name: dockerfile
    build:
      context: ./
      dockerfile: ./Dockerfile
      labels:
        - "URL="https://cloud.pierreau.fr/"
    image: pierreau/mon-image:v1.0
    ports:
      - 55555:80
...
```

```
[docker@Rocky ~]$ docker-compose up --build
[docker@Rocky ~]$ docker run -ti dockerfile
```

E. Arrêt et désinstallation

Arrêt de l'environnement :

```
[docker@Rocky ~]$  docker-compose -f ./docker-compose.yml stop
```

Suppression de la pile :

```
[docker@Rocky ~]$  docker-compose -f ./docker-compose.yml down
```

Supprimer l'ensemble des conteneurs crées avec docker-compose :

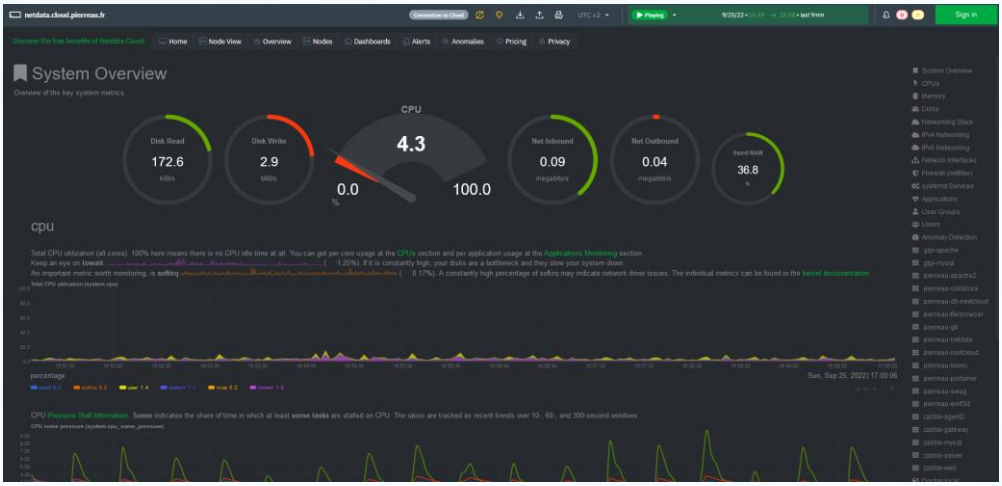
```
[docker@Rocky ~]$  docker-compose rm
```

XII. NETDATA

OK, maintenant que l'on maîtrise les volumes et docker-compose, je vous propose un autre outil de supervision : https://github.com/netdata/netdata

Netdata est un outil open source conçu pour collecter des mesures en temps réel, telles que l'utilisation du processeur, l'activité du disque, l'utilisation de la bande passante, les visites de sites du serveur Host et de ses conteneurs

Cet outil est conçu pour visualiser l'activité du système et des conteneurs dans le plus grand détail possible.




```
[docker@Rocky ~]$  vi docker-compose.yml
```

```
---
version: '3.7'
services:
  netdata:
    image: netdata/netdata
    container_name: netdata

    ports:
      - 19999:19999
    # PGID est obtenu avec : grep docker /etc/group | cut -d ':' -f 3
    environment:
      - PGID=120
    deploy:
      resources:
        limits:
          cpus: '0.3'
          memory: 256M
        reservations:
          cpus: '0.2'
          memory: 128M
    restart: on-failure
    cap_add:
      - SYS_PTRACE
    security_opt:
      - apparmor:unconfined
    volumes:
      - netdataconfig:/etc/netdata
      - netdatalib:/var/lib/netdata
      - netdatacache:/var/cache/netdata
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /etc/passwd:/host/etc/passwd:ro
      - /etc/group:/host/etc/group:ro
      - /proc:/host/proc:ro
```

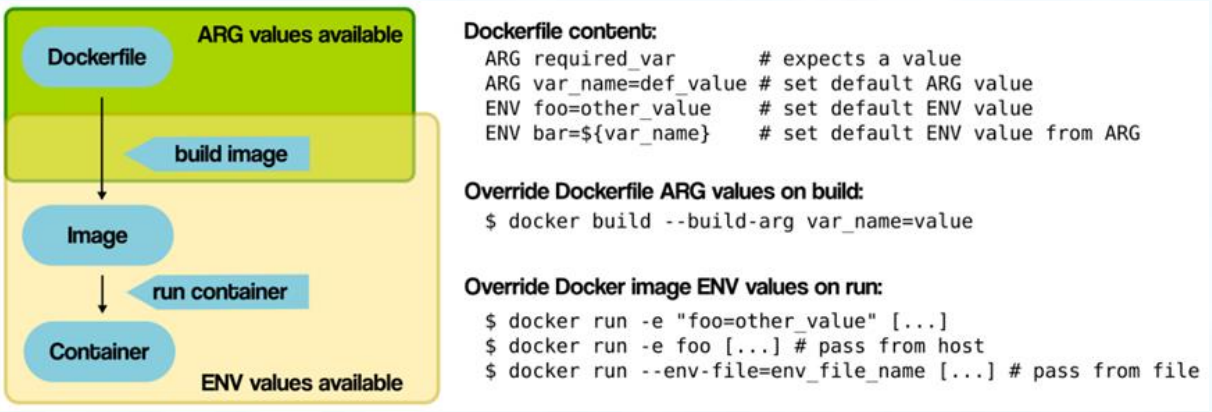
```
      - /sys:/host/sys:ro
      - /etc/os-release:/host/etc/os-release:ro

volumes:
  netdataconfig:
    name: netdata-config
  netdatalib:
    name: netdata-lib
  netdatacache:
    name: netdata-cache
```

```
[docker@Rocky ~]$  docker-compose up
```

XIII. LES VARIABLES ARG ET ENV

ARG est utilisé pour construire nos images docker (docker build). Ces variables n'ont pas de persistance dans le conteneur, mais peuvent être visibles avec la commande docker history
Attention, pas de mots de passe ou information sensible dans ce type de variable !

Les variables ENV font partie de l'image, et vont persister dans les conteneurs.



A. ENV

Les variables ENV sont prises en charge par ces instructions :

- ADD, COPY, ENV, EXPOSE, FROM, LABEL, STOPSIGNAL, USER, VOLUME, WORKDIR, ONBUILD

1. docker run

```
[docker@Rocky ~]$ vi ./.env
```

```
MYFOLDER=/root

# Format non pris en compte dans le conteneur (mal interprété) :
# MYDATE=`date +%A,%d/%m/%Y-%H:%M`
```

```
[docker@Rocky ~]$ export MYVAR1=$HOSTNAME
[docker@Rocky ~]$ export MYDATE=`date +%A,%d/%m/%Y-%H:%M`

[docker@Rocky ~]$ docker run -dti --name pierreau.env --env-file ./.env --
env MYDATE -e MYVAR1 -e MYVAR3=Pierre alpine:latest sh

[docker@Rocky ~]$ docker exec -ti pierreau.env sh

root@2640a1a9450c:/# echo $MYFOLDER
/root
root@2640a1a9450c:/# echo $MYDATE
Monday,12/12/2022-12:12
root@2640a1a9450c:/# echo $MYVAR3
Pierre
```

2. Les fichiers d'environnements / docker-compose

Certains paramètres peuvent être centralisés sous forme de variables dans un fichier caché (**.env**), pour des raisons de sécurité, entres-autres.

Ce fichier est par défaut stocké dans le même répertoire que **docker-compose.yml**

```
[docker@Rocky ~]$ vi .env
```

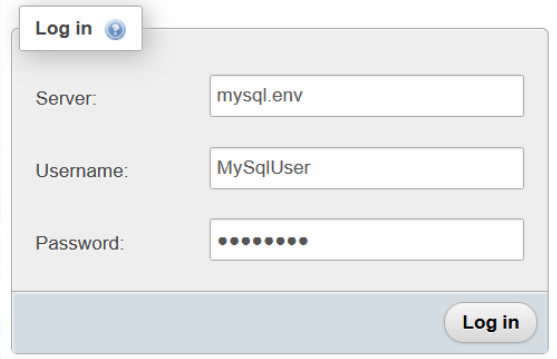
```
MYSQL_ROOT_PASSWORD=MySqlRootPwd
MYSQL_USER=MySqlUser
MYSQL_PASSWORD=MySqlPwd
MYSQL_DATABASE=MySqlDB
```

```
[docker@Rocky ~]$ vi docker-compose.yml
```

```
---
version: '3.7'

services:

  mysql:
    image: mysql:oracle
    container_name: mysql.env
    restart: always
    ports:
      - 3306:3306
    env_file:
      - .env
    # Optionnel si la syntaxe les variables .env sont identiques
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
    volumes:
      - ./config/mysql/my.cnf:/etc/mysql/conf.d/my.cnf
      - mysqldata:/var/lib/mysql

    healthcheck:
      test: mysqladmin ping -h 127.0.0.1 -u root --
password=$$MYSQL_ROOT_PASSWORD
     # test: "/usr/bin/mysql --user=root --password=$$MYSQL_ROOT_PASSWORD
--execute \"SHOW DATABASES;\""
     # test: mysql ${MYSQL_DATABASE} --user=${MYSQL_USER} --
password='${MYSQL_PASSWORD}' --silent --execute "SELECT 1;"
      interval: 1m
      timeout: 10s
      retries: 5

  phpmyadmin:
    image: phpmyadmin:latest
    container_name: phpmyadmin.env
    restart: on-failure:3
    ports:
      - 8080:80
    environment:
      - PMA_ARBITRARY=1
volumes:
  mysqldata:
```

Vérifier la prise en charge des variables !

Vérifier l'adresse du serveur mariadb…



Le fichier des variables peut être spécifié dans la création du stack :

```
[docker@Rocky ~]$ docker-compose --env-file .env config ...
```

Les variables sont récupérables dans le conteneur !

```
[docker@Rocky ~]$ docker exec -ti mysql.env sh
sh-4.4# env
MYSQL_PASSWORD=MySQLPwd
HOSTNAME=da4655740525
MYSQL_DATABASE=MySqlDB
MYSQL_ROOT_PASSWORD=MyRootPwd
```

Utilisation avec git :

Certains fichiers d'environnement peuvent contenir des mots de passe, et il peut être plus sûr de les garder localement. Dans ce cas, nous pouvons créer un fichier qui va omettre le transfert et le partage d'informations sensibles vers le dépôt :

```
[docker@Rocky ~]$ vi .gitignore
```

```
.env
```

B. ARG

1. Dockerfile

```
[docker@Rocky ~]$ MYDATE=`date +%A,%d/%m/%Y-%H:%M`
[docker@Rocky ~]$ export MYDATE
```

```
[docker@Rocky ~]$ vi Dockerfile
```

```
FROM   alpine:latest
 ARG FOLDER=/
 ARG MYDATE
 RUN echo "Date : ${MYDATE}"
 RUN echo "Dossier de travail : ${FOLDER}"
 WORKDIR ${FOLDER}
 RUN echo "Date de l'image : ${MYDATE}" > /DATE.txt
```

```
[docker@Rocky ~]$ docker build --build-arg MYDATE -t 'pierreau:arg' .
```

```
[docker@Rocky ~]$ docker run -dti --name pierreau.arg pierreau:arg sh

[docker@Rocky ~]$ docker exec -ti pierreau.arg ls
DATE.txt ...

[docker@Rocky ~]$ docker exec -ti pierreau.arg cat DATE.txt
Monday,12/12/2022-12:13
```

2. docker-compose

```
[docker@Rocky ~]$ vi docker-compose.yml
```

```
---
version: '3.7'

services:
  pierreau_env:
    container_name: pierreau.env
    build:
      context: .
      dockerfile: Dockerfile
      args:
        FOLDER: /tmp
```

```
[docker@Rocky ~]$ docker-compose up -p "mon-application" --build
[docker@Rocky ~]$ docker run -ti pierreau.env
```

Les informations sensibles ne sont pas à gérer dans les variables ARG car elles sont lisibles via cette commande :

```
[docker@Rocky ~]$ docker image history pierreau.env
IMAGE          CREATED        CREATED BY                                      SIZE
COMMENT
25e2e9b845fc   23 hours ago   |2 FOLDER=/root MYDATE=Tuesday,05/04/2022-14…   43B
2a7d25209770   23 hours ago   /bin/sh -c #(nop) WORKDIR /root                 0B
ab0745ac7f46   23 hours ago   |2 FOLDER=/root MYDATE=Tuesday,05/04/2022-14…   0B
```

XIV. LES RÉSEAUX

A. Créer un réseau

1. Reseau classique

```
[docker@Rocky ~]$  docker network create -d bridge --subnet 172.18.0.0/16 --ip-range=172.18.137.0/24     --subnet=2a01:e0a:172:3910::/64     --gateway 172.18.137.1 --ipv6 frontend
[docker@Rocky ~]$  docker network ls
```

2. Réseau Internal

Les réseaux virtuels de Docker permettent des découpages en différentes zones, et prennent en charge IPV6. De manière à protéger nos informations (ransomware…), il est très fortement recommandé d'utiliser de réseaux isolés (**--internal**), non exposés sur des ports externes, pour les bases de données.

```
[docker@Rocky ~]$  docker network create --subnet=172.19.0.0/16 --internal backend
```

Utiliser des reseaux déjà existant :

```
---
version: '3.7'

networks:
  frontend:
    external: true
    name: frontend
  backend:
    external: true
    name: backend

volumes:
  MariaDB.Data:
    name: mariadb.data
    driver: local

services:

  phpmyadmin:
    image: phpmyadmin:latest
    container_name: pierreau.phpmyadmin
    depends_on:
      - mariadb
    restart: on-failure
    environment:
      - PMA_ARBITRARY=1
    networks:
      - frontend
      - backend
    ports:
      - "8080:80"

  mariadb:
    image: mariadb:latest
    container_name: pierreau.mariadb
    restart: always
    networks:
```

```
    - backend
    expose:
      - "3306"
    environment:
      MYSQL_ROOT_PASSWORD: MariaDBpwd
      MYSQL_DATABASE: MariaDBdb
      MYSQL_USER: MariaDBuser
      MYSQL_PASSWORD: MariaDBpwd
    volumes:
      - MariaDB.Data:/var/lib/mysql
```

Vérifier les ports (non) exposés avec la commande :

```
[docker@Rocky ~]$  netstat -tulnp |sort -k 5,5
```

3. Création à la volée

```
---
version: '3.7'

# Création à la volée
networks:
  myfrontend:
    name: myfrontend
    driver: bridge
  mybackend:
    name: mybackend
    driver: bridge
    internal: true
```

B. IP fixes

Bien qu'il soit souvent inutile de créer des conteneurs avec des adresses IP fixes, cela peut être intéressant dans certains cas (conteneur zabbix-agent2 par exemple).

1. Dans un réseau existant

```
---
version: '3.7'
```

```
networks:
  myfrontend:
    name: myfrontend
    external: true

services:
  nginx:
    image: nginx:latest
    container_name: pierreau.nginx
    networks:
      myfrontend:
        ipv4_address: 172.22.0.3
        ipv6_address: 2001:3984:3989::10
    ports:
     - "8080:80"
    environment:
     - NGINX_HOST=nginx.pierreau.fr
     - NGINX_PORT=80
    volumes:
     - ./templates:/etc/nginx/templates
    dns:
      - 1.1.1.1
    extra_hosts:
      - "gougleu:8.8.8.8"
```

2. Dans un nouveau réseau

```
---
version: '3.7'

networks:
# Création d'un réseau à la volée
  newnetwork:
    ipam:
      driver: default
      config:
        - subnet: 172.22.0.0/24
        - subnet: 2001:3984:3989::/64

services:
  nginx:
    image: nginx:latest
    container_name: pierreau.nginx
    networks:
      newnetwork:
        ipv4_address: 172.22.0.3
        ipv6_address: 2001:3984:3989::10
```

XV. DOCKER MACHINE

Docker-machine est un outil qui permet de provisionner des machines physiques ou virtuelles, afin d'héberger des conteneurs. Ces machines auront les rôles de « Manager » et « Worker ».

Ce projet n'évolue plus, et est obsolète !

A. Installation

Vérifier la dernière version sur Github : https://github.com/docker/machine/releases/

```
[docker@Rocky ~]$  base=https://github.com/docker/machine/releases/download/v0.16.2
[docker@Rocky ~]$  curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine
[docker@Rocky ~]$  mv /tmp/docker-machine /usr/local/bin/docker-machine
[docker@Rocky ~]$  chmod +x /usr/local/bin/docker-machine
```

Un outil d'aide aux commandes (completion) peut aussi être intallé :

```
[docker@Rocky ~]$  curl -L
https://raw.githubusercontent.com/docker/machine/v0.16.0/contrib/completion
/bash/docker-machine.bash -o /etc/bash_completion.d/docker-machine
[docker@Rocky ~]$  exec $SHELL -l
```

On vérifie la version installée :

```
[docker@Rocky ~]$  docker-machine version
```

La liste des options disponibles s'affiche avec la commande :

```
[docker@Rocky ~]$  docker-machine
```

B. Création d'un cluster VirtualBox

1. Installer virtualbox

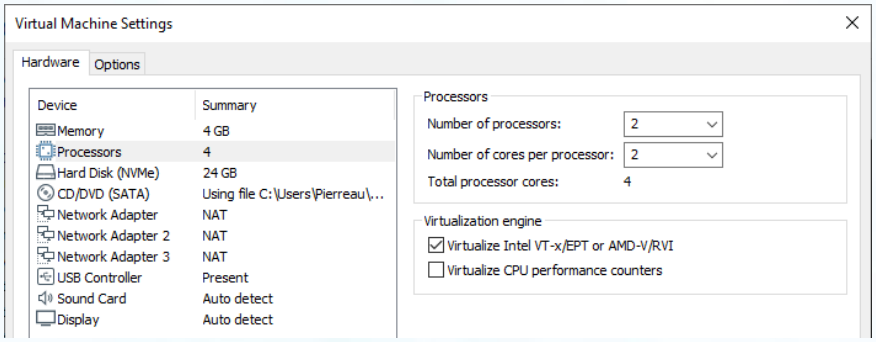
https://www.virtualbox.org/wiki/Linux_Downloads

Virtualbox est installé pour héberger les nodes :

```
[docker@Rocky ~]$  dnf install binutils kernel-devel kernel-headers libgomp make
patch gcc glibc-headers glibc-devel
[docker@Rocky ~]$ dnf config-manager --add-
repo=https://download.virtualbox.org/virtualbox/rpm/el/virtualbox.repo
[docker@Rocky ~]$  dnf update
[docker@Rocky ~]$  dnf repolist -v
[docker@Rocky ~]$  dnf search virtualbox
[docker@Rocky ~]$  rpm --import https://www.virtualbox.org/download/oracle_vbox.asc
[docker@Rocky ~]$  dnf search virtualbox
[docker@Rocky ~]$  dnf install VirtualBox-6.1
[docker@Rocky ~]$  dnf install https://dl.fedoraproject.org/pub/epel/epel-release-
latest-8.noarch.rpm -y
[docker@Rocky ~]$  dnf install dkms
[docker@Rocky ~]$  /sbin/vboxconfig
[docker@Rocky ~]$  VBoxManage --version

[docker@Rocky ~]$  VirtualBox
```

2. Création de 3 machines

Vérifier que VT-X/AMD-v soient activés sur la machine virtuelle VMware :



L'opération consiste à mettre à disposition 1 Manager et 2 Workers :

```
[docker@Rocky ~]$  docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-memory "2048" --virtualbox-disk-size "25000" manager

[docker@Rocky ~]$  docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-memory "2048" --virtualbox-disk-size "25000" worker1

[docker@Rocky ~]$  docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-memory "2048" --virtualbox-disk-size "25000" worker2
```

Contrôle :

```
[docker@Rocky ~]$  docker-machine ls
```

Mises à jour des machines :

```
[root@localhost ~]# docker-machine upgrade manager
[root@localhost ~]# docker-machine upgrade worker1
[root@localhost ~]# docker-machine upgrade worker2
```

Récupérer l'adresse IP du manager :

```
[docker@Rocky ~]$  docker-machine ip manager
IP-Manager
```

Cette adresse va être utilisée pour initialiser un cluster Swarm, en se connectant en SSH à la machine « Manager » :

```
[docker@Rocky ~]$  docker-machine ssh manager
docker@manager:~$ docker swarm init --advertise-addr IP-Manager
```

Cette commande créée un « Node », défini le rôle de la machine « manager », et renvoie un token à utiliser pour déployer les workers. Copier la ligne générée avec la commande « docker swarm join... »

Se connecter sur les Workers, et coller cette commande afin de définir leurs rôles :

```
[docker@Rocky ~]$  docker-machine ssh worker1
docker@worker1:~$ docker swarm join --token abcd1234 IP-Manager:2377
docker@worker1:~$ exit

[docker@Rocky ~]$  docker-machine ssh worker2
```

```
docker@worker2:~$ docker swarm join --token abcd1234 IP-Manager:2377
docker@worker2:~$ exit
```

Déployer un conteneur nginx dans une machine :

```
[docker@Rocky ~]$  docker run --name web01 -d --hostname worker1 nginx
[docker@Rocky ~]$  docker ps
```

Arrêt d'une machine

```
[docker@Rocky ~]$  docker-machine stop worker1
```

Démarrage

```
[docker@Rocky ~]$  docker-machine start worker1
```

Définir la machine active

```
[docker@Rocky ~]$  docker-machine env worker1
```

Connaître la machine active

```
[docker@Rocky ~]$  docker-machine active
```

Suppression d'une machine

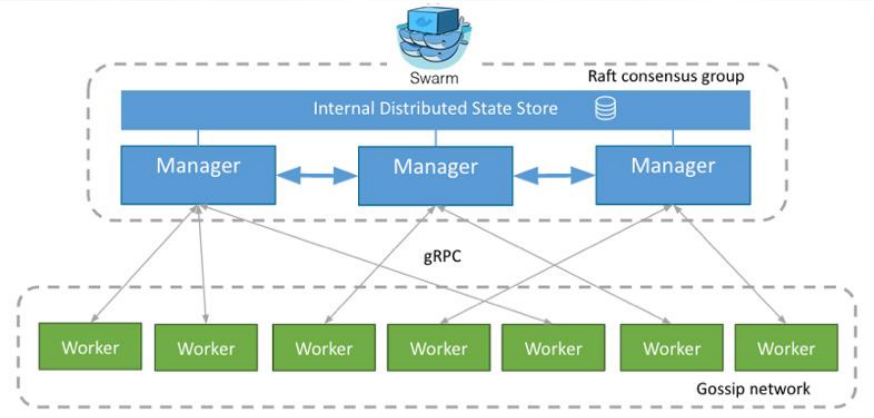
```
[docker@Rocky ~]$  docker-machine rm worker1
```

XVI. SWARM

A. Vue d'ensemble

Swarm est l'orchestrateur intégré nativement dans Docker depuis la version 1.12. Swarm repose sur les notions de nœuds « Manager », Manager « leader » (ou primaire), et « Worker ».

Le Manager maintient l'état du cluster et orchestre les tâches (tasks) sur les noeuds de type worker
Le Worker reçoit les tâches envoyées par les noeuds de type manager, et exécute les tâches.



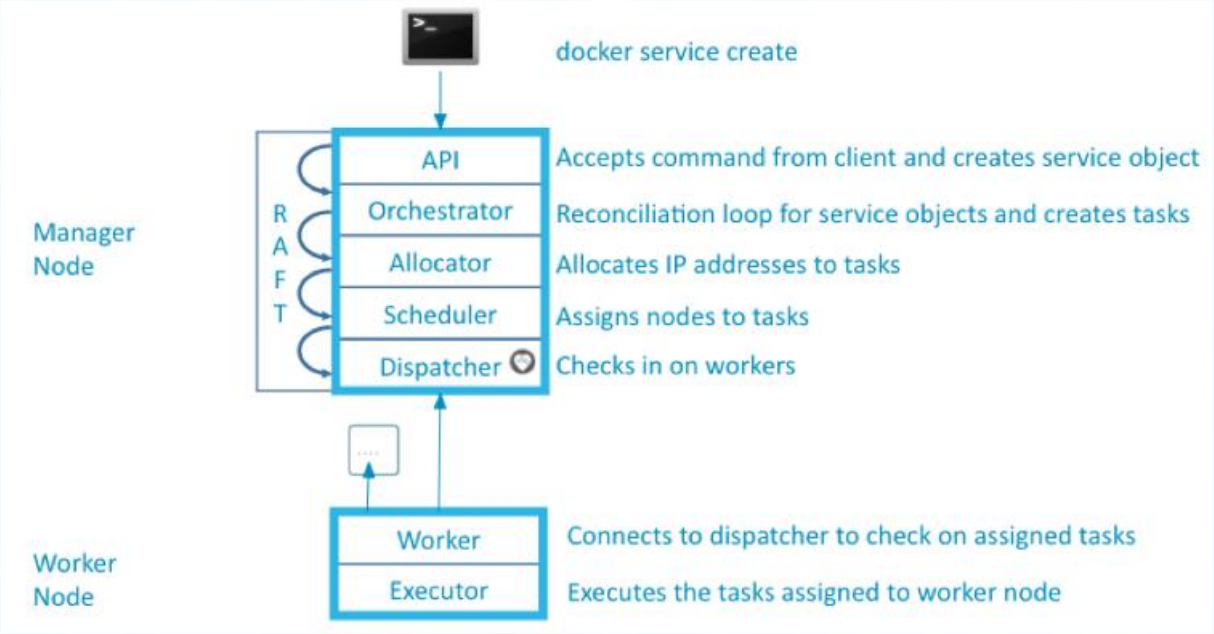
Le groupe de consensus **Raft** est un quorum permettant la haute disponibilité entre les nœuds managers. Les noeuds worker communiquent avec le(s) nœud(s) Manager(s) en utilisant le protocole **gRPC**, basé sur HTTP/2. Cette communication permet au Manager la gestion du versionning des conteneurs (un même niveau de version n'est pas obligatoire), la progression de leurs tâches, et un contrôle de disponibilité des Workers.

Le rôle des noeuds étant dynamique, si un Manager s'arrête de fonctionner, un Worker peut automatiquement basculer en Manager afin d'en assurer une redondance.

B. Briques techniques

Dans le vocabulaire Docker, un service correspond à un conteneur déployé dans un cluster. Le Manager gère le cycle de vie de ces services (déploiement / exécution / résilience).

Gestion des services



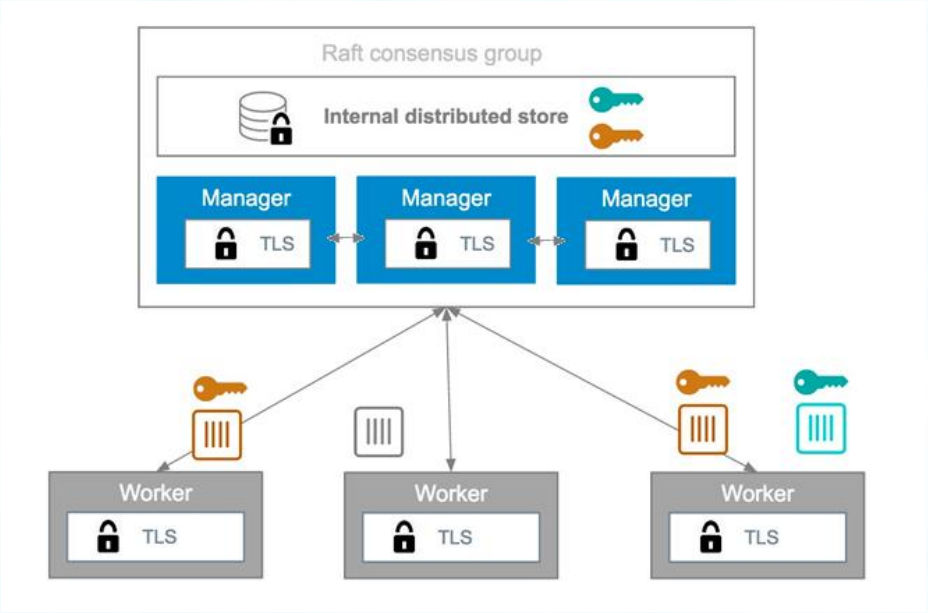
En cas de perte d'un nœud,

- Le module Dispatcher le détecte
- l'Orchestrator réconcilie l'état du cluster avec le nombre d'instances prévu initialement
- le module Allocator alloue des ressources nécessaires pour instancier le nouveau nœud.

C. Réseau virtuel et sécurité

1. Certificats

L'ensemble des échanges dans un cluster Docker est sécurisé nativement, par la création et le déploiement automatique de certificats TLS entre les conteneurs.



2. Docker network

Swarm va créer un réseau virtuel spécifique afin d'interagir avec ses conteneurs :

```
[docker@Rocky ~]$  docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
969b95ed19c0        bridge              bridge              local
ee931dcac4ec        docker_gwbridge     bridge              local
7eb375f5aa6f        host                host                local
670641df8c2e        none                null                local
```

```
[docker@Rocky ~]$  ifconfig -a
docker_gwbridge: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.18.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
```

Il existe 4 types de réseaux spécifiques, afin d'interagir avec les conteneurs :

- **bridge** : crée un réseau interne pour les conteneurs (NAT)
- **host** : ce type de réseau permet au conteneur d'avoir la même interface que l'hôte
- **overlay** : réseau interne entre plusieurs hôtes
- **none** : comme le nom l'indique, aucun réseau pour les conteneurs.

La création d'un réseau dédié n'est pas compliquée :

```
[docker@Rocky ~]$  docker network create --driver bridge SwarmNetwork
[docker@Rocky ~]$  docker network ls
[docker@Rocky ~]$  docker network inspect SwarmNetwork
```

Les paramètres du réseau créé peuvent être spécifiés :

```
[docker@Rocky ~]$  docker network create --driver=bridge --subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --gateway=172.28.5.254 SwarmNetwork

[docker@Rocky ~]$  docker network create -d bridge --ipv6 --subnet 10.0.0.0/8 --subnet=2a01:e0a:172:3910::/64 --gateway 10.10.10.1 pierreauIPV6
```

Les informations (Containers, Subnet, Gateway, etc…) du réseau « bridge » sont consultables via :

```
[docker@Rocky ~]$  docker network inspect bridge
```

L'ouverture de différents ports sur un conteneur dans un réseau spécifique s'effectue de manière semblable (pour un serveur de messagerie) :

```
[docker@Rocky ~]$  docker run --network SwarmNetwork -itd --name postfix -p 110:110 -p 143:143 -p 465:465 -p 587:587 -p 993:993 -p 995:995 postfix:latest
```

D. Mise en oeuvre

L'exercice suivant consiste à installer un cluster de serveurs nginx

1. Ouverture des ports

```
[docker@Rocky ~]$  firewall-cmd --permanent --add-port=2377/tcp
[docker@Rocky ~]$  firewall-cmd --permanent --add-port=7946/tcp
[docker@Rocky ~]$  firewall-cmd --permanent --add-port=7946/udp
[docker@Rocky ~]$  firewall-cmd --permanent --add-port=4789/udp
[docker@Rocky ~]$  firewall-cmd --permanent --add-port=80/tcp
[docker@Rocky ~]$  firewall-cmd --reload
[docker@Rocky ~]$  systemctl restart docker
```

2. Initialisation d'un noeud

Swarm nécessite au moins un « manager » pour gérer les nœuds du cluster. L'exemple suivant met en place un manager accessible sur l'adresse IP *MANAGER-IP* (port par défaut : 2377). Cette adresse IP correspond à une adresse du réseau local *192.168.x.y*.

```
[docker@Rocky ~]$  docker swarm init --advertise-addr <MANAGER-IP>
Swarm initialized: current node (tl9ps5zgym09ior21isb1zcmy) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-Token \
    MANAGER-IP:2377
```

Alternative : l'initialisation peut s'effectuer sur le manager :

```
[docker@Rocky ~]$   docker-machine ssh mymanager "docker swarm init --
advertise-addr <MANAGER-IP>"
```

La commande swarm init génère deux tokens : un pour l'ajout de manager, l'autre pour l'ajout d'un worker :

```
[docker@Rocky ~]$  docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-bla-bla MANAGER-IP:2377

[docker@Rocky ~]$  docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-bla-bla2 MANAGER-IP:2377
```

Alternative : la jonction peut s'effectuer sur les workers :

```
[docker@Rocky ~]$  docker-machine ssh worker "docker swarm join \
--token SWMTKN-1-bla-bla2 MANAGER-IP"
```

Vérification du noeud :

```
[docker@Rocky ~]$  docker node ls
ID                            HOSTNAME              STATUS  AVAILABILITY  MANAGER STATUS
tl9ps5zgym09ior21isb1zcmy *   centos   Ready   Active        Leader

[docker@Rocky ~]$  docker node inspect centos --pretty
```

Nous pouvons modifier l'état d'un nœud de cette manière :

```
[docker@Rocky ~]$  docker node update --availability pause localhost.localdomain
[docker@Rocky ~]$  docker node update --availability active localhost.localdomain
```

3. Création d'un service

Un service est la formalisation de tâches qui seront exécutées par les noeuds worker.

Le service web sera à l'écoute sur le port 80 :

```
[docker@Rocky ~]$  docker service create -p 80:80 --name web nginx:latest
```

Un navigateur web permet de vérifier que le serveur est fonctionnel : http://MANAGER-IP/

Les informations du service sont visibles avec :

```
[docker@Rocky ~]$  docker service inspect web
```

4. Scalabilité

La scabilité permet la continuité d'un service dans de bonnes conditions lors d'une montée en charge. Swarm permet de dupliquer des conteneurs afin d'effectuer un paraléllisme, et d'assurer ainsi une fluidité lors d'une forte sollicitation.

La commande suivante va créer 3 instances du conteneur ngnix :

```
[docker@Rocky ~]$  docker service scale web=3
web scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
```

Le contrôle de l'état du service s'effectue via :

```
[docker@Rocky ~]$  docker service ps web
ID             NAME    IMAGE          NODE                    DESIRED STATE     CURRENT STATE
elg4dqob1ux1   web.1   nginx:latest   localhost.localdomain   Running           Running 3 minutes ago
ouvvOmw1jdi8   web.2   nginx:latest   localhost.localdomain   Running           Running about a minute ago
v1if8y8fkhw2   web.3   nginx:latest   localhost.localdomain   Running           Running about a minute ago
```

5. Les secrets

Les secrets Docker offrent un moyen sécurisé de stocker des informations sensibles (login / mot de passe, certificats auto-signés, ou clés SSH). Effectivement, ces informations ne devraient pas se trouver dans nos Dockerfiles, ni dans des fichiers docker-compose.yml, et encore moins sur des dépôts Git !

Mauvais exemple pour docker-compose :

```
---
version: '3.9'

...
  postgres:
    image: postgres
    restart: always
    ports:
    - 5432:5432
    environment:
      POSTGRES_USER: PostgresUser
      POSTGRES_PASSWORD: PostgresPWD
      POSTGRES_DB: PostgresDBname
...
```

Nous allons créer des fichiers locaux :

```
[docker@Rocky ~]$  vi .db_PostgresUser.txt
```

```
PostgresUser
```

```
[docker@Rocky ~]$  vi .db_PostgresPassword.txt
```

```
PostgresPWD
```

```
[docker@Rocky ~]$  vi .db_PostgresDB.txt
```

```
PostgresDBname
```

…puis modifier docker-compose.yml

```
[docker@Rocky ~]$  vi docker-compose.yml
```

```
---
version: '3.8'

...
  postgres:
    image: postgres
    restart: always
    ports:
    - 5432:5432
    environment:
      POSTGRES_USER: '/run/secrets/db_ PostgresUser'
      POSTGRES_PASSWORD: '/run/secrets/db_PostgresPassword'
      POSTGRES_DB: '/run/secrets/db_PostgresDB'
     secrets:
       - db_PostgresUser
       - db_PostgresPassword
       - db_PostgresDB

secrets:
   db_PostgresUser:
     file: ./.db_PostgresUser.txt
   db_PostgresPassword:
     file: ./.db_PostgresPassword.txt
   db_PostgresDB:
     file: ./.db_PostgresDB.txt

...
```

XVII. SUPERVISION

A. Commandes

Plusieurs commandes Docker nous permettent une surveillance des isolateurs.

Affichez l'information sur l'ensemble du système

```
[docker@Rocky ~]$  docker info
```

Conteneurs

```
[docker@Rocky ~]$  docker inspect <container>
[docker@Rocky ~]$  docker container logs <container>
```

Ports mappés

```
[docker@Rocky ~]$  docker port <container>
```

Evènements : les commandes suivantes s'exécutent sur un terminal libre

```
[docker@Rocky ~]$  docker system events
[docker@Rocky ~]$  docker system events --filter 'container=test' --filter 'event=stop'
[docker@Rocky ~]$  docker events --filter 'event=restart' --since=60m
```

Etat des process

```
[docker@Rocky ~]$  docker top <container>
```

Journaux système

```
[docker@Rocky ~]$  docker logs --tail 0 -f <container>
```

Statistiques du conteneur

```
[docker@Rocky ~]$  docker stats ContDeb1
CONTAINER     CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
ContDeb1      0.00%     476 KiB / 3.844 GiB   0.01%     648 B / 648 B     0 B / 0 B     1
```

Spécification des colonnes :

- .Container : nom ou ID du conteneur
- .Name : nom du conteneur
- .ID : identifiant du conteneur
- .CPUPerc : pourcentage de CPU
- .MemUsage : utilisation de la mémoire
- .NetIO : flux réseau
- .BlockIO : utilisation du disque dur en Lecture/Écriture
- .MemPerc : pourcentage de mémoire
- .PIDs : nombre de PID

Exemple :

```
[docker@Rocky ~]$ docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemPerc}}\t{{.PIDs}}"
```

Stockage utilisé :

```
[docker@Rocky ~]$  docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              17                  16                  2.141GB             126.3MB (5%)
Containers          58                  17                  327.1MB             5.19kB (0%)
Local Volumes       9                   7                   400.7MB             0B (0%)
Build Cache         0                   0                   0B                  0B
```

Informations du système :

```
[docker@Rocky ~]$  docker system info
```

Tailles des images, conteneurs et volumes :

```
[docker@Rocky ~]$  docker system df -v
```

Un package peut être installé en supplément : ctop

```
[docker@Rocky ~]$  dnf install ctop
```

Nettoyage des images, volumes… :

```
[docker@Rocky ~]$  docker system prune
```

A partir de la version 17.06.1 de Docker, les volumes ne sont plus éffacés par défaut, et il faut spécifier ces options :

```
[docker@Rocky ~]$  docker system prune -a --volumes
```

B. Sécurité

Docker propose un outil pour auditer la sécurité des conteneurs :

```
[docker@Rocky ~]$ git clone https://github.com/docker/docker-bench-security.git
```

C. Outils GUI

En plus de Portainer, d'autres environnements graphiques peuvent être installés afin de contrôler l'environnement Docker :

1. Shipyard

https://shipyard-project.com/

Interface simpliste et minimaliste.

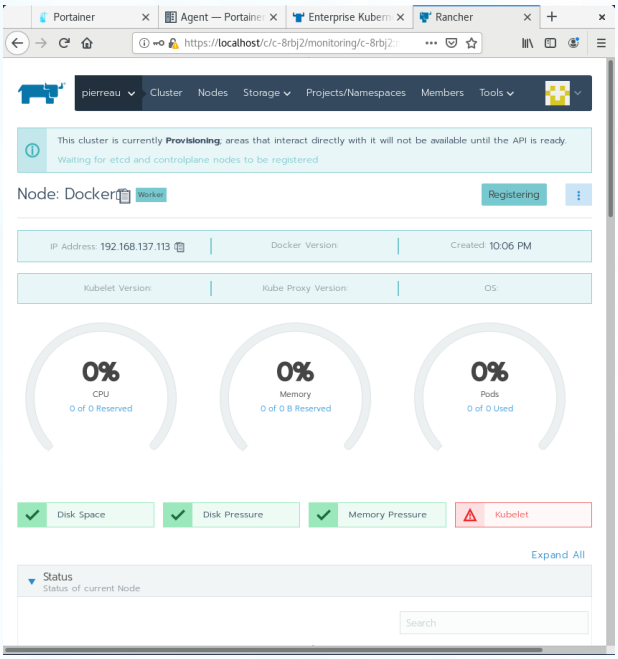
http://localhost:8080/

Login par défaut : admin / shipyard

2. Rancher

Interface élaborée permettent de travailler avec des instances dans les clouds (Amazon, Azure…) via Kubernetes.

http://rancher.com/



XVIII. Autres outils de conteneurisation

A. En production

1. Proxmox

Proxmox VE (Virtual Environment) est un hyperviseur open source (licence AGPL) conçu sur QEMU/KVM et LXC.

Des images sont disponibles sur ce hub : https://www.turnkeylinux.org/

2. Apache Mesos

Fortement orienté Big Data, Mesos est un environnement Open Source de gestion de cluster, qui fonctionne en parallèle d'applications distribuées telles que Hadoop, Spark, Kafka, Elasticsearch…

1. k3s

Développé par Rancher. Support possible.

1. Microk8s

Développé par Canonical. Support possible. Prêt pour la haute disponibilité.

B. Hors production

1. Docker Desktop

Installation : https://www.docker.com/products/docker-desktop/

2. Minikube

Minikube est un outil qui facilite l'exécution locale de Kubernetes. Minikube exécute un cluster Kubernetes à nœud unique dans une machine virtuelle.

```
[docker@Rocky ~]$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube
[docker@Rocky ~]$  mkdir -p /usr/local/bin/
[docker@Rocky ~]$  install minikube /usr/local/bin/
```

Quitter le compte root

Installation de l'ISO de minikube sur Virtualbox :

```
[k8s@ Rocky ~]$ minikube start --vm-driver=virtualbox
  Downloading VM boot image ...
  Creating virtualbox VM (CPUs=2, Memory=2000MB, Disk=20000MB) ...
  Preparing Kubernetes v1.17.0 on Docker '19.03.5' ...
  Downloading kubeadm v1.17.0
  Downloading kubelet v1.17.0
  Pulling images ...
  Launching Kubernetes ...
⌛  Waiting for cluster to come online ...
  Done! kubectl is now configured to use "minikube"

[k8s@ Rocky ~]$ minikube status
[k8s@ Rocky ~]$ minikube dashboard --url
```

192.168.137.111:6443

Arêt de minicube :

```
[k8s@ Rocky ~]$ minikube stop
```

XIX. Ressources

Documentation

https://docs.docker.com

Images gratuites

https://hub.docker.com/

Kubernetes

https://kubernetes.io/

https://www.linuxfoundation.org/

DevSecOps

Recommandations de sécurité relatives au déploiement de conteneurs docker (ANSSI) :

https://www.ssi.gouv.fr/guide/recommandations-de-securite-relatives-au-deploiement-de-conteneurs-docker/

- S'abonner aux alertes du CERT (Computer emergency response team)
- Evaluer les risques selon leurs indices CVSS
- Adopter un SMSI (système de management de la sécurité de l'information), avec l'implémentation de la norme ISO 27001

- Consulter et corriger les « Common Vulnerabilities and Exposures » :

- Adopter les bonnes pratiques du site Cyber malveillance :

Exemple : évaluer le niveau de cybersécurité de votre site Internet