

Créer et administrer ses conteneurs virtuels d'applications

https://www.orsys.fr/formation-docker-creer-administrer-vos-conteneurs-virtuels-applications.html

IMPORTANT : avez-vous validé ces prérequis pour la formation ?
https://www.orsys.fr/qcm/?stagecd=DOK&languecd=fr

ATTENTION : je n'ai aucune obligation de moyens,
cependant mes résultats dépendent de vos prérequis

ORSYS formation
Learning for success*

Cette formation explique la mise en œuvre de conteneurs avec Docker.

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

Nous continuerons avec une approche sécurité et supervision,
en incluant les bonnes pratiques de mise en production CI / CD.

Bonne lecture…



Pierre ROYER Manager | Architecte | Formateur #numérique

Copyright © 2026, Pierre ROYER - https://pierreau.fr/pro/

I. INDEX

I. INDEX 2
II. PREAMBULE 6
A. CE DOCUMENT 6
B. CONVENTIONS 6
C. OBJECTIFS PEDAGOGIQUES 7
III. INTRODUCTION 8
A. CONTEXTE AGILE 8
B. DEVOPS 8
C. LA SECURITE AVEC DEVSECOPS 10
D. CATTLE NOT PETS 12
IV. LA VIRTUALISATION 13
A. HYPERVISEUR DE TYPE 1 (PARAVIRTUALISATION) 13
B. HYPERVISEUR DE TYPE 2 (EMULATEURS) 13
C. LES CONTENEURS 14
D. LINUX CONTAINERS 14
E. COMPARAISON DES CONTENEURS LINUX 15
F. PRESENTATION DE DOCKER 15
G. EXEMPLE D'IMPLEMENTATION 15
V. INSTALLATION 17
A. PROCEDURE 17
1. Installation manuelle 17
2. Installation via un script 17
B. POST-INSTALLATION 17
VI. PREMIERS PAS 20
A. APERÇU DES COMMANDES 20
B. INSTALLATION D'UN CONTENEUR 21
1. Récupération d'une image 21
2. Création d'un conteneur 21
3. Démarrage de conteneurs 22
4. Arrêt / suppression de conteneurs / images 22
5. Copie de fichiers 23
6. Paramètres de démarrage 24
C. MODIFICATION DES CARACTERISTIQUES 24
D. EXERCICE : SERVEUR WEB PERSONNALISE 24
VII. DOCKERFILE 26
A. LES PRINCIPALES DIRECTIVES 26
B. MULTI STAGED BUILDS 29
C. BUILDKIT 29
1. Les instances 30
2. Création d'une image ARM64 30
D. GIT 30
E. SUPERVISORD 31
VIII. IMAGES & LAYERS & CONTAINERS 32
A. MODELE EN COUCHES 32
B. PILOTES ET SYSTEMES DE FICHIERS 32
C. CONVERSION D'UN CONTENEUR EN IMAGE 33
D. EXPLORATION ET OPTIMISATION D'UNE IMAGE 33
E. OPTIONS DE CONVERSION 34
F. ARCHIVAGE D'UNE IMAGE 34
G. GESTION DES IMAGES 34

H. UTILISATION DE DEPOT 34
I. HARDENED IMAGES 35
J. ROOTFS 35
1. Alpine Linux 35
2. Rocky linux 36
IX. VOLUMES 37
A. CREATION D'UN VOLUME 37
B. CREATION A LA VOLEE 38
C. DROITS SUR LES CONTENEURS 38
D. GESTION DES VOLUMES 38
E. BIND ET TMPFS 38
1. Montage d'un répertoire sur l'hôte 38
2. Montage bind d'un fichier sur l'hôte 38
3. Montage bind avec docker compose 39
4. Montage d'un volume dans la mémoire vive 39
5. Montage d'une unité de stockage 39
X. DOCKER COMPOSE 41
A. APACHE+PHP 41
B. SGBD 44
C. DEMARRAGE SELECTIF DES SERVICES 45
1. docker compose 45
2. profiles 45
3. include 46
D. DOCKER COMPOSE + DOCKERFILE 46
E. ARRET ET DESINSTALLATION 47
XI. QUELQUES OUTILS D'ADMINISTRATION 48
A. PORTAINER 48
B. DOCKHAND 49
C. ARCANE 49
D. DTOP 50
E. DOCKLY 50
F. NETDATA 51
XII. HEALTHCHECK 53
A. LIGNE DE COMMANDE 53
B. DOCKERFILE 53
C. DOCKER COMPOSE 54
1. Service Apache 54
2. Service PostgreSQL 54
3. Service MariaDB 54
4. Service MySQL 54
5. Service MongoDB 55
6. Service Redis 55
7. Service SSH 55
8. Annuaire OpenLDAP 55
D. DEPENDANCES 55
XIII. LES VARIABLES ARG, ENV ET LES SECRETS 57
A. ARG 57
1. Dockerfile 57
2. docker compose 58
B. ENV 58
1. docker run 58
2. Dockerfile & Entrypoint 59
3. Les fichiers d'environnements 59
C. LES SECRETS 61
1. Le principe 61
2. Exemple concret 62
XIV. LES RESEAUX 63

A. CREER UN RESEAU 63
1. Réseau host 63
2. Réseau Classique (bridge) 63
3. Réseau Internal 64
4. Utiliser des réseaux déjà existants 64
5. Création à la volée 66
6. Réseau overlay 66
7. Réseau none 66
B. IP FIXES 66
1. Dans un réseau existant 66
2. Dans un nouveau réseau 67
C. DURCISSEMENT 67
D. RUSTNET 68
XV. LES FORMATS 69
XVI. SAUVEGARDES & RESTAURATIONS DES VOLUMES 70
A. COPIE DE VOLUME 70
B. COPIE D'UN REPERTOIRE MONTE 70
C. SAUVEGARDES SQL 71
1. PostgreSQL 71
2. MariaDB 71
3. MySQL 71
XVII. SECURITE 73
A. LINUX CAPABILITIES 73
1. Les privilèges 74
2. Le mode privilégié 74
B. NAMESPACE 75
1. Principes 75
2. Création d'un conteneur natif 76
C. CGROUPS 79
1. Ressources en mémoire vive 80
2. Ressources en mémoire partagée 81
3. Ressources processeur 81
4. Ressources PIDs 82
5. Ressources stockage 82
6. Docker compose 83
7. Ressources réseau 83
8. Métriques 83
D. AUDITS 83
1. Trivy 84
2. bench-security 84
E. FRONT-END 84
1. Solutions 84
2. Entêtes HTTP 84
3. Fail2ban 85
4. Crowdsec 86
XVIII. SUPERVISION 87
A. COMMANDES 87
B. OUTILS GUI 88
1. Rancher 88
2. OpenNebula 89
C. ZABBIX 90
XIX. RESSOURCES 91
XX. EXERCICES 92
A. SECURITY BY DEFAULT 92
1. SSH 92
2. SSL 92
B. PRIVACY BY DESIGN 93

C. ACCOUNTABILITY 93
XXI. ANNEXE 94
A. OUTILS DE CONTENEURISATION 94
1. En production 94
2. Hors production 95

II. PREAMBULE

A. Ce document

Informations

Nom du document	DevOps.Docker.docx	Référence	DEVOPS-DOCKER
Version	2026.09.20	Pages	95
Date de création	07/03/2017	Dernière modification :	27/09/2026
Auteur :	Pierre ROYER Tél : (+33) 614 672 909 https://www.linkedin.com/in/pierreau	Contributeur(s) :	
Mode de diffusion	☐ Confidentiel ☐ Restreint ☒ Interne ☐ Libre	Diffusion :	Clients et stagiaires
Annexes :	https://pierreau.fr/pro/		

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
[docker@Rocky ~]$ désigne un compte utilisateur local

Les commandes sont en gras, le contenu d'un fichier est encadré, les caractères en italique sont des exemples de paramètres à adapter selon notre environnement :

```
[root@Server ~]# vi /etc/hosts
```

```
127.0.0.1                 localhost
::1                       localhost ip6-localhost ip6-loopback
192.168.1.5               rpi5b rpi5b.cloud.pierreau.fr
2a01:cafe:face:feed::5    rpi5b rpi5b.cloud.pierreau.fr
```

Information utile	Attention particulière	Risque important
		

Copyright © 2026, Pierre ROYER - https://pierreau.fr

C. Objectifs pédagogiques

Présentation de la formation

Docker est un projet open source qui 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.

Cette formation vous permettra de comprendre les concepts d'isolation et de conteneurisation dans un environnement Linux, de créer vos propres images, conteneurs, réseaux virtuels, via Docker Engine, Dockerfile, Docker compose...

Elle s'effectuera sur la dernière version de Docker, sur Rocky Linux ou Ubuntu server.

Objectifs

- Installer, paramétrer et exploiter Docker dans un contexte d'intégration continue (CI / CD)
- Utiliser et créer ses propres images personnalisées
- Paramétrer, déployer, automatiser, et standardiser les environnements
- Gérer ses conteneurs et la sécurité des environnements en maîtrisant les ressources système
- Créer des réseaux sécurisés et isolés, afin de protéger ses données sensibles
- Gérer ses volumes : statiques, bind, RAM
- Sauvegardez et restaurez vos données
- Publier ses services sur le réseau
- Utiliser les HealthCheck pour auditer l'état de santé des conteneurs
- Gérer les variables ARG et ENV, et les secrets
- Intégrer une approche DevSecOps par défaut
- Utiliser les formats pour filtrer les informations
- Utiliser des interfaces graphiques (Portainer, Dockhand, Arcane, Netdata…)
- Utiliser les "capabilities", ou le mode privilégié
- Acquérir une autonomie sur Docker.

Public concerné

- Administrateur / ingénieurs systèmes / réseaux, DevOps
- Développeurs, Lead dev.
- Responsable de pôle technique, CTO
- Scrum master, Chefs de projets infrastructures
- Ecoles d'Ingénieurs numériques.

Pré-recquis

- Aisance sur les systèmes Linux et l'éditeur VI.

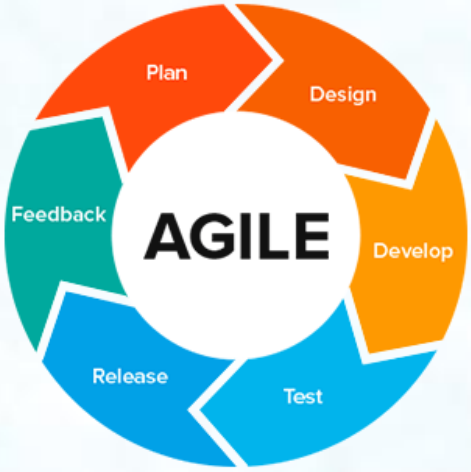
Copyright © 2026, Pierre ROYER - https://pierreau.fr

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 (Time To Market).

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.

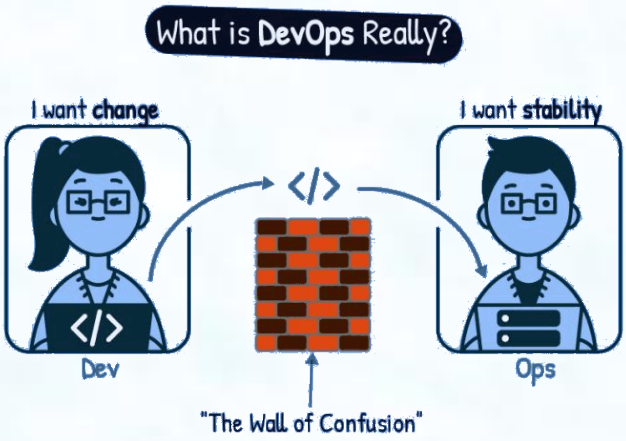
Copyright © 2026, Pierre ROYER - https://pierreau.fr

Le cycle DevOps



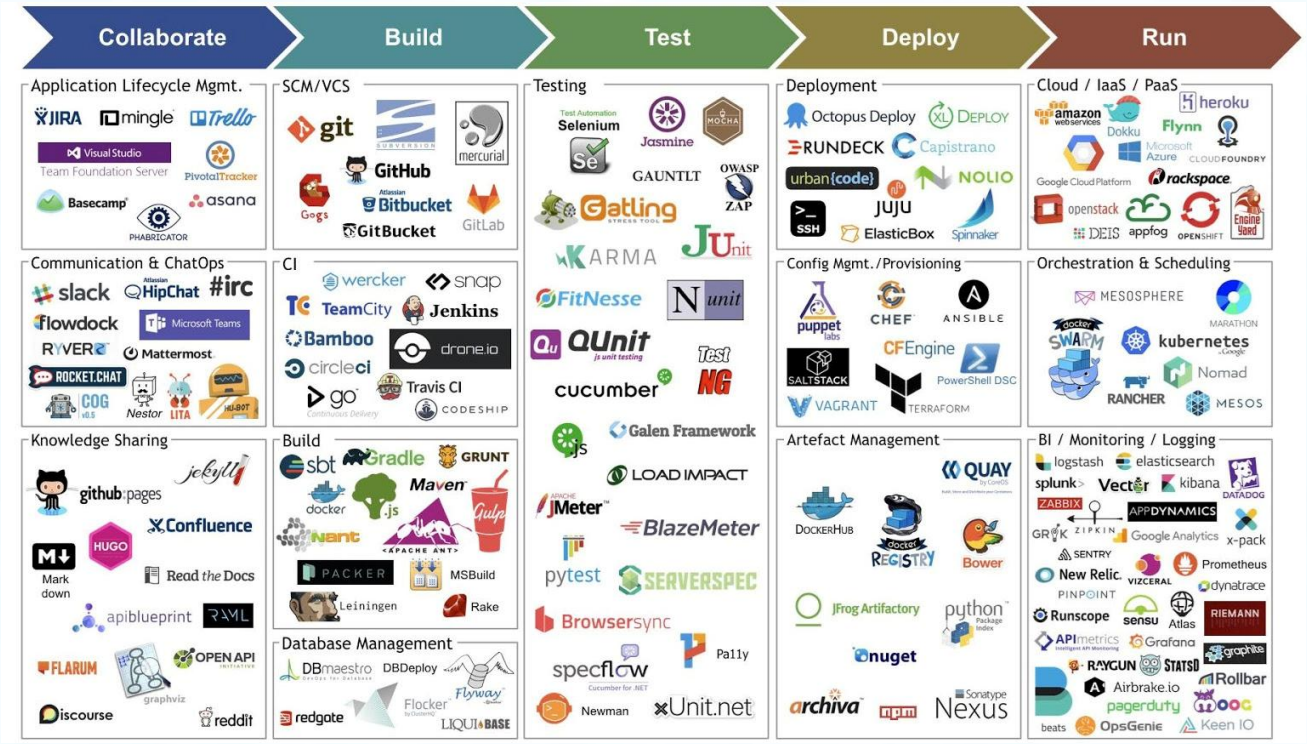
Patrick Debois est le fondateur du DevOps

Le DevOps vise à répondre à des objectifs complémentaires, et contradictoires :
- Les Dev doivent s'adapter rapidement et continuellement
- Les Ops doivent garantir stabilité, sécurité, conformité
- Briser le « mur de la confusion », en unifiant le développement logiciel (dev) et l'administration des infrastructures informatiques (ops).



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) :

Copyright © 2026, Pierre ROYER - https://pierreau.fr



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.

Copyright © 2026, Pierre ROYER - https://pierreau.fr



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… Gardez à l'esprit ces principes :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

- La question n'est pas de savoir si vous allez être attaqué, mais quand…
- La sécurité dépend toujours du maillon le plus faible
- Appliquez la règle du moindre privilège
- La théorie du chaos : de petites variations dans l'état initial peuvent entrainer de grandes variations dans l'état final (effet papillon).

D. Cattle not pets

Il s'agit d'un des principes de DevOps.

Les "pets" sont constitués de serveurs traités comme des animaux de compagnie, c'est-à-dire qu'ils sont choyés et soignés individuellement, avec une grande attention portée à leur santé et à leur bien-être

En revanche, dans un environnement "cattle", les serveurs sont traités comme du bétail, c'est-à-dire qu'ils sont considérés comme des ressources jetables qui peuvent être facilement remplacées en cas de problème. Les serveurs sont créés et détruits rapidement et facilement, et les mises à jour sont effectuées en utilisant des scripts automatisés qui permettent de mettre rapidement en place des configurations standardisées. Docker est donc clairement orienté "cattle".

Outre les classiques offres d'hébergement IaaS, PaaS, SaaS, le CaaS (Container as a Service) est devenu monnaie courante dans les Clouds privés, hybrides, et publics.

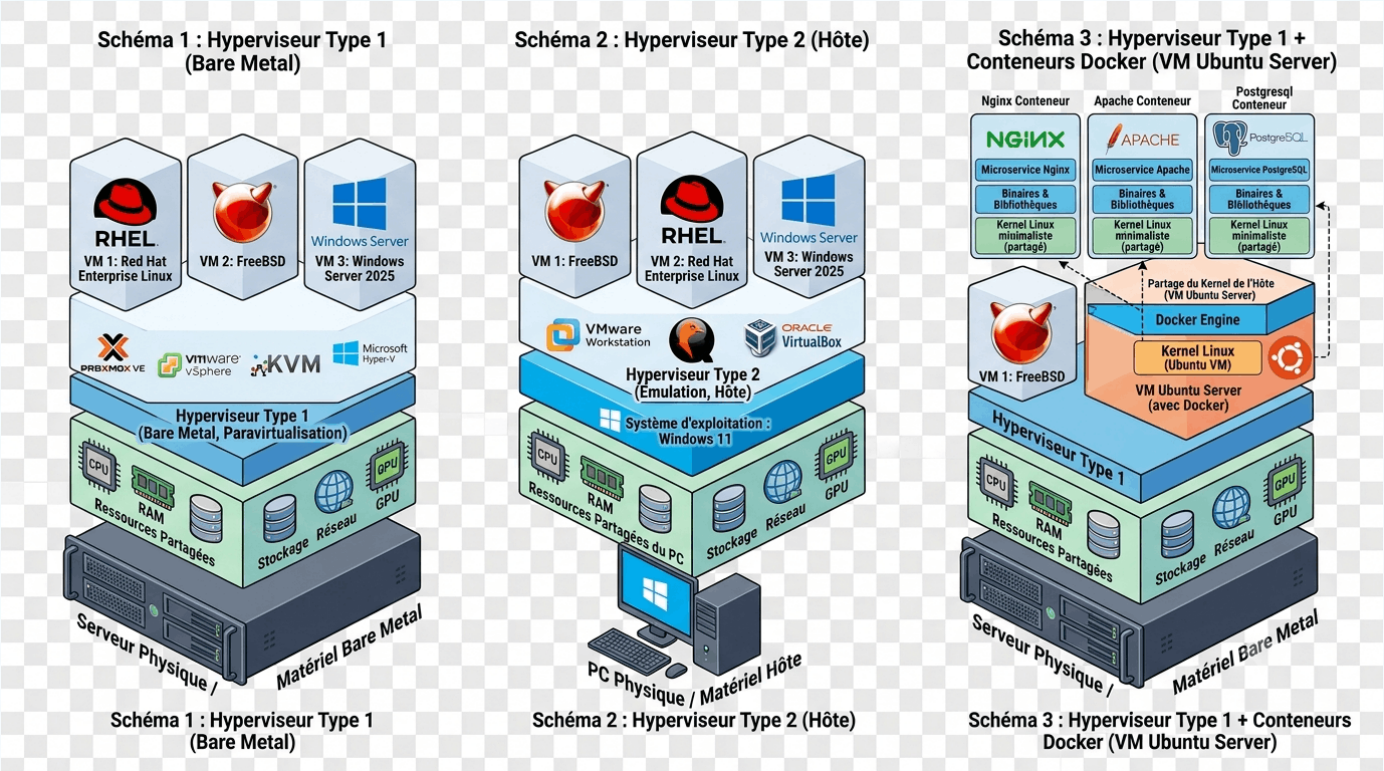
Copyright © 2026, Pierre ROYER - https://pierreau.fr

IV. LA VIRTUALISATION

La virtualisation de serveurs est un concept qui existe depuis 1972 : IBM le pratiquait 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, AMD-V, ou ARM VHE (Virtualization Host Extensions).



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.

Les solutions présentes sur le marché sont Proxmox, VMware vSphere (ESX), KVM, Citrix hypervisor, Microsoft Hyper-V Server, Oracle VM, Parallels Server

B. 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,

Copyright © 2026, Pierre ROYER - https://pierreau.fr

mémoire, carte graphique, interfaces réseau...) et non aux ressources matérielles réelles. Les performances en émulation sont dégradées du fait de cette simulation.

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

C. Les conteneurs

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)

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.

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

D. 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 storage ls`
- `lxc network ls`
- `lxc image list images: architecture=$(uname -m) type=container`
- `lxc launch images:rockylinux/9/amd64 ct-rocky`
- `lxc ls status=running`
- `lxc file push ./index.html ct-rocky/tmp/`
- `lxc-info`
- `lxc list`

Copyright © 2026, Pierre ROYER - https://pierreau.fr

- `lxc-stop -n ct-rocky`
- …

E. Comparaison des conteneurs Linux

	Docker	Podman	Incus	LXC
Éditeur	Docker, Inc. Le moteur open source sous-jacent, Moby, est gouverné par la communauté.	Red Hat + communauté open source active.	Communauté Linux Containers	Communauté Linux Containers
Cas d'usage	CI/CD, microservices	Mode rootless (sécurité), intégration systemd	Conteneurs système	Conteneurs système
Nombre d'images	~13 millions, dont presque 900 images "officielles" et "hardened".	Utilise les mêmes images OCI que Docker.	Quelques dizaines de distributions Linux disponibles	Même catalogue que Incus
Dépôts	hub.docker.com	hub.docker.com quay.io	images.linuxcontainers.org	images.linuxcontainers.org
Orchestrateurs	Docker Swarm Kubernetes HashiCorp Nomad.	Kubernetes Quadlets	Incus cluster Terraform / OpenTofu	LXD Proxmox VE

F. Présentation de Docker

Docker est un logiciel développé en langage Go, sous licence Apache 2.0, qui a été distribué en tant que projet open source à partir de mars 2013. Originellement, Docker utilisait LXC ; il s'appuie actuellement sur libcontainer.

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.

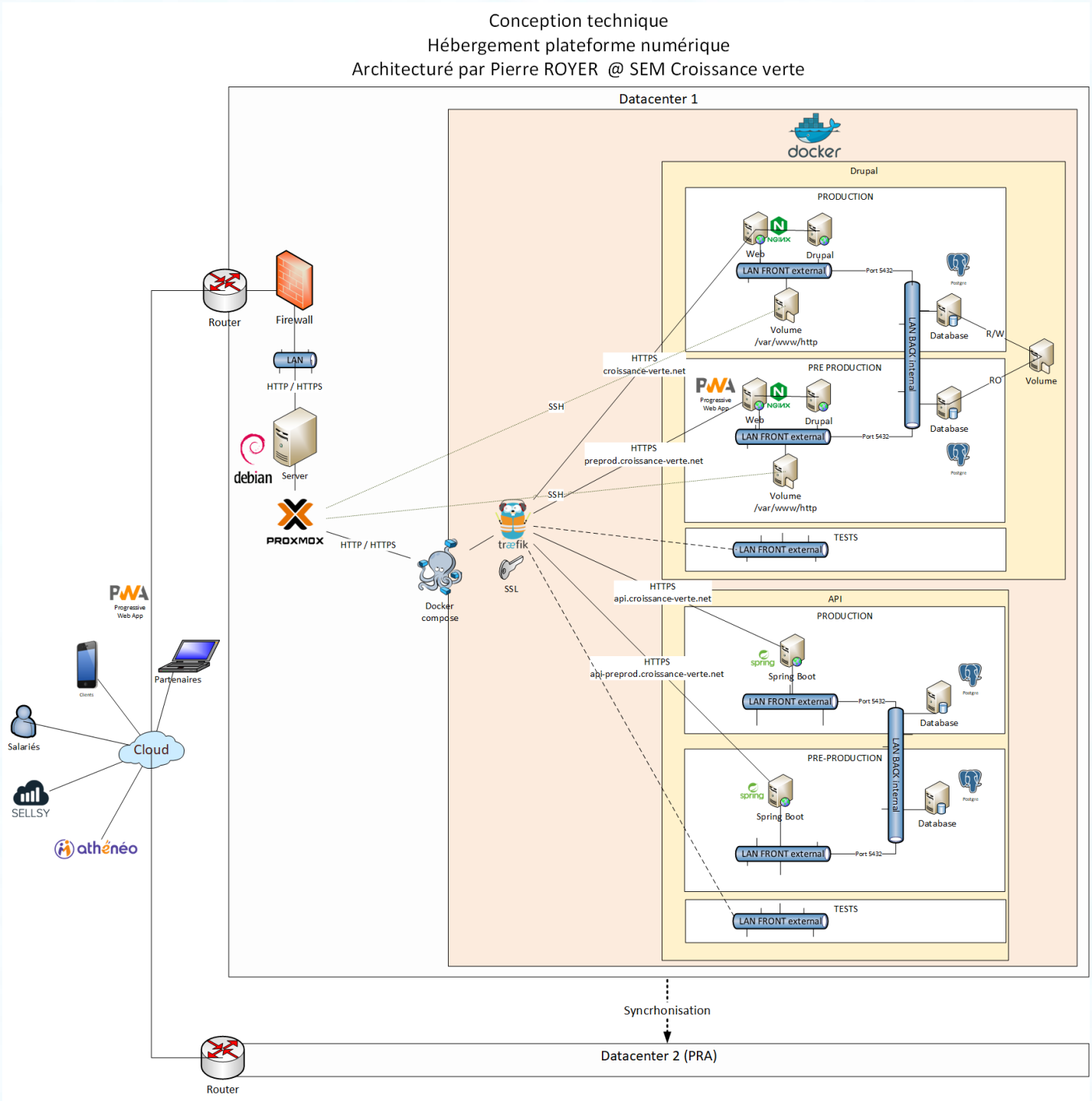
Docker utilise des fonctionnalités natives de Linux comme :
- son noyau
- ses namespaces : isolation des processus dans une ressource partagée
- ses Cgroups (control groups) : fonctionnalités pour limiter l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.)

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.

Copyright © 2026, Pierre ROYER - https://pierreau.fr



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 officiel 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…

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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 10 Minimal.

A. Procédure

Pour des raisons évidentes de sécurité sur un serveur Linux, il faut protéger (et donc isoler) la partition root du système (/) des partitions utilisées par docker.

Lors de l'installation su système, il faut :

- Prévoir un espace de stockage suffisant (au moins 10 Go) sur le filesystem `/var/lib/docker`
- Créer un autre filesystem isolé pour le compte docker (`/home`), qui va héberger les volumes de données persistantes de type « bind ».

Paramétrer une IP fixe et une passerelle sur votre serveur
Faire correspondre le nom du serveur aux adresses des interfaces réseau : `/etc/hosts`
Déclarer les DNS : `/etc/resolv.conf`
Nommer le serveur :

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

1. Installation manuelle

Suivre la procédure sur le site officiel : https://docs.docker.com/engine/install/

Le groupe `docker` à été créé dans /etc/group

Créer un compte de service docker avec le même groupe :

```
[root@Rocky ~]# useradd -m -s /bin/bash -g docker docker
```

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

2. Installation via un script

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

B. Post-installation

Configuration du driver pour le stockage (optionnel)

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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

```
{
    "storage-driver": "vfs"
}
#{
#    "storage-driver": "devicemapper"
#}
```

Configuration des DNS :

```
[root@Rocky ~]$  vi /etc/default/docker
```

```
DOCKER_OPTS="--dns 1.1.1.1 --dns 1.0.0.1 --dns 2606:4700:4700::1111
--dns 2606:4700:4700::1001"
```

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

Utiliser le compte créé pour gérer docker :

```
[root@Rocky ~]# su - docker
```

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
```



Alias

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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

```
alias dsta='docker start'
alias dsto='docker stop'
alias dps='docker ps -a --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"'
alias dstoa='docker stop $(docker ps -a -q)'
alias dex='docker exec -ti'
alias dat='docker attach'
alias dstat='docker stats'
alias dlog='docker logs'
alias drm='docker rm'
alias dim='docker images -a'
alias drmi='docker rmi'
alias dbp='docker builder prune'
alias dcp='docker compose'
alias dvol='docker volume ls --format "table {{.Name}}\t{{.Mountpoint}}"'
alias dsdf='docker system df -v'
etc...
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

VI. Premiers pas

A. Aperçu des commandes

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

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Swarm Commands:
  swarm       Manage Swarm

Commands:
  attach       Attach local standard input, output, and error streams to a running
container
  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
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  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
  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
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  save       Save one or more images to a tar archive (streamed to STDOUT by default)
  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
  wait        Block until one or more containers stop, then print their exit codes

  wait        Block until one or more containers stop, then print their exit codes
```

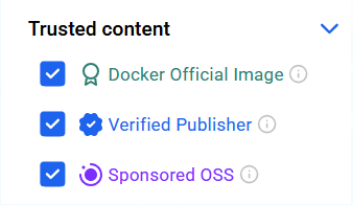
Copyright © 2026, Pierre ROYER - https://pierreau.fr

B. Installation d'un conteneur

1. Récupération d'une image

Docker hub

Je recommande vivement l'utilisation de ce filtre :



Effectivement, des centaines d'images sont identifiées comme étant malveillantes, du cryptomineur à la porte dérobée…

Les images officielles Docker sont organisées et documentées de manière claire, tout en promouvant les meilleures pratiques. Elles sont régulièrement mises à jour, et constituent un point de départ fiable pour de nombreuses applications.

Éditeur vérifié : ces images sont maintenues par les organisations partenaires de Docker, qui vérifient l'authenticité du contenu de leurs dépôts.

Open Source Sponsorisé : publiées et maintenues par des projets open source sponsorisés par Docker.

Terminal

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

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

Quelques autres images :

```
[docker@Rocky ~]$  docker pull debian:trixie-slim
[docker@Rocky ~]$  docker pull bitnami/minideb
[docker@Rocky ~]$  docker pull ubuntu:latest
[docker@Rocky ~]$  docker pull almalinux:latest
[docker@Rocky ~]$  docker pull almalinux:minimal
[docker@Rocky ~]$  docker pull rockylinux/rockylinux:10
[docker@Rocky ~]$  docker pull alpine
```

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)

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[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 un conteneur une fois utilisé et stoppé :

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

Exécution d'une commande à la création d'un conteneur Fedora :

```
[docker@Rocky ~]$  docker run -ti --name ContFed1 fedora cat /etc/os-release
Debian GNU/Linux 11 \n \l
```

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
```



Lors de la présentation de Docker, nous avons évoqué le fait qu'un conteneur partage le kernel de son hôte ; c'est le moment de le prouver…

```
[docker@Rocky ~]$  uname -r
[docker@Rocky ~]$  docker exec -ti ContDeb0 uname -r
[docker@Rocky ~]$  cat /proc/version
[docker@Rocky ~]$  docker exec -ti ContDeb0 cat /proc/version
```

4. Arrêt / suppression de conteneurs / images

Accéder au shell d'un conteneur déjà démarré

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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 -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 arrêtés

```
[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 ~]$   APP=my-app
[docker@Rocky ~]$   for i in {1..3}; do docker create --name "$APP-$i" busybox ; done
[docker@Rocky ~]$   docker ps -a | grep "$APP"
[docker@Rocky ~]$   docker ps -a | grep "$APP" | 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 fichier.txt conteneur:/tmp/
```

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

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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 mémoire vive, 3 tentatives de redémarrage en cas de crash, et mappage du port externe 80 vers le port interne 80.

```
[docker@Rocky ~]$  docker run -dti --name ContNgx1 --hostname nginx --memory 256m --restart=on-failure:3 -p 80:80 nginx
```

nginx répond bien à partir du serveur hôte (http://*localhost*:80/)



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 à venir.

C. Modification des caractéristiques

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

```
[docker@Rocky ~]$  docker update --restart=always ContNgx1
[docker@Rocky ~]$  docker update --restart=unless-stopped ContNgx1
[docker@Rocky ~]$  docker update --restart=no (par défaut)
```

`unless-stopped` : le conteneur ne redémarrera pas en même temps que le démon Docker
`always` : le démarrage du conteneur est assuré à chaque démarrage du serveur, si le démon Docker est dans le statut `enable`

Vérifications :

```
[docker@Rocky ~]$  docker inspect ContNgx1
```

D. Exercice : serveur web personnalisé

Créer ce 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>
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
    <h1>Le site Internet de notre formation</h1>
    <p>Notre serveur web est fonctionnel !</p>
  </body>
</html>
```

Créer et démarrer un conteneur avec ces caractéristiques :

- Image : ubuntu:latest
- Nom : webserver
- Hostname : webserver
- Ports : 80:80
- Mémoire RAM max : 16M
- Politique de démarrage : always

Dans le conteneur :

- Installer le package apache
- Démarrer le service

Sur l'hôte, vérifier que Apache soit activé : http://:ip_hote:80

Copier le fichier index.html dans le conteneur, au bon endroit :

```
[docker@Rocky ~]$  docker cp index.html webserver:/emplacement/
```

Sur l'hôte, vérifier que la mise à jour soit correctement effectuée : http://:ip_hote:80

Copyright © 2026, Pierre ROYER - https://pierreau.fr

VII. DOCKERFILE

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

A. Les principales directives

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

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

# Reference image
 FROM ubuntu:latest

# Meta data (OCI standard)
  LABEL org.opencontainers.image.title="My image"
  LABEL org.opencontainers.image.description="My description"
  LABEL        org.opencontainers.image.authors="Pierre        ROYER
<mail@pierreau.fr>"
  LABEL org.opencontainers.image.url="https://pierreau.fr/pro/"
  LABEL org.opencontainers.image.version="2"
  LABEL org.opencontainers.image.revision="1"
  LABEL                    org.opencontainers.image.created="2016-11-
28T15:00:00+02:00" # UTC+2 (rfc3339)
# Meta data (not standard)
  LABEL Environment="production"  Site="Nantes"

# Installing packages without interruption (Debian)
#  ARG DEBIAN_FRONTEND=noninteractive

# Commands executed during image creation
# Change mirror source for best performances
  RUN      sed      -i      's/ports.ubuntu.com/fr.ports.ubuntu.com/'
/etc/apt/sources.list.d/ubuntu.sources

  RUN apt-get update && apt-get upgrade -y
  RUN apt install -y --no-install-recommends \
          apache2 \
          cron \
          curl \
          openssh-server \
          tzdata \
          vim \
          wget

# Remove unnecessary files to optimize image size
  RUN apt clean ; rm -rf /var/lib/apt/lists/*

# Setup time zone
  ENV TZ=Europe/Paris
  RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone
  RUN dpkg-reconfigure --frontend noninteractive tzdata

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

# Copying files from host to container
#  RUN  cd /var/www/html/
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
#  RUN  mv index.html index.html.old
#  COPY index.html /var/www/html/

#  RUN  wget https://cloud.pierreau.fr/index.html

# Allows adding URLs or archives
  ADD https://cloud.pierreau.fr/index.html .

# Mount points created in container
  VOLUME   ["/var/www/html", "/etc/apache2", "/var/log/apache2/"]

# Adding a user
  RUN   useradd -m -s /bin/bash -p MOt2P@$$ sshuser
  USER  sshuser
# Specifies default directory
  WORKDIR /home/sshuser
  RUN   echo "df -h" > .profile
  USER  root

# ENTRYPOINT command (not overridable by docker run) launched at
container startup
# COPY  entrypoint.sh /root/
# RUN   chmod 700 /root/entrypoint.sh
# ENTRYPOINT ["/root/entrypoint.sh"]

# Command (overloadable by docker run) launched when the container
starts
  CMD  ["/usr/sbin/apachectl", "-D", "FOREGROUND"]

# For information purposes, specifies the ports/protocols used
  EXPOSE 80 22/tcp 123/udp
```

CMD sert à exécuter des commandes simples lors de chaque démarrage du conteneur.
ENTRYPOINT est utilisé pour lancer des scripts à chaque démarrage du conteneur.



Il ne peut y avoir qu'une seule directive CMD



L'instruction EXPOSE ne publie pas de port/protocole, mais permet d'indiquer ceux destinés à être publiés. Par défaut, tous les ports d'un conteneur sont ouverts dans un même réseau…



Les labels peuvent être très utiles pour la communication inter-conteneurs, ou pour faciliter leur organisation :

```
[docker@Rocky ~]$  docker ps --filter "label=Environment=production"
```

Exemple de entrypoint.sh

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

```
#!/bin/bash
set -euo pipefail # Immediate exit if even one of the commands fails

# Uncomment the desired service :
# /usr/sbin/cron start
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
# /usr/bin/crontab /etc/cron.d/mycron
# service ssh restart && bash (remove && bash)
# /usr/sbin/apache2ctl -D FOREGROUND (remove -D FOREGROUND)

# Only the last service launched MUST be in Foreground

exec "$@"  # Support for arguments passed in script
```

Un autre exemple, plus complexe, pour initialiser un environnement MariaDB : https://github.com/MariaDB/mariadb-docker/blob/master/docker-entrypoint.sh

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 les commandes

```
[docker@Rocky ~]$  docker history pierreau/ubuntu.apache
[docker@Rocky ~]$  docker history --no-trunc pierreau/ubuntu.apache
```

```
[docker@Rocky ~]$  docker run -dti --name ubuntu.apache -p 50080:80
-m 16M pierreau/ubuntu.apache
```



Comme nous avons pu le constater, lorsqu'un conteneur n'a rien à faire, il s'arrête automatiquement après avoir démarré. Une astuce pour laisser un conteneur actif est de lui donner une tâche en boucle :

```
[docker@Rocky ~]$  docker run -dti --name wait alpine sleep infinity
```

Ou :

```
# Stay up
 CMD ["/bin/bash", "-c", "sleep infinity"]
```



La création d'images génère un cache, et de l'espace de stockage. Il est possible de purger ce cache avec la commande :

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



Un outil permet de vérifier la syntaxe d'un Dockerfile : https://hub.docker.com/r/hadolint/hadolint

```
[docker@Rocky ~]$  docker run --rm -i hadolint/hadolint < Dockerfile
```



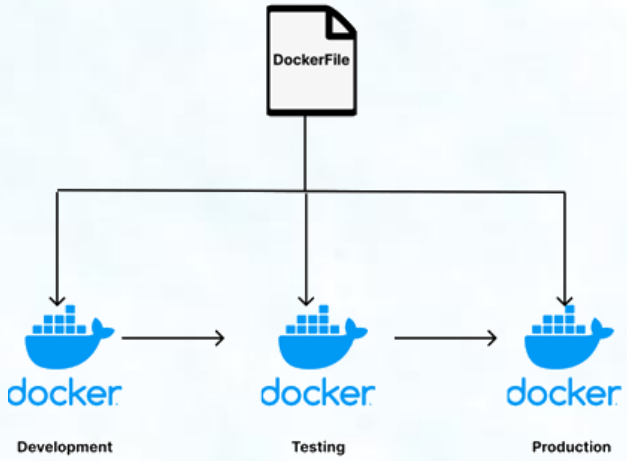
La référence complète des directives Dockerfile :
https://docs.docker.com/reference/dockerfile/

Copyright © 2026, Pierre ROYER - https://pierreau.fr

B. Multi Staged Builds

Depuis la version v17.05, Docker a introduit les builds « multi-étapes ». L'objectif étant d'optimiser la taille des images en fonction des environnements.

Pour illustrer ce contexte, imaginons que nous souhaiterions exécuter un binaire, résultant d'une compilation avec gcc (programme en C) dans un conteneur. Nous avons besoin du compilateur, et du code source pour générer un exécutable. Or, ce dont nous avons besoin en environnement de production, c'est simplement le programme compilé.



L'idée est de compiler le programme dans un conteneur temporaire (environnement de développement), et d'utiliser ce binaire pour les tests et la production :

```
[docker@Rocky ~]$  vi Message.c
```

```
#include <stdio.h>
int main () {
  printf ("\t Bonjour ! \n");
  return 0;
}
```

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

```
FROM alpine:latest AS mybuildstage
RUN apk add --no-cache build-base
COPY Message.c .
RUN gcc -o Message Message.c
FROM alpine:latest
COPY --from=mybuildstage Message .
CMD ["./Message"]
```

```
[docker@Rocky ~]$  docker build -t message:1.0 .
[docker@Rocky ~]$  docker run -ti --name mytapp message:1.0
         Bonjour !
```

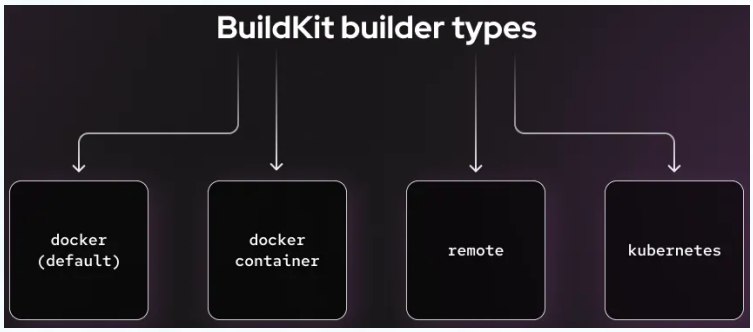
C. Buildkit

Buildkit étend les capacités de docker build, en y ajoutant des fonctionnalités de construction supplémentaires.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

1. Les instances

L'ajout d'instances spécifiques permettent de déployer des images pour des plateformes différentes de cette utilisée (AMD / ARM, noeud distant, Kubernetes).



L'instance par défaut pour la création d'images avec docker build s'appuie sur le pilote par défaut : « docker ».

```
[docker@Rocky ~]$  docker buildx ls
[docker@Rocky ~]$  docker buildx inspect default
```

2. Création d'une image ARM64

Pour construire une image dédiée aux plateformes Raspberry pi, il faut dans un premier temps créer un Dockerfile, et une nouvelle instance basée sur le pilote docker-container.

```
[docker@Rocky ~]$  docker buildx create --use --bootstrap \
  --name armbuilder --driver docker-container \
  --platform linux/arm64,linux/arm/v8

[docker@Rocky ~]$  docker buildx ls
[docker@Rocky ~]$  docker buildx inspect armbuilder

[docker@Rocky ~]$  docker buildx build . --tag arm-image:0.1 --load --platform linux/arm64
```

Arrêt et destruction d'une instance :

```
[docker@Rocky ~]$  docker buildx stop armbuilder && docker buildx rm armbuilder
```

Nettoyage de l'espace de stockage :

```
[docker@Rocky ~]$  docker buildx du
[docker@Rocky ~]$  docker buildx prune
```

Références : https://docs.docker.com/reference/cli/docker/buildx/

D. Git

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

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
.htpasswd
acme.json
pierreau.fr.crt
id_ecdsa.pub
...
```

E. supervisord

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

https://docs.docker.com/engine/containers/multi-service_container/



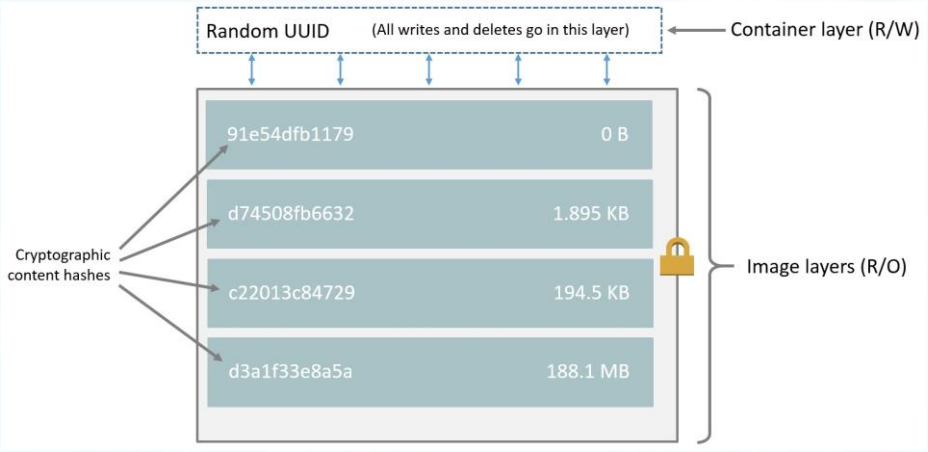
Dans le cadre d'une architecture microservices, un conteneur n'exécute qu'un seul service…

Copyright © 2026, Pierre ROYER - https://pierreau.fr

VIII. 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.

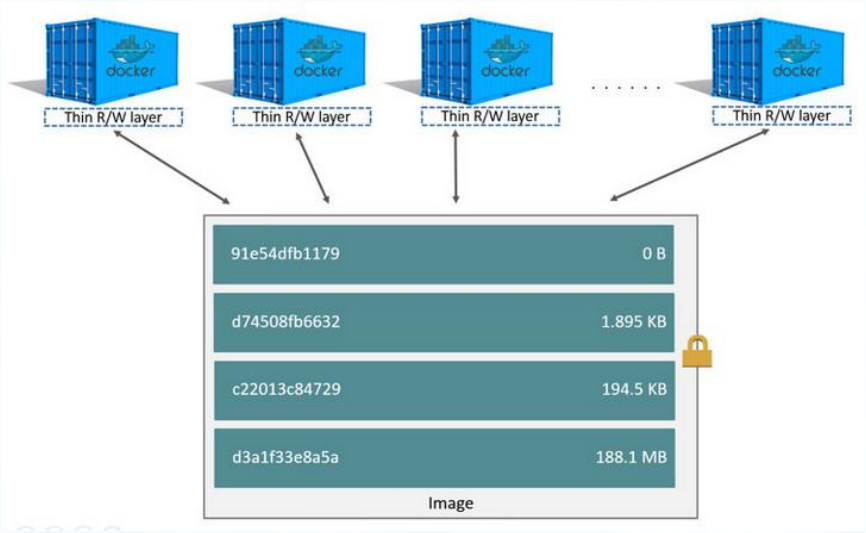


Source : www.docker.com

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.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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

Driver	(sup)porté par
overlay2 / 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. Exploration et optimisation d'une image

L'utilitaire « Dive » permet d'analyser le contenu d'une image et la taille des layers, afin de pouvoir diminuer la taille de cette image.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$ docker run -ti --rm -v /var/run/docker.sock:/var/run/docker.sock wagoodman/dive <Nom_de_l'image>
```

E. Options de conversion

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

```
[docker@Rocky ~]$ docker commit -c "ENV BUILDER=Pierre" debian_nginx.updated debian/nginx:updated
```

…lors du lancement du conteneur construit à partir de cette image, une variable d'environnement « Builder » sera créée.

F. 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
```

G. 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
```

H. 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 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
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

D'autres gestionnaires de dépôts existent, tel que Harbor : https://goharbor.io/
La connexion s'effectue avec l'URL du dépôt :

```
[docker@Rocky ~]$  docker login harbor.cloud.pierreau.fr
[docker@Rocky ~]$  docker tag xfce4:latest harbor.cloud.pierreau.fr/repo/xfce4:1
[docker@Rocky ~]$  docker push harbor.cloud.pierreau.fr/repo/xfce4:1
```

I. Hardened Images

Les images Docker durcies sont optimisées pour réduire les surfaces d'attaque, et limiter les vulnérabilités. Elles sont construites en appliquant ces bonnes pratiques de sécurité :

- Réduction de la taille : moins de paquets inutiles, moins de risques de vulnérabilités
- Utilisateur non-root : exécution des processus avec des privilèges minimaux
- Mises à jour de sécurité : corrections des failles CVE (Common Vulnerabilities and Exposures) connues, et maintenance régulière
- Configuration sécurisée par défaut : désactivation des services ou fonctionnalités non essentiels
- Scan de vulnérabilités : vérification automatique des failles connues avant la publication.

Ces images sont hébergées ici : https://hub.docker.com/hardened-images/catalog, et accessibles via notre compte docker :

```
[docker@Rocky ~]$  docker login dhi.io
[docker@Rocky ~]$  docker pull dhi.io/debian-base:trixie
```

Documentation : https://docs.docker.com/dhi/get-started/

J. Rootfs

Nous pouvons aussi créer nos images vides non pas à partir de Docker hub, mais depuis des sites des différentes distributions :

1. Alpine Linux

```
docker@Debian:~ $  vi rootfs.sh
```

```
#!/bin/bash

# Remove old image
if [[ -d rootfs ]] ; then rm -rf rootfs ; fi
mkdir rootfs
if [[ -e alpine-minirootfs-*.tar.gz ]] ; then rm alpine-minirootfs-*.tar.gz ; fi

# Find last OS version
PROC=$(uname -m)

# Donwnload last rootfs
wget http://dl-cdn.alpinelinux.org/alpine/latest-stable/releases/$PROC/latest-releases.yaml
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
OS_RELEASE=$(grep version ./latest-releases.yaml | uniq | awk '{print
$2}')
wget                    http://dl-cdn.alpinelinux.org/alpine/latest-
stable/releases/$PROC/alpine-minirootfs-$OS_RELEASE-$PROC.tar.gz

tar -xzvf alpine-minirootfs-*.tar.gz -C rootfs
tar -C rootfs -c . | docker import - alpine-arm-custom
rm -rf rootfs
```

2. Rocky linux

```
[docker@Rocky ~]$  mkdir rocky-rootfs
[docker@Rocky ~]$  sudo dnf install --installroot=$(pwd)/rocky-rootfs
--releasever=9  --setopt=install_weak_deps=False  --nodocs  -y  rocky-
release
[docker@Rocky ~]$  tar -C rocky-rootfs -c . | docker import - rocky-
custom
```

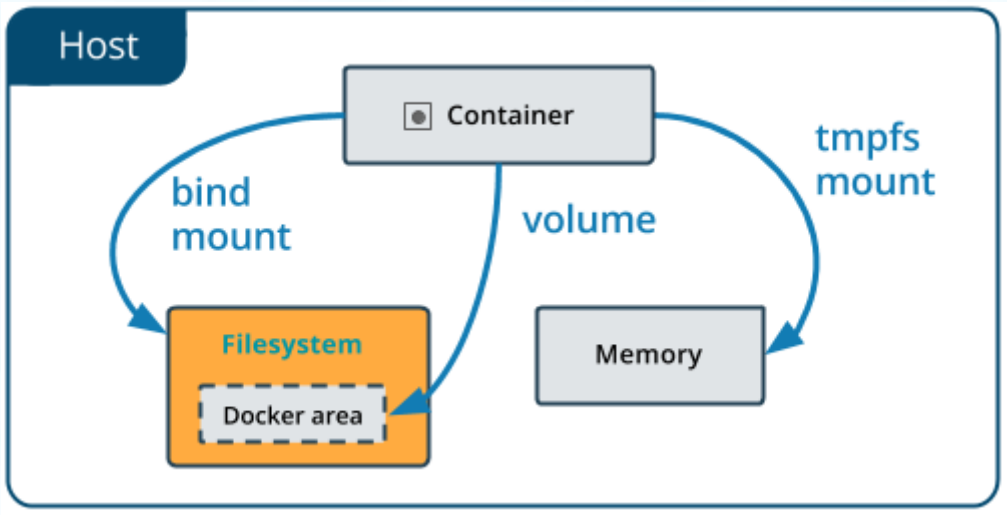
Maintenant que vous avez compris le principe, je vous laisse chercher pour les autres distributions (debootstrap ?)…

Copyright © 2026, Pierre ROYER - https://pierreau.fr

IX. VOLUMES

Un volume permet à des conteneurs de stocker des informations persistantes. Il peut être partagé entre plusieurs containers. Différents types de stockage donnent accès à un volume :

- Un conteneur de données (dans `/var/lib/docker/volumes/`)
- Un répertoire ou un fichier partagé sur l'hôte (bind) : `/home/docker/`
- Une zone en mémoire vive (tmpfs) :



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

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 Mount1 --hostname Mount1 -v
DataVolume1:/ExtVolume busybox:latest
root@cont1:/# ls ExtVolume
root@cont1:/# touch ExtVolume/test
root@cont1:/# ls ExtVolume
test
<CTRL P> <CTRL Q>
[docker@Rocky ~]$  docker stop Mount1
[docker@Rocky ~]$  docker run -ti --name Mount2 --hostname Mount2 -v
DataVolume1:/ExtVolume busybox
root@cont2:/# ls ExtVolume
test
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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 Mount3 -ti -v DataVolume1:/ExtVolume busybox
[docker@Rocky ~]$  docker run -ti --name Mount4 -v DataVolume1:/ExtVolume busybox
```

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 Mount5 -v DataVolume1:/ExtVolume:ro busybox
```

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

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:/usr/local/apache2/htdocs -p 60081:80 httpd:latest

[docker@Rocky ~]$ docker exec web-server ls /usr/local/apache2/htdocs/
index.html
```



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

```
[docker@Rocky ~]$ docker run -dti --name web-server2 -v $(pwd)/HTML:/usr/local/apache2/htdocs -p 60081:80 httpd:latest
```

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

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[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
```



La modification d'un fichier dans un répertoire monté s'effectue en temps réel dans le conteneur, mais ce n'est pas le cas pour un fichier « bindé » : il faut arrêter et redémarrer le conteneur.

3. Montage bind avec docker compose

```
[docker@Rocky ~]$ mkdir ~/html
```

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

4. Montage d'un volume dans la mémoire vive

- Pour des I/O élevées. Performances accrues par rapport à un SSD
- Réduction des fuites de données (non persistantes)
- Pas d'usure des SSD / disques

```
[docker@Rocky ~]$ docker run -dti --name nginx.boost --mount type=tmpfs,destination=/var/cache/nginx,tmpfs-size=64m nginx:latest

[docker@Rocky ~]$ df -h -t tmpfs
```

Avec Docker compose :

```
services:
  app:
    image: nginx:latest
    tmpfs:
      - /var/cache/nginx:mode=755,size=64m
```



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.

5. Montage d'une unité de stockage

Il est possible d'utiliser un device du host directement dans un conteneur :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$ docker run -ti --name hostdev --
device=/dev/nvme0n1:/dev/hostssd alpine:latest
```

La même chose, en lecture seule :

```
[docker@Rocky ~]$ docker run -ti --name hostdev --
device=/dev/sda1:/dev/hostssd:r alpine:latest
```

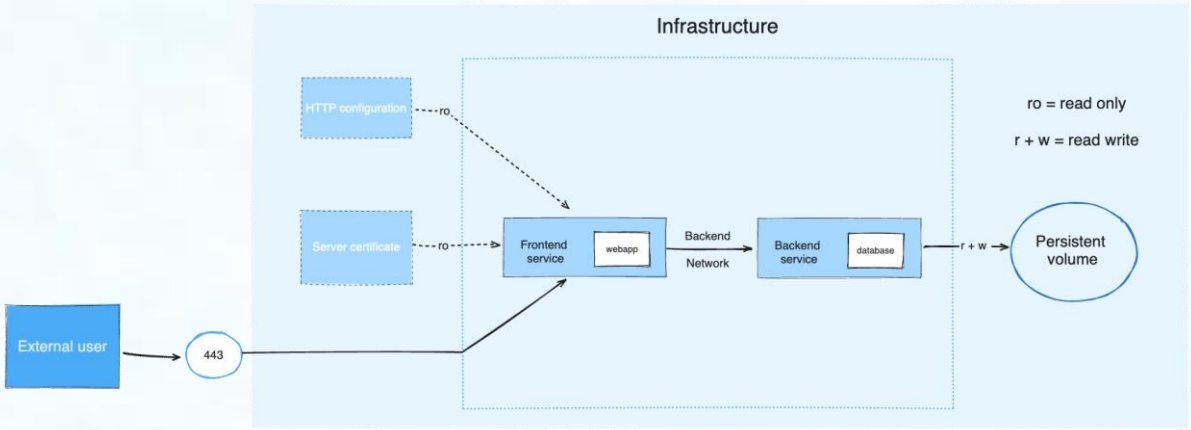
Copyright © 2026, Pierre ROYER - https://pierreau.fr

X. DOCKER COMPOSE



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

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



Source :https://docs.docker.com/compose/intro/compose-application-model/



La syntaxe d'un fichier docker compose doit respecter les spécifications YALM (syntaxe **déclarative** qui décrit l'état souhaité des objets) :

- L'indentation est homogène : ici 2 espaces
- Pas de tabulation à la place des espaces.



A chaque lancement de VI(M), le fichier de configuration `$HOME/.vimrc` est lu. Nous pouvons modifier (ou créer) ce fichier, afin de faciliter l'édition YAML :

```
se nu
set tabstop=3
set expandtab
set autoindent
syntax on
colorscheme industry
```

La liste et l'application des thèmes dans vi s'effectuer en mode commande :
`:colorscheme [space] [Ctrl+d]`
`:colorscheme blue`

A. Apache+PHP

Nous pourrions envisager d'installer PHP dans un conteneur sur lequel Apache est démarré, cependant, nous nous éloignons du principe de micro services faiblement couplés. Nous allons donc implémenter un conteneur pour les contenus statiques, avec une version

Copyright © 2026, Pierre ROYER - https://pierreau.fr

spécifique de Apache, et un pour les contenus dynamiques, avec une version spécifique de PHP.

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

```
<!DOCTYPE html>
<html lang="fr">
    <head>
    <meta charset="utf-8">
    <title>Page HTML</title>
    </head>
    <body>
        <h1>index.html</h1>
        <br>
        <p><a            title="index.php"            rel="noopener"
href="index.php">index.php</a></p>
     </body>
</html>
```

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

```
<?php phpinfo();?>
```

```
[docker@Rocky ~]$  vi vhost.conf
```

```
# Délégation des fichiers .php à PHP-FPM via FastCGI
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so

ServerName apache.php

<VirtualHost *:80>
    DocumentRoot /var/www/html
    DirectoryIndex index.html

    <Directory /var/www/html>
        AllowOverride All
        Require all granted
    </Directory>

    # Transmet les requêtes .php au conteneur "php" sur le port 9000
    ProxyPassMatch                                  ^/(.*\.php(/.*)?)$
fcgi://php:9000/var/www/html/$1
</VirtualHost>
```

```
[docker@Rocky ~]$  vi compose.yaml
```

```
version: '3.8'

services:
  apache:
    image: httpd:2.4
    container_name: apache2.4
    command: >
      sh    -c    "echo    'Include    conf/extra/vhost.conf'    >>
/usr/local/apache2/conf/httpd.conf && httpd-foreground"
    ports:
      - "51180:80"
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
    volumes:
      - ./vhost.conf:/usr/local/apache2/conf/extra/vhost.conf
      - ./html:/var/www/html
    depends_on:
      - php

  php:
    image: php:8.3-fpm
    container_name: php8.3
    volumes:
      - ./html:/var/www/html
```



Depuis la version v2.25.0 de docker compose, la directive `version: 'x.y'` est obsolète. Certains exemples sur Internet utilisent un fichier "docker-compose.yml" ; attention, car ces syntaxes datent de Compose V2 (2022), et sont potentiellement elles aussi obsolètes !

Installation de la pile :

```
[docker@Rocky ~]$  docker compose [-f ./compose.yaml] up -d
```

Etat de la pile :

```
[docker@Rocky ~]$  docker compose ps
```

Les logs :

```
[docker@Rocky ~]$  docker compose logs -ft
```

Liste des conteneurs :

```
[docker@Rocky ~]$  docker compose ps
```

Liste des images utilisées :

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

Liste et état de l'ensemble des piles :

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

Les autres options :

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



Par défaut, le nom d'une pile correspond au nom de répertoire courant. Il est possible de lui donner un nom plus explicite :

```
[docker@Rocky ~]$  docker compose -p "my-app" up -d
```

Ou créer un fichier `.env` qui sera automatiquement pris en compte par docker compose :

```
COMPOSE_PROJECT_NAME=my-app
```



Utiliser le nom de la pile, s'il a été défini précédemment :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$  docker compose -p "my-app" images
[docker@Rocky ~]$  docker compose -p "my-app" ps
```

B. SGBD

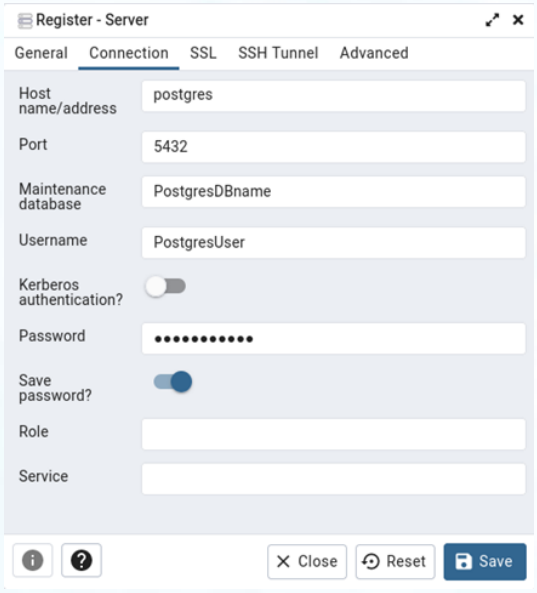
```
[docker@Rocky ~]$  vi compose.yaml
```

```
volumes:
  Postgres-Data:
    name: Postgres-Data
    labels:
      com.example.description: "My Database volume"
      com.example.db: "PostgreSQL 18.6"
      com.example.department: "IT/Ops"

services:
  postgres:
    image: postgres:latest
    pull_policy: always
    container_name: postgres
    hostname: postgres
    restart: always
    # ports:
      # - "5432:5432"
    expose:
      - 5432
    environment:
      - POSTGRES_USER=PostgresUser
      - POSTGRES_PASSWORD=PostgresPWD
      - POSTGRES_DB=PostgresDBname
      - TZ=Europe/Paris
    volumes:
      - Postgres-Data:/var/lib/postgresql/data
      - /etc/localtime:/etc/localtime:ro

  pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin
    depends_on:
      - postgres
    restart: on-failure
    ports:
      - "51080:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: user@pierreau.fr
      PGADMIN_DEFAULT_PASSWORD: SuperSecret
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr



Test du label :

```
[docker@Rocky ~]$ docker volume ls --filter "label=com.example.db=PostgreSQL 18.3"
```

C. Démarrage sélectif des services

1. docker compose

Nous pouvons ne démarrer qu'un seul service de docker compose (sauf s'il y a des dépendances) :

```
services:
  postgres:
    image: postgres:latest

  pgadmin:
    image: dpage/pgadmin4
    depends_on:
      - postgres

  # Container without dependency, not started
  adminer:
    image: adminer:latest
    container_name: adminer
```

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

Dans ce cas, `adminer` ne sera pas démarré.

2. profiles

Les profils définissent les services à démarrer que s'ils sont explicitement spécifiés.

```
services:
  # Always started
    image: postgres:latest
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
  pgadmin:
  # Explicitly started
    image: dpage/pgadmin4
    profiles:
      - prod

  # Explicitly started
  adminer:
  # Explicitly started
    image: adminer:latest
    container_name: adminer
    profiles:
      - test
      - dev
```

```
[docker@Rocky ~]$  docker compose --profile prod up
[docker@Rocky ~]$  docker compose --profile prod --profile test up
```

3. include

Il est possible de générer un socle applicatif avec plusieurs fichiers :

```
[docker@Rocky ~]$  vi compose.yaml
```

```
services:
  service1:
...

include :
  - Service2.yaml
  - Service3.yaml
#  - Service4.yaml
...
```

```
[docker@Rocky ~]$  vi Service2.yaml
```

```
services:
  service2:
...
```

D. docker compose + Dockerfile

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

```
 FROM nginx:alpine

# Copying file(s) from host to container
 COPY nginx.conf /etc/nginx/nginx.conf
```

```
[docker@Rocky ~]$ vi compose.yaml
```

```
services:
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
  monservice:
    container_name: dockerfile
    build:
      context: ./
      dockerfile: ./Dockerfile
      labels:
        - org.opencontainers.image.version="2"
        - org.opencontainers.image.revision="3"
    image: my/image:2.3
...
```

```
[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 ./compose.yaml stop
```

Suppression de la pile :

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

Suppression de la pile, des images et des volumes :

```
[docker@Rocky ~]$  docker compose down --rmi all --volumes -v
```

Supprimer l'ensemble des conteneurs crées avec docker compose :

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



Quelques exemples d'environnements sont consultables ici : https://github.com/docker/awesome-compose

Les références de docker compose sont ici : https://docs.docker.com/compose/compose-file/compose-file-v3/

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XI. Quelques outils d'administration

OK, maintenant que l'on maîtrise Dockerfile, docker compose et les volumes, je vous propose des outils de supervision de nos ressources.

A. Portainer

Interface simple et fonctionnelle, permet de gérer les images, conteneurs, réseaux, volumes, services, stacks… : http://portainer.io/

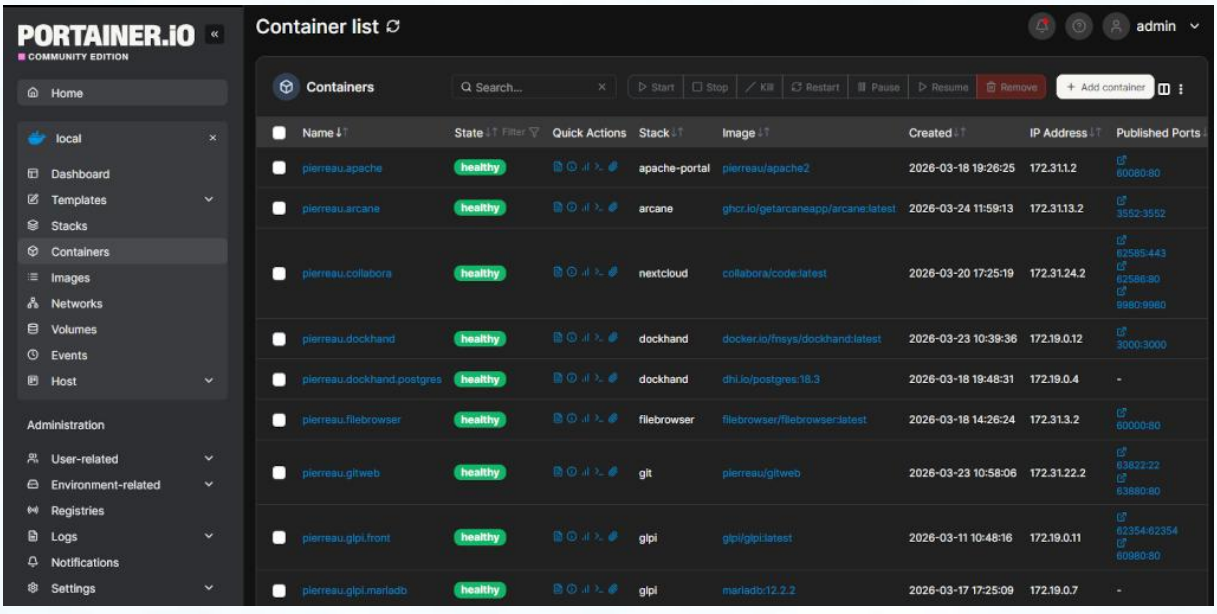
On créé un volume persistant :

```
[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 9000:9000 --cpuset-cpus="0"  --restart=on-failure \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data portainer/portainer-ce
```

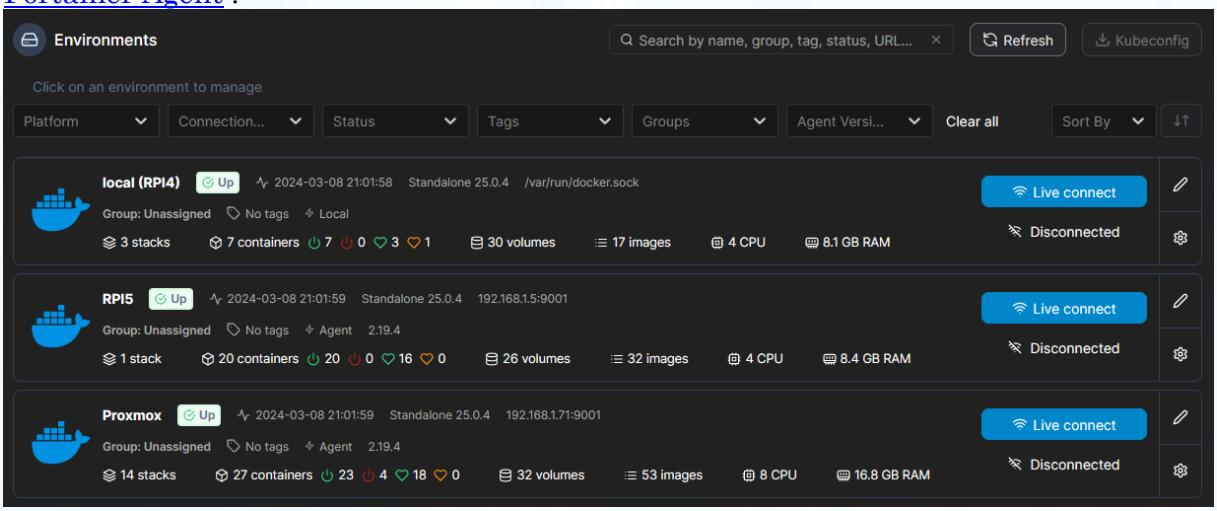
L'accès s'effectue à l'URL http://*localhost*:9000/



Dans cette même interface, il est possible d'y ajouter plusieurs environnements, via

Copyright © 2026, Pierre ROYER - https://pierreau.fr

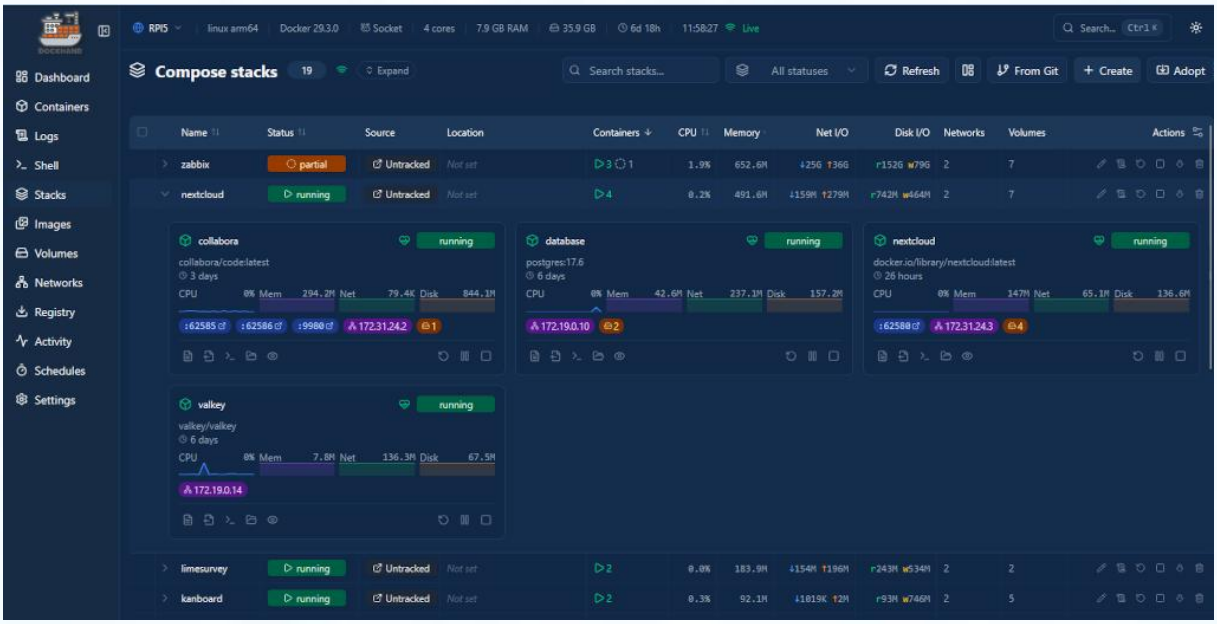
Portainer Agent :



B. Dockhand

Une autre interface rappelant Portainer, mais offrant une meilleure supervision : https://dockhand.pro/#quick-start

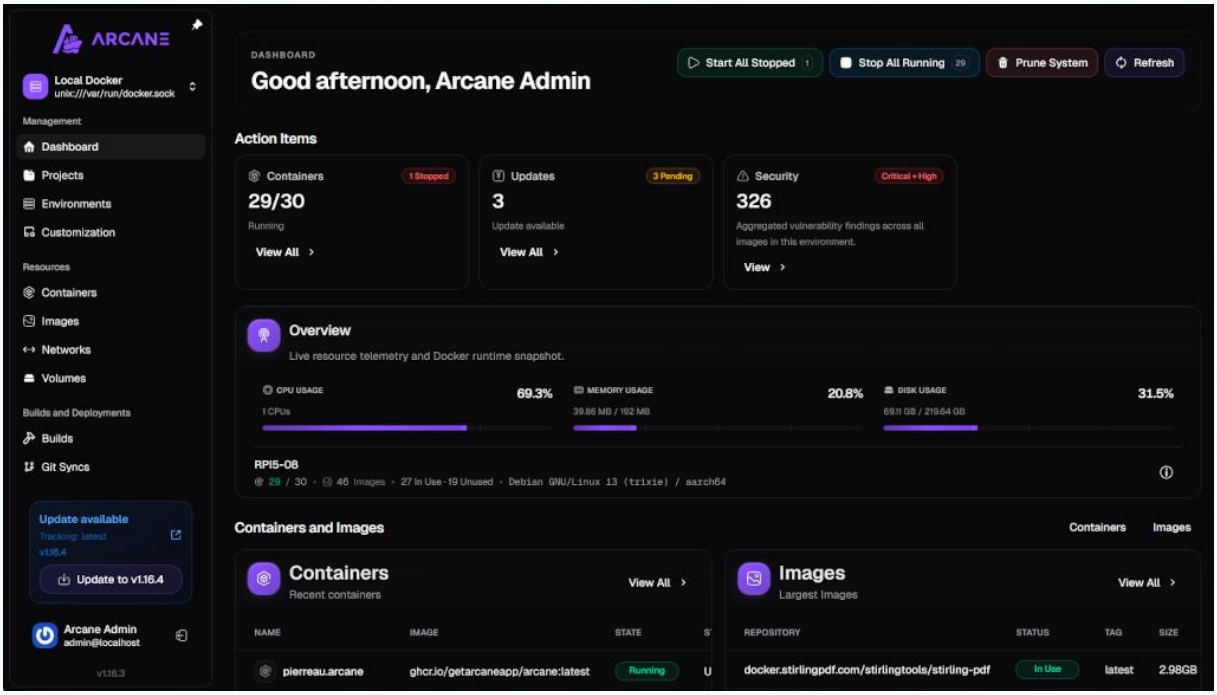
```
[docker@Rocky ~]$  docker run -d --name dockhand -p 3000:3000 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v dockhand_data:/app/data \
  --restart unless-stopped \
  fnsys/dockhand:latest
```



C. Arcane

Avec une interface graphique très réussie, cet outil vaut aussi le détour : https://getarcane.app/

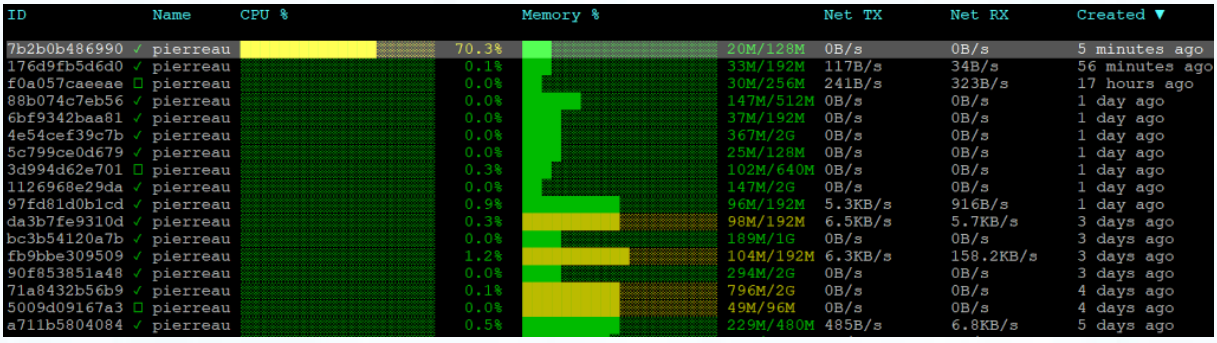
Copyright © 2026, Pierre ROYER - https://pierreau.fr



D. dtop

Dtop est une interface CLI qui affiche en temps réel les ressources des conteneurs, et leurs logs : https://github.com/amir20/dtop

```
[docker@Rocky ~]$ alias dtop='docker run -v /var/run/docker.sock:/var/run/docker.sock --name dtop --rm -it ghcr.io/amir20/dtop'
```



E. Dockly

Dockly est utilisable en mode terminal, dans un conteneur (non compatible ARM64) : https://github.com/lirantal/dockly

```
[docker@Rocky ~]$ docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock -m 128M --memory-swap 128M --cpus .3 lirantal/dockly
```

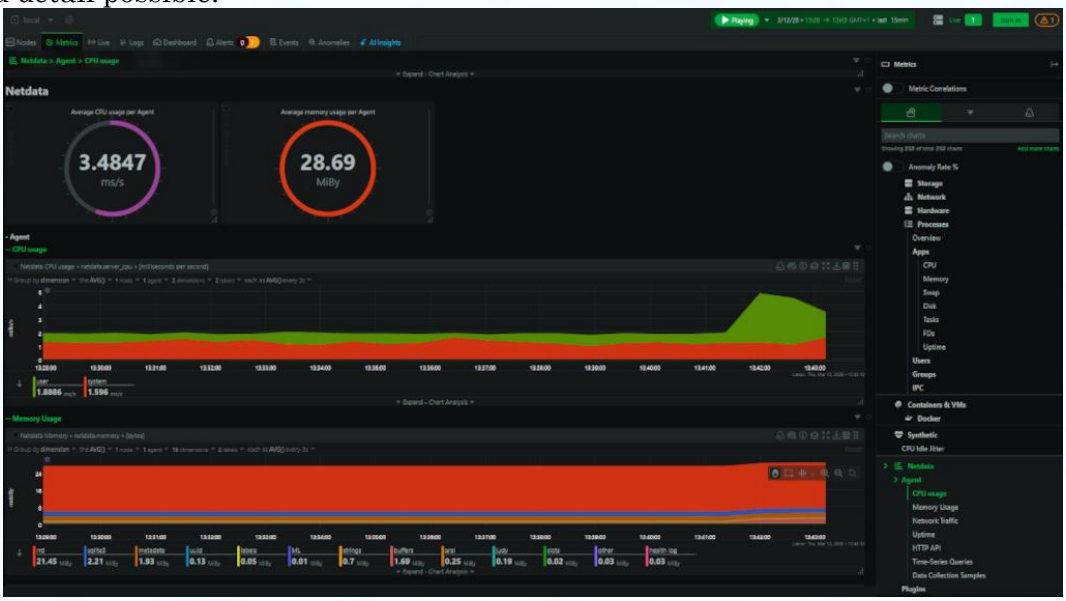
Copyright © 2026, Pierre ROYER - https://pierreau.fr



F. 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… pour le serveur Host et ses conteneurs : https://github.com/netdata/netdata

L'interface est conçue pour visualiser l'activité du système et des conteneurs dans le plus grand détail possible.




```
[docker@Rocky ~]$  vi compose.yaml
```

```
services:
  netdata:
    image: netdata/netdata
    container_name: netdata

    ports:
      - 19999:19999
   # PGID is obtained with: grep docker /etc/group | cut -d ':' -f 3
    environment:
      - PGID=1001
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
    memswap_limit: 256M
    deploy:
      resources:
        reservations:
          cpus: '0.2'
          memory: 128M
        limits:
          cpus: '0.5'
          memory: 256M
    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
```



Les informations sensibles de votre environnement sont accessibles sans identification. Il est impératif de passer par un reverse-proxy avec une authentification au préalable…



D'autres outils pour Docker sont plus orientés CI/CD : https://dokploy.com/



Ce type d'outil conteneurisé nécessite un accès spécifique à Docker Engine, afin de gérer notre environnement. Cela s'effectue via une API qui va utiliser le socket situé dans **/var/run/docker.sock**

Nous pouvons faire de même, en utilisant la commande curl (et jq) :

```
curl        --silent        --unix-socket        /var/run/docker.sock
"http://localhost/containers/json?all=1" | jq -r '.[].Names[0]'

curl --silent --unix-socket /var/run/docker.sock \
  "http://localhost/images/json" | \
  jq  -r  '.[]  |  "Image:  \(.RepoTags[0]  //  "N/A")  -  Créée  le:
\(.Created)"'
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XII. HEALTHCHECK

La directive HEALTHCHECK permet le contrôle des services dans le conteur. L'état de santé ce dernier peut ensuite être interprété pas des outils de contrôles (lazydocker…), de supervision (Zabbix…), d'administration (Portainer…) ou des orchestrateurs (Kubernetes…).

Les paramètres de cette directive :

`start-period` : délai pour l'initialisation (bootstrap) d'un conteneur (par défaut : 0s)

`start-interval` : intervalle entre les vérifications pendant la période de démarrage (par défaut : 5s)

`interval` : intervalle entre les vérifications après la période de démarrage (par défaut : 30s)

`timeout` : la vérification est considérée comme ayant échoué si elle dépasse le délai d'expiration (par défaut : 30s)

`retries` : nombre de tentatives consécutives d'échecs avant que le conteneur soit "unhealthy" (par défaut : 3)

… et la commande qui permet de paramétrer quel service on veut contrôler.

A. Ligne de commande



Le package curl doit être installé pour superviser Apache

```
[docker@Rocky ~]$   docker run -d -p 55080:80 --name healthcheck --health-cmd='curl -f http://localhost:80 || exit 1' --health-start-period=10s --health-interval=30s --health-timeout=5s httpd:latest
```

B. Dockerfile

```
FROM ubuntu:latest

  RUN apt-get update && apt upgrade -y && apt install apache2 && apt clean

HEALTHCHECK --start-period=30s --start-interval=10s --interval=1m --timeout=10s --retries=5 CMD curl -f http://localhost/ || exit 1
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

C. Docker compose

1. Service Apache

```
services:
  apache:
    image: httpd:latest
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      # alternative : test: ["CMD", "wget", "-O", "/dev/null",
"http://localhost:80"]
      start_period: 30s
      interval: 30s
      timeout: 3s
      retries: 3
```

2. Service PostgreSQL

```
services:
  postgresql:
    image: postgres:latest
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U PostgresUser -d
PostgresDatabase"]

ou

      test: ["CMD-SHELL", "pg_isready -U $$POSTGRES_USER -d
$$POSTGRES_DB"]
```

3. Service MariaDB

```
services:
  mariadb:
    image: mariadb:latest
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--su=mysql", "--connect", "-
-innodb_initialized"]
```

4. Service MySQL

```
services:
  mysql:
    image: mysql:latest
    healthcheck:
      test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]
       ou
       test: mysqladmin ping -h 127.0.0.1 -u root --
password=$$MYSQL_ROOT_PASSWORD
       ou
       test: "mysql ${MYSQL_DATABASE} --user=${MYSQL_USER} --
password='${MYSQL_PASSWORD}' --silent --execute "SELECT 1;"
       ou
       test: "/usr/bin/mysql --user=root --
password=$$MYSQL_ROOT_PASSWORD --execute \"SHOW DATABASES;\""
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

5. Service MongoDB

```
services:
  mongodb:
    image: docker.io/bitnami/mongodb
    healthcheck:
      test:     echo     'db.runCommand("ping").ok'     |     mongosh
localhost:27017/test --quiet
```

6. Service Redis

```
services:
  redis:
    image: redis:latest
    healthcheck:
      test: ["CMD", "redis-cli","ping"]
```

7. Service SSH

```
...
      test: netstat -nlp | grep -qE "LISTEN.*sshd"
```

8. Annuaire OpenLDAP

```
...
      test: ["CMD", "ldapsearch", "-Q", "-LLL", "-Y", "EXTERNAL",
"-H", "ldapi:///", "-b",
"cn=pierreau,ou=linux,dc=cloud,dc=pierreau,dc=fr"]
```

D. Dépendances

Il est maintenant possible de gérer les conditions requises au démarrage d'un service non pas par le fait qu'il soit simplement "up", mais en s'assurant qu'il soit pleinement fonctionnel :

```
...
services:
  roundcube:
    image: roundcube/roundcubemail
    depends_on:
      mysql:
        condition: service_healthy

  mysql:
    image: mysql:latest
    healthcheck:
...
```

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

⠸ Container mysql      Starting
⠸ Container mysql      Waiting
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

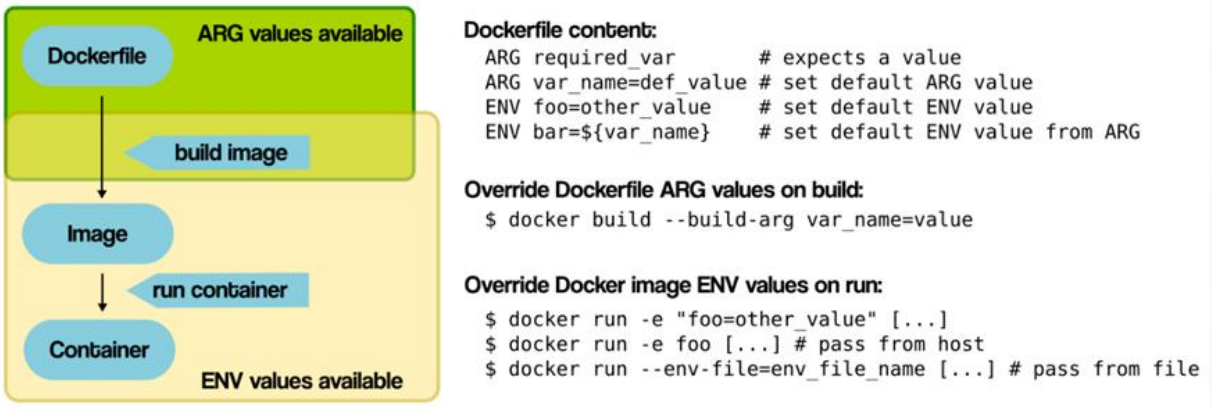
```
✓ Container mysql      Healthy
✓ Container roundcube  Started
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XIII. LES VARIABLES ARG, ENV ET LES SECRETS

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 vont persister dans les conteneurs.



A. ARG

1. Dockerfile

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

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

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

```
[docker@Rocky ~]$ docker build --build-arg TIMESTAMP -t 'pierreau/vars.arg' .

[docker@Rocky ~]$ docker run -ti --name vars.arg pierreau/vars.arg sh

/tmp # cat /DATE.txt
Monday,12/12/2022-12:13
```



Les informations transmises avec ARG ne sont pas visibles dans le conteneur, mais dans l'historique de la création de l'image :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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

2. docker compose

```
[docker@Rocky ~]$ vi compose.yaml
```

```
services:
  env:
    container_name: vars.arg
    build:
      context: .
      dockerfile: Dockerfile
      image: pierreau/vars.arg
      args:
        MY_LOGIN: "Admin"
        MY_PWD: "CrazyPassword"
```

Vérifier la prise en charge des variables !

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

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

B. 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

# Cette variable sera-t-elle prise en compte dans le conteneur :
# TIMESTAMP=$(date +%A,%d/%m/%Y-%H:%M)
```

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

[docker@Rocky ~]$ docker run -ti --name vars.env --env-file ./.env --
-env TIMESTAMP -e MYVAR1 -e MYVAR3=Pierre alpine:latest sh

[docker@Rocky ~]$ docker exec -ti vars.env env | grep MY
MYVAR1=ubuntu
MYVAR3=Pierre
TIMESTAMP=$(date +%A,%d/%m/%Y-%H:%M)
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
TIMESTAMP= Monday,12/12/2022-12:12
MYFOLDER=/root
```

2. Dockerfile & Entrypoint

Les variables peuvent évidemment être exploitées à chaque création du conteneur dans un fichier **entrypoint** :

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

```
FROM alpine:latest

 COPY  entrypoint.sh /root/
 RUN   chmod 700 /root/entrypoint.sh
 ENTRYPOINT ["/root/entrypoint.sh"]
```

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

```
#!/bin/sh
set -euo pipefail

echo "Date de création du conteneur : ${TIMESTAMP}" > /DATE.txt

exec "$@"
```

```
[docker@Rocky ~]$ docker build -t environment .
[docker@Rocky ~]$ docker run -ti --env TIMESTAMP --name environment environment cat /DATE.txt
```

3. Les fichiers d'environnements

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 **compose.yaml**

```
[docker@Rocky ~]$ export TIMESTAMP=$(date +%A,%d/%m/%Y-%H:%M)
[docker@Rocky ~]$ vi .env
```

```
MYSQL_ROOT_PASSWORD=MySqlRootPwd
MYSQL_USER=MySqlUser
MYSQL_PASSWORD=MySqlPwd
MYSQL_DATABASE=MySqlDB
TZ=Europe/Paris
COMPOSE_PROJECT_NAME=mysql-phpmyadmin
```

```
[docker@Rocky ~]$ vi compose.yaml
```

```
services:
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
  mysql:
    image: mysql:latest
    container_name: mysql.env
    restart: always
    ports:
      - 3306:3306
    env_file:
      - .env
    volumes:
      - ./config/mysql/my.cnf:/etc/mysql/conf.d/my.cnf
      - mysqldata:/var/lib/mysql

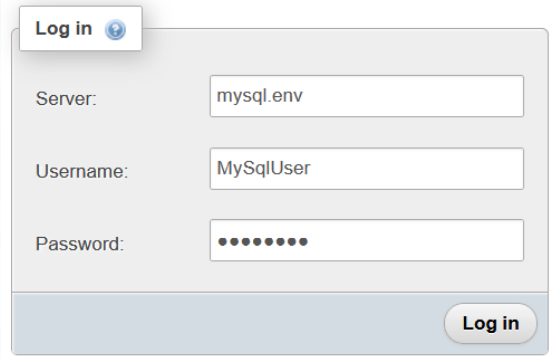
  phpmyadmin:
    image: phpmyadmin:latest
    container_name: phpmyadmin.env
    restart: on-failure:3
    ports:
      - 57080:80
    environment:
      - PMA_ARBITRARY=1
volumes:
  mysqldata:
```

La vérification des variables transmises (sans monter la stack) s'effectue avec la commande :

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

Pour des raisons de modularité et de facilité de maintenance, il peut être intéressant d'avoir un fichier de variables pour chaque service…

Vérifier l'adresse du serveur mySQL…



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

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

Les variables sont récupérables avec `docker inspect` !

```
[docker@Rocky ~]$ docker inspect --format '{{ index (index .Config.Env) }}' mysql.env
```

Les variables sont récupérables dans le conteneur et les process !

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$ docker exec -ti mysql.env sh
sh-4.4# env
MYSQL_ROOT_PASSWORD=MySqlRootPwd
MYSQL_USER=MySqlUser
MYSQL_PASSWORD=MySqlPwd
MYSQL_DATABASE=MySqlDB
```

Ne jamais utiliser la directive ENV pour les informations sensibles ! Préférez l'usage des secrets (chapitre suivant).

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
```

C. Les secrets

Ce mécanisme permet d'intégrer des informations sensibles dans un conteneur, sans qu'elles ne soient lisibles avec `docker inspect`, `docker history` ou la commande `env`.

1. Le principe

```
[docker@Rocky ~]$ vi .secret-password.txt
```

```
SuperSecret!!!
```

```
[docker@Rocky ~]$ vi compose.yaml
```

```
secrets:
  my_password:
    file: .secret-password.txt

services:

  server:
    image: opensuse/leap
    container_name: secret
    hostname: secret

    environment:
      SECRET_PWD_FILE: '/run/secrets/my_password'

    secrets:
      - my_password

    entrypoint: ["/bin/sh", "-c", "sleep infinity"]
```

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$ docker exec -ti secret cat /run/secrets/my_password
SuperSecret!!!
```

Les secrets sont stockés dans le pseudo file system **/run**, monté en tmpfs (RAM).

2. Exemple concret

Définir les variables dans chacun de ces fichiers :

- .db_name
- .db_root_password
- .db_mysql_user
- .db_mysql_password

```
services:
  sgbd:
    image: mysql:latest
    container_name: sgbd
    environment:
      - MYSQL_DATABASE_FILE=/run/secrets/db_name
      - MYSQL_ROOT_PASSWORD_FILE=/run/secrets/db_root_password
      - MYSQL_USER_FILE=/run/secrets/db_mysql_user
      - MYSQL_PASSWORD_FILE=/run/secrets/db_mysql_password
    secrets:
      - db_name
      - db_root_password
      - db_mysql_user
      - db_mysql_password
    ports:
      - 3306:3306

secrets:
  db_name:
    file: ./.db_name.txt
  db_root_password:
    file: ./.db_root_password.txt
  db_mysql_user:
    file: ./.db_mysql_user.txt
  db_mysql_password:
    file: ./.db_mysql_password.txt
```

- Quel est le résultat de la commande `docker compose config` ?
- Vérifiez la connexion à la base de données via le port publié.

Pour certaines images, les variables « secret » peuvent être suffixées avec `_FILE`

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to some of the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

```
$ docker run --name some-postgres -e POSTGRES_PASSWORD_FILE=/run/secrets/postgres-passwd -d postgres
```

Currently, this is only supported for `POSTGRES_INITDB_ARGS`, `POSTGRES_PASSWORD`, `POSTGRES_USER`, and `POSTGRES_DB`.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XIV. LES RESEAUX

A. Créer un réseau

1. Réseau host

Le conteneur partage l'adresse IP du host, et ses ports. Le réseau host est créé lors de l'installation de Docker ; il ne peut pas y avoir de réseau host supplémentaire :

```
[docker@Rocky ~]$  docker network ls
...
491d804beb2b   host             host      local
...
```



Il n'y a pas d'isolation du conteneur sur ce type de réseau, donc aucune sécurité !

2. Réseau Classique (bridge)

Un réseau bridge est créé par défaut, et permet la communication vers les réseaux externes. Toutefois, tous les conteneurs crées dans ce réseau peuvent tous communiquer ensemble, et ne sont donc pas isolés (sécurité). De plus, ce réseau par défaut ne gère pas la résolution DNS entre les conteneurs. La solution est de créer nos propres réseaux, avec nos paramètres spécifiques.

IPV4

```
[docker@Rocky ~]$    docker network create -d bridge --subnet
172.18.0.0/16 --gateway 172.18.0.1 --ip-range=172.18.0.0/24 --label
"com.example.purpose=frontend" frontend
[docker@Rocky ~]$  docker network ls
```

IPV6

Pour la gestion de l'IPV6, il faut ajouter sa prise en charge dans le fichier `/etc/docker/daemon.json`

Mon fournisseur d'accès m'octroie ce réseau pour mes adresses Global Unicast :

```
{
  "ipv6": true,
  "fixed-cidr-v6": "2a01:e0a:173:3910::/64"
}
```

Ce qui me donne une plage :

```
2a01:0e0a:0173:3910:0000:0000:0000:0000 - 2a01:0e0a:0173:3910:ffff:ffff:ffff:ffff
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Afin de modifier les paramètres de ce fichier de configuration, je dois arrêter les conteneurs et redémarrer docker (voire le serveur). Le réseau bridge par défaut est ensuite activé pour l'IPV6.

L'usage de réseaux IPV6 est encore expérimental **pour Docker** (notamment pour le filtrage avec iptable), et apporte peu de valeur ajoutée : on accède aux conteneurs via leurs ports, et non via leurs adresses. Toutefois, il peut être très intéressant d'utiliser l'adresse IPV6 Global Unicast de l'**interface physique de l'hôte**, et d'abandonner enfin l'obsolète IPV4 :

- Plus de translation d'adresse (NAT).
- Multiples adresses sur une même interface réseau
- Prise en charge de SLAAC : Stateless Address Autoconfiguration (plus besoin de serveur DHCP)
- Prise en charge native de IPsec (protocole de sécurisation de bout en bout)
- Prise en charge de la QoS, et du Multicasting
- Meilleures performances : simplification du routage (en-têtes réduits de moitié, car pas de somme de contrôle)
- Meilleure prise en charge des jumbo frames…

https://www.arcep.fr/cartes-et-donnees/nos-cartes/ipv6/carte-interactive-ipv6.html

Le rattachement d'un conteneur démarré à un réseau IPV4 ou IPV6 peut s'effectuer à la volée :

```
[docker@Rocky ~]$  docker network connect frontend container
```

Accès à un conteneur web : https://[2a01:e0a:172:3910::8cca:a54f]/

3. Réseau Internal

https://docs.docker.com/reference/cli/docker/network/create/#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
```

4. Utiliser des réseaux déjà existants

```
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: phpmyadmin
    hostname: front
    depends_on:
      - mariadb
    restart: on-failure
    environment:
      - PMA_ARBITRARY=1
    networks:
      - frontend
    ports:
      - "58080:80"

  mariadb:
    image: mariadb:latest
    container_name: mariadb
    hostname: back
    restart: always
    networks:
      - backend
      - frontend
    expose:
      - "3306"
    environment:
      MARIADB_ROOT_PASSWORD: MariaDBpwd
      MARIADB_DATABASE: MariaDBdb
      MARIADB_USER: MariaDBuser
      MARIADB_PASSWORD: MariaDBpwd
      MARIADB_ALLOW_EMPTY_ROOT_PASSWORD=1
    volumes:
      - MariaDB.Data:/var/lib/mysql
```





Vérifier les ports (non) exposés (conteneurs de bases de données) avec la commande :

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

Vérifier que ce même conteneur soit totalement isolé de l'extérieur :

```
root@back:/# ping 1.1.1.1
ping: connect: Network is unreachable
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr



Il est possible de publier une plage de ports :

```
...
    ports:
      - 62000-63000:62000-63000
...
```

5. Création à la volée

```
networks:
  myfrontend:
    name: myfrontend
    labels:
      com.example.purpose: "App frontend"
    driver: bridge
  mybackend:
    name: mybackend
    internal: true
```



Docker fourni par défaut des réseaux privés de classe B : 172.16.0.0 - 172.31.255.255 avec un CIDR /16. Cela signifie qu'il n'y a que 16 réseaux disponibles en IPV4 ! Les chapitres suivants expliquent comment créer plus de réseaux, plus petits…

6. Réseau overlay

Utilisé pour interconnecter des nœuds (swarm nodes). De moins en moins utilisé car l'orchestrateur swarm est remplacé par d'autres orchestrateurs (Kubernetes, Rancher, Apache Mesos…)

7. Réseau none

Comme le réseau de type host, on ne peut pas créer de réseau none supplémentaire. Le réseau none ne permet pas de communications vers le host, ni vers d'autres conteneurs d'un réseau, bridge ou d'une même stack docker compose.

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

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
services:
  nginx:
    image: nginx:latest
    container_name: 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:
      - 9.9.9.9
    extra_hosts:
      - "cflare:1.1.1.1"
```



Les fichiers `/etc/resolv.conf` et `/etc/hosts` sont réinitialisés à chaque démarrage des conteneurs. Modifier leurs contenus dans les conteneurs n'est donc pas pérenne ; c'est pour cela que les directives `dns` et `extra_hosts` existent.

2. Dans un nouveau réseau

```
networks:
# Creating a network on the fly
  newnetwork:
    ipam:
      driver: default
      config:
        - subnet: 172.22.0.0/24
        - subnet: 2001:3984:3989::/64

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

C. Durcissement

Par défaut, tous les conteneurs sont accessibles depuis tous les réseaux (0.0.0.0). Nous pouvons le vérifier avec la commande suivante :

```
iptables -t nat -L DOCKER -n
Chain DOCKER (2 references)
target   prot opt source      destination
DNAT     tcp  --  0.0.0.0/0   0.0.0.0/0     tcp dpt:60666 to:172.31.66.2:80
DNAT     tcp  --  0.0.0.0/0   10.11.12.13   tcp dpt:60080 to:172.31.1.2:80
```

Il peut être intéressant de limiter la surface d'attaque, en spécifiant uniquement l'adresse IP de l'hôte :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
...
    ports:
      - 10.11.12.13:60080:80
```

D. Rustnet

Rustnet est une interface de surveillance réseau écrite en Rust. Elle permet d'obtenir une visibilité en temps réel sur les connexions réseau, avec des informations détaillées sur l'état des connexions, la gestion de leur cycle de vie, et une inspection approfondie des paquets. https://github.com/domcyrus/rustnet

```
[docker@Rocky ~]$ docker run --rm -it --cap-add=NET_RAW --cap-add=BPF
--cap-add=PERFMON --net=host ghcr.io/domcyrus/rustnet:latest
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XV. LES FORMATS

Les informations de configurations sont nombreuses, et il est possible de filtrer les requêtes avec les formats. Quelques exemples :

```
[docker@Rocky ~]$ docker container ls -a --format "table {{.ID}}\t{{.Image}}\t{{.Command}}\t{{.Status}}"

[docker@Rocky ~]$ docker stats --no-stream --format "table {{.Name}}\t{{.MemUsage}}\t{{.MemPerc}}\{{.CPUPerc}}\t{{.PIDs}}" | sort -k 2,2hr

[docker@Rocky ~]$ docker ps -a --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}" | sort -k1,1

[docker@Rocky ~]$ docker inspect -f "{{.Name}} {{.State.Status}}" container

[docker@Rocky ~]$ docker volume inspect --format "{{.Mountpoint}}" volume
```

Utilisation de **range** / **end** pour boucler sur des informations redondantes :

```
[docker@Rocky ~]$ docker inspect --format "{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}" container

[docker@Rocky ~]$ docker container inspect -f="{{range .NetworkSettings.Networks}}{{.MacAddress}}{{end}}" container

[docker@Rocky ~]$ docker inspect -f='{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' container

[docker@Rocky ~]$ docker inspect -f '{{range .Config.Env}}{{printf "%s\n" .}}{{end}}' container

[docker@Rocky ~]$ docker network inspect -f '{{range .Containers}}{{printf "%s -> %s\n" .Name .IPv4Address}}{{end}}' frontend
```

Utilisation avec des alias :

```
[docker@Rocky ~]$ alias dstat='docker stats --no-stream --format "table
{{.Name}}\t{{.MemUsage}}\t{{.MemPerc}}\{{.CPUPerc}}\t{{.NetIO}}\t{{.
BlockIO}}\t{{.PIDs}}" | sort -k 2h'
```

Mises à jour de toutes les images non « buildées » et taguées par l'utilisateur « pierreau » :

```
[docker@Rocky ~]$ docker images --format "{{.Repository}}:{{.Tag}}" | grep -v pierreau | xargs -L1 docker pull
```

Les formats disponibles sont spécifiés dans la documentation officielle : https://docs.docker.com/engine/reference/commandline/ps/

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XVI. Sauvegardes & restaurations des volumes

Pour les volumes de type « bind », la question ne se pose pas. Pour les autres, voici quelques idées qui permettent d'automatiser les processus via crontab, par exemple.



Il est important de couper toutes les transactions clients avant d'effectuer ces opérations.

A. Copie de volume

```
[docker@Rocky ~]$ SOURCE_VOL="wordpress_mariadb" ; DEST_VOL="wordpress_mariadb_copy"
[docker@Rocky ~]$ docker run --rm -v "$SOURCE_VOL":/source -v "$DEST_VOL":/destination alpine cp -a /source/. /destination
```

B. Copie d'un répertoire monté

Partons de ce principe :

```
volumes:
  wordpress:
    name: wordpress
services:
  wordpress:
    volumes:
      - wordpress:/var/www/html
```

Pour la sauvegarde :

```
#!/bin/bash
CONTAINER=wordpress
DIRECTORY=/var/www/html
MyVolume=/root/$(date +%Y%m%d-%H-%M)_var-www-html.tgz
docker exec -ti --env $MyVolume --env $DIRECTORY $CONTAINER sh -c "tar czf $MyVolume $DIRECTORY"
docker cp $CONTAINER:$MyVolume .
docker exec -ti $CONTAINER sh -c "rm $MyVolume"
```

Dans l'autre sens :

```
#!/bin/bash
CONTAINER=wordpress
DIRECTORY=/var/www/html
export MyVolume=$(ls *var-www-html.tgz | sort | tail -1)
docker cp $MyVolume $CONTAINER:/root/$MyVolume
docker exec -ti --env $DIRECTORY $CONTAINER sh -c "rm -rf $DIRECTORY/*"
docker exec -ti --env $MyVolume --env $DIRECTORY $CONTAINER sh -c "tar xzf /root/$MyVolume -C /"
docker exec -ti --env $MyVolume $CONTAINER sh -c "rm /root/$MyVolume"
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Autres solutions, plus simples :

```
[docker@Rocky ~]$  docker run --rm --volumes-from wordpress -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /var/www/html

[docker@Rocky ~]$  docker run --rm --volumes-from wordpress -v $(pwd):/backup ubuntu bash -c "cd /var/www/html && tar xvf /backup/backup.tar --strip 1"
```

C. Sauvegardes SQL

1. PostgreSQL

```
[docker@Rocky ~]$  docker exec container bash -c "pg_dump -U user -d database" > pg_dump.sql

[docker@Rocky ~]$  docker exec container bash -c "dropdb -U user -d database --force"
[docker@Rocky ~]$  docker exec container bash -c "createdb -U user -d database"

[docker@Rocky ~]$ docker exec -i container bash -c "psql -U user -d database" < pg_dump.sql
```

2. MariaDB

```
[docker@Rocky ~]$  docker exec -ti glpi.mariadb mariadb-dump -u root -p'PVzzM0rP' database > MariaDB.sql

[docker@Rocky ~]$ docker exec -ti glpi.mariadb mariadb -u root -p'PVzzM0rP ' glpi < MariaDB.sql
```

3. MySQL

Sauvegardes :

```
#!/bin/bash
CONTAINER=limesurvey.mariadb
MyFileName=surveys_dump.sql
MyDump=/root/`date +%Y%m%d-%H-%M`_$MyFileName
MyDB=limesurvey
MyDBUser=limesurvey
MyDBpass=MySuperPass
docker exec -ti --env $MyDump -e $MyDB -e $MyDBUser -e $MyDBpass $CONTAINER sh -c "mysqldump --no-tablespaces -u $MyDBUser -p$MyDBpass $MyDB > $MyDump"
if [ $? -ne 0 ] ; then echo -e "\n\n\t Echec sur le dump !" && exit 1 ; fi
docker exec -ti --env $MyDump $CONTAINER sh -c "tar czf $MyDump.tgz $MyDump >/dev/null"
if [ $? -ne 0 ] ; then echo -e "\n\n\t Echec sur la compression !" && exit 2 ; fi
docker cp $CONTAINER:$MyDump.tgz .
if [ $? -ne 0 ] ; then echo -e "\n\n\t Echec sur l'exportation !" && exit 3 ; fi
docker exec -ti --env $MyDump $CONTAINER sh -c "rm $MyDump"
docker exec -ti --env $MyDump $CONTAINER sh -c "rm $MyDump.tgz"
exit 0
```

Dans l'autre sens :

```
#!/bin/bash
CONTAINER=limesurvey.mariadb
```

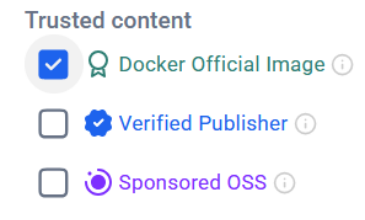
Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
MyFileName=surveys_dump.sql
if ! [ -f *$MyFileName.tgz ] ; then echo -e "\n\t Il n'existe pas de ficher
$MyFileName.tgz à restaurer ! \n" && exit 1 ; fi
MyVolume=limesurvey_data
MyDB=limesurvey
MyDBUser=limesurvey
MyDBpass=MySuperPass
MyTarDump=$(ls *$MyFileName.tgz | sort | tail -1)
MySQLDump=$(echo "$MyTarDump" | sed -r 's/.sql.tgz/.sql/')
docker compose down
docker volume rm $MyVolume
docker compose up -d
echo "Lauching database, wait a few seconds..."
while (docker ps | grep $CONTAINER | grep -v healthy) ; do sleep 3 ; done
docker cp $MyTarDump $CONTAINER:/root/$MyTarDump
if [ $? -ne 0 ] ; then echo -e "\n\t L'importation de l'archive à échoué ! \n" &&
exit 2 ; fi
docker exec -ti --env $MyTarDump $CONTAINER sh -c "tar xzf /root/$MyTarDump && rm
/root/$MyTarDump"
if [ $? -ne 0 ] ; then echo -e "\n\t La décompression de l'archive à échoué ! \n"
&& exit 3 ; fi
docker exec -ti -e $MyDB -e $MyDBpass $CONTAINER sh -c "mysql -u root -p$MyDBpass
--execute 'DROP DATABASE $MyDB;'"
docker exec -ti -e $MyDBpass $CONTAINER sh -c "mysql -u root -p$MyDBpass --execute
'CREATE DATABASE $MyDB;'"
docker exec -ti -e $MySQLDump -e $MyDB -e $MyDBpass -e $MyDBUser $CONTAINER sh -c
"mysql -u $MyDBUser -p$MyDBpass $MyDB < /root/$MySQLDump"
docker exec -ti -e $MySQLDump $CONTAINER sh -c "rm /root/$MySQLDump"
docker exec -ti -e $MyDBUser -e $MyDBpass $CONTAINER sh -c "mysql -u $MyDBUser -
p$MyDBpass --execute 'SHOW DATABASES;'"
docker exec -ti -e $MyDB -e $MyDBUser -e $MyDBpass $CONTAINER sh -c "mysql -u
$MyDBUser -p$MyDBpass --execute 'SHOW TABLES FROM $MyDB;'"
exit 0
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XVII. SECURITE

D'une part, il est indispensable de privilégier les images de confiance :

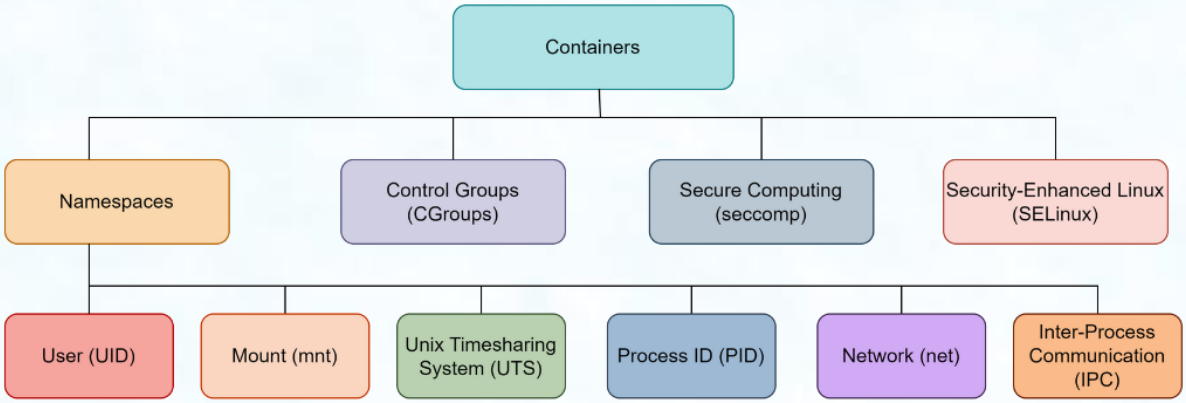


D'autre part, DevSecOps implique un environnement de pré-production et de production. Avec Docker, il est simple de dupliquer ces deux derniers environnements, et d'avoir une pré-prod "ISO" prod (identique).

Les chapitres précédents nous ont permis de comprendre certaines notions de sécurité : réseau "internal", montage en lecture seule… Ces principes ne couvrent pas l'ensemble des bonnes pratiques, et des usages à adopter en environnement de production.

L'ANSSI propose un document avec des recommandations qui vont dans le sens des chapitres qui suivent :
https://messervices.cyber.gouv.fr/guides/recommandations-de-securite-relatives-au-deploiement-de-conteneurs-docker

Nous allons nous appuyer sur les mécanismes que portent nativement les systèmes Unix / Linux afin de sécuriser nos environnements.



Toutes les solutions de conteneurisation utilisent les principes suivants :

A. Linux capabilities

Exécuter un conteneur avec les droits identiques au root de l'hôte est évidemment vivement déconseillé (principe du moindre privilège). Les "capabilities" permettent d'accéder à certains privilèges dans le conteneur, sans pour autant attribuer l'ensemble des droits. Je peux faire le parallèle à la commande `sudo` qu'il ne faudrait jamais utiliser avec tous les droits (`ALL`), mais uniquement en fonction des commandes root nécessaires.

Afficher les "capabilities" des process en cours :

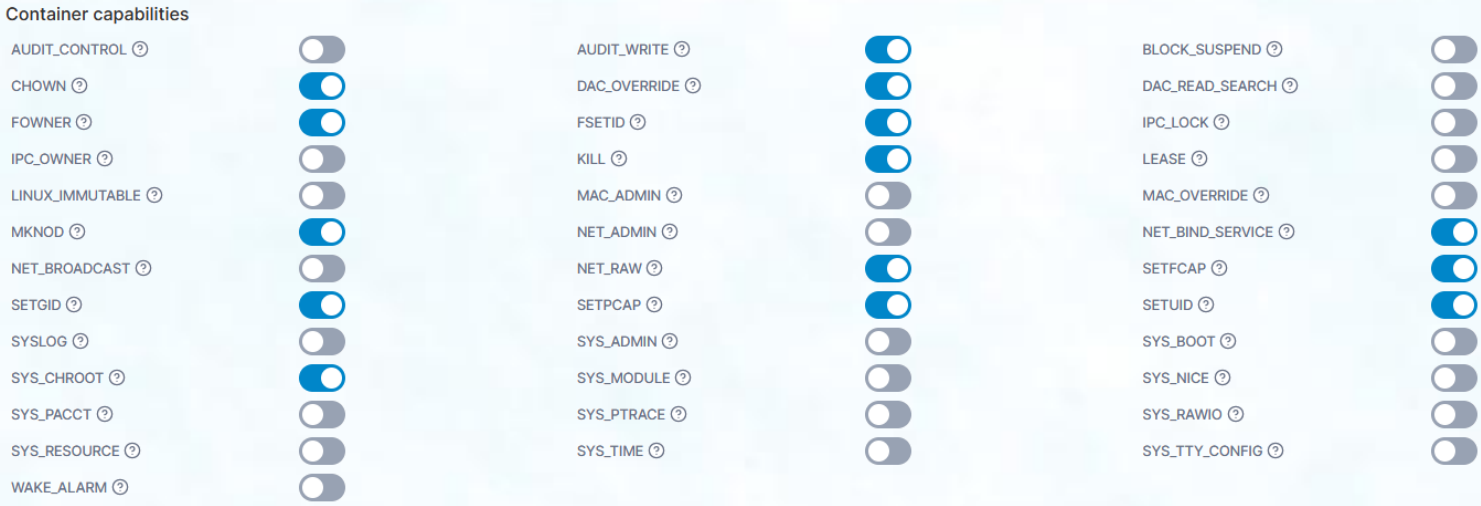
```
root@Debian:~  apt install libcap-ng-utils
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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

1. Les privilèges

Par défaut, Docker active 14 privilèges :



Portainer : Containers - *Container* - Duplicate/Edit / - Capabilities

`AUDIT_WRITE` Write records to kernel auditing log.
`CHOWN` Make arbitrary changes to file UIDs and GIDs.
`DAC_OVERRIDE` Bypass file read, write, and execute permission checks.
`FOWNER` Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
`FSETID` Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
`KILL` Bypass permission checks for sending signals.
`MKNOD` Create special files using mknod.
`NET_BIND_SERVICE` : Bind a socket to internet domain privileged ports (port numbers less than 1024).
`NET_RAW` Use RAW and PACKET sockets.
`SETFCAP` Set file capabilities.
`SETGID` Make arbitrary manipulations of process GIDs and supplementary GID list.
`SETPCAP` Modify process capabilities.
`SETUID` Make arbitrary manipulations of process UIDs.
`SYS_CHROOT` Use chroot, change root directory.

Afin de paramétrer les "capabilities" :

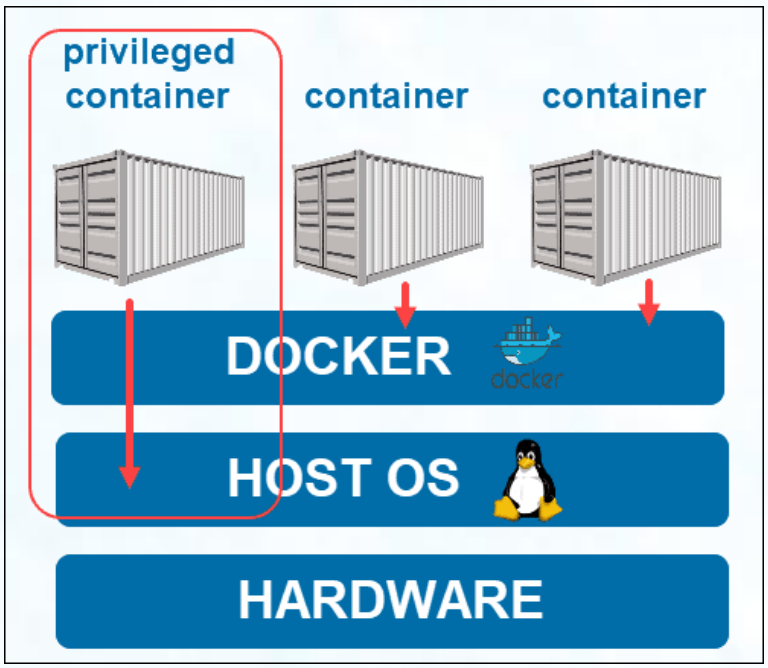
```
[docker@Rocky ~]$  docker run -ti --name privileged --cap-drop AUDIT_WRITE --cap-drop MKNOD --cap-drop SETFCAP --cap-add=SYS_BOOT ubuntu:latest
```

Plus d'informations ici : https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

2. Le mode privilégié

Par défaut, un conteneur n'a pas accès directement aux ressources de l'hôte. Toutefois, il peut être nécessaire, dans certains cas, de l'autoriser.

Copyright © 2026, Pierre ROYER - https://pierreau.fr



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

Pour des raisons d'isolation et de sécurité, ces usages sont à proscrire par défaut.

Comme évoqué avec les Namespaces UID, l'UID / GID dans le conteneur sont 0:0 (root). Afin d'en limiter les privilèges, nous pouvons exécuter le conteneur avec un autre UID / GID :

```
[docker@Rocky ~]$  docker run -ti --name privileged --privileged --user=1001:1001 alpine
```

En mode non privilégié, il est par exemple possible d'installer une instance Docker dans un conteneur (à titre très expérimental) …

Je vous conseille de visionner cette vidéo, afin d'étendre ce chapitre : https://www.youtube.com/watch?v=PL-h_mqcJzY

B. Namespace

1. Principes

Un Namespace (espace de nommage) permet l'isolation de ressources partagées (conteneurs), dans un environnement commun (serveur). Les Namespaces permettent de cloisonner des ressources spécifiques du système (application ou processus), et sont particulièrement utiles pour créer des environnements sécurisés et indépendants.

- Namespace USER
  - Le namespace USER isole les identités (UID/GID) et les privilèges (capabilities).
- UTS (UNIX Time-sharing System)

Copyright © 2026, Pierre ROYER - https://pierreau.fr

    - UTS est le nom historique de la structure "utsname", et n'a aucun rapport avec la gestion du temps Ce namespace permet à un groupe de processus d'avoir son propre hostname, sans affecter celui de l'hôte ni des autres conteneurs.
- IPC (Inter-Process Communication)
    - Ce namespace donne à un groupe de processus ses propres objets inter-processus :
        - Mémoire partagée lue et écrite par plusieurs processus
        - Files de messages System V et POSIX
        - Sémaphores qui protègent une ressource par un verrou, ou limitent le nombre d'accès simultanés à celle-ci.
- PID (Processus Identifier)
    - Les processus ne peuvent interagir qu'avec les processus de leur propre namespace ou des namespace enfants. Ils permettent notamment d'isoler (et de protéger) le PID 1 (/sbin/init), processus parent de tous les autres processus de l'hôte.
- NET (network)
    - Un namespace réseau regroupe des processus qui ont leur propre pile réseau, totalement isolée de la pile réseau du reste du système. Ces processus ont leurs propres interfaces réseau, et leurs propres règles iptable.
- MNT (Mount) : isole / gère les points de montage et l'accès au stockage.
- Time : gestion de l'horloge et de la synchronisation : NTP, boot du serveur, boot + mise en veille.

Explique le namespace « CGROUP. USER »

La liste des Namespaces existants est visible avec cette commande :

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

Les Namespaces de chaque processus sont renseignés ici :

```
[root@Rocky ~]# ll /proc/xxx/ns/
```

2. Création d'un conteneur natif

Création d'un Namespace USER

```
[root@Rocky ~]#  unshare --map-root-user bash
```

`--map-root-user` n'est pas obligatoire si on est root. Il doit être utilisé pour créer un Namespace sans être root, et appliquer les " capabilities" correspondant au Namespace créé.

Namespace UTS

Capabilities : CAP_SYS_ADMIN

```
[root@Rocky ~]#  unshare --map-root-user --uts
[root@Rocky ~]#  hostname NewName
[root@Rocky ~]#  hostname
Rocky
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Le prompt garde l'ancien nom, car bash affecte le hostname qu'à son démarrage.

```
[root@Rocky ~]#  exec bash
[root@NewName ~]#
```

NET

Capabilities : CAP_NET_ADMIN, CAP_NET_RAW, CAP_NET_BIND_SERVICE

```
[root@Rocky ~]#  ip -br link list
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
eth0             UP             88:a2:9e:25:8c:fe <BROADCAST,MULTICAST,UP,LOWER_UP>
wlan0            DOWN           88:a2:9e:25:8d:00 <BROADCAST,MULTICAST>
...
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens18            UP             4a:6f:42:d6:e7:90 <BROADCAST,MULTICAST,UP,LOWER_UP>
[root@Rocky ~]#  unshare --net
[root@Rocky ~]#  ip -br link list
lo               DOWN           00:00:00:00:00:00 <LOOPBACK>
[root@Rocky ~]#  ip link set lo up
[root@Rocky ~]#  ip -br link list
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
[root@Rocky ~]#  exit
```

PID

Capabilities : CAP_SYS_ADMIN

```
[root@Rocky ~]#  unshare --pid
-bash: fork: Cannot allocate memory
```

A ce stade, il faut créer un fork du Shell pour créer un nouveau processus enfant, et assurer une cohérence avec le nouveau Namespace :

```
[root@Rocky ~]#  unshare --pid --fork --mount-proc
```

- l'option **--fork** créé un nouveau processus. Cette action nécessaire pour permettre à notre bash d'avoir un nouveau PID
- l'option **--mount-proc** demande à unshare de faire le nécessaire pour que le pseudo filesystem /proc soit cohérent avec le Namespace de PID.

```
[root@Rocky ~]#  lsns --type pid
        NS TYPE NPROCS PID USER COMMAND
4026533642 pid       2   1 root -bash
```

Le PID interne est bien 1. Si l'on ouvre un autre terminal sans sortir du Namespace, le "vrai" PID du process est bien différent :

```
        NS TYPE NPROCS     PID USER     COMMAND
4026533642 pid       1 3279264 root     -bash
```



Les droits root sont hérités à l'intérieur du Namespace (s'il est créé par root) :

```
[root@Rocky ~]#  touch /root/ESSAI
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Les conteneurs Docker ne doivent donc JAMAIS être gérés par le compte root.

UID

Capabilities : CAP_SETUID, CAP_SETGID, CAP_CHOWN, CAP_FOWNER…

```
[docker@Rocky ~]#  id
uid=1001(docker) gid=1001(docker) groups=1001(docker)
[docker@Rocky ~]#  unshare -U
nobody@Rocky:~$ id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
```

Par défaut, les UID / GID ne sont pas d'origine. Nous pouvons toutefois lancer un conteneur avec des UID / GID spécifiques :

```
[docker@Rocky ~]#  docker run -ti --name ugid --user 1001:1001 alpine
~ $ id
uid=1001 gid=1001 groups=1001
```

Ou bien

```
[docker@Rocky ~]#  docker run -ti --name ugid --user $(id -u):$(id -g) alpine
```

Ou bien

```
[docker@Rocky ~]$  docker run -ti --name privileged --privileged --user="pierreau" alpine:latest
```



Attention lors de l'utilisation de volumes : l'utilisateur doit disposer des droits adéquats.



Ces UID / GID sont récupérables dans un Dockerfile, à condition de les spécifier en tant qu'argument (ARG).

IPC

Capabilities : CAP_SYS_ADMIN

Comment créer un namespace IPC ?

La file de messages peut être lue via

```
[root@Rocky ~]#  ipcs -q
```

Récapitulatif

L'encapsulation des Namespaces est l'une des clés du fonctionnement des conteneurs.

```
[root@Rocky ~]#  unshare --user --map-root-user --pid --fork --mount-proc --uts --ipc --net bash
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Création d'un filesystem

Le système de fichiers :

```
[docker@Rocky ~]#  apt install debootstrap -y
[docker@Rocky ~]#  mkdir -p ~/my-container/rootfs
[docker@Rocky   ~]#      debootstrap   stable   ~/my-container/rootfs
http://deb.debian.org/debian
```

Les Namespaces :

```
[docker@Rocky ~]#  unshare --pid --fork --mount --uts --ipc --net --
mount-proc --root=my-container/rootfs /bin/bash
```

C. Cgroups

Les Cgroups Limitent l'usage des ressources pour un ensemble de processus. L'état des Cgroups est accessible dans le dossier /sys/fs/cgroup/



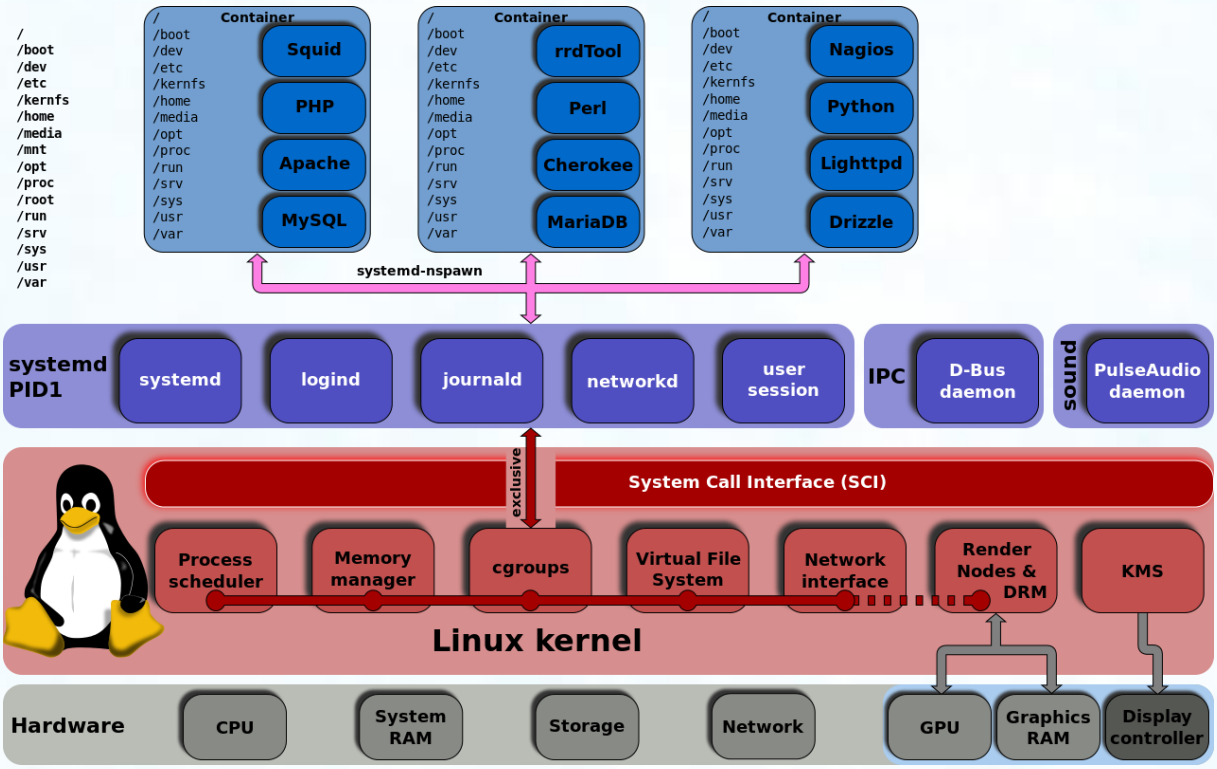
Par défaut, lors de la création d'un conteneur, TOUTES les ressources de l'hôte lui sont attribué : Les Cgroups doivent impérativement être implémentés en contexte de production, sans quoi un seul conteneur peut prendre l'ensemble des ressources des autres conteneurs, voire de l'hôte…



En fonction de la version des processeurs ou de la configuration du kernel, certaines Cgroups ne sont pas actifs. Ceux qui sont supportés sont dans le fichier /sys/fs/cgroup/cgroup.controllers

Positionnement des Cgroups :

Copyright © 2026, Pierre ROYER - https://pierreau.fr



Source : Wikipédia

La section qui nous concerne est celle qui contrôle les ressources Hardware.

Les Cgroups actifs sont visibles comme cela :

[docker@Rocky ~]# lscgroup

La hiérarchie des Cgroups est visible avec ceci :

```
[docker@Rocky ~]#  systemd-cgls
```

1. Ressources en mémoire vive

Afin d'éviter les crashs et les débordements systèmes du type « Out Of Memory Exception », il est important de procéder à quelques réglages. Nous pouvons modifier les paramètres des ressources d'un conteneur, lors de sa création (`docker run`), ou à la volée (`docker update`) une fois démarré :

```
[docker@Rocky ~]$  docker update --memory-reservation 16m --memory 32m --memory-swap 64m Container
```

`memory-reservation` est la limite « soft » de la mémoire vive allouée
`memory` est la limite « hard » de la RAM allouée (minimum 6 megabytes)



- `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 pourrait y avoir des échanges avec le stockage physique, et donc des ralentissements.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

Vérifications :

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



- Un conteneur ne peut pas avoir moins de 6 Mo de RAM
- Un autre paramètre assez proche de **memory-swap** existe : **mem_swappiness** (défini en pourcentage) ; moins strict, et qui contrôle la propension (tendance) du conteneur à utiliser la swap.

Tester les limites en allouant une quantité de mémoire vive pendant 10 sec :

```
[docker@Rocky ~]$  stress-ng --vm 1 --vm-bytes 512M --vm-keep -t 10s
```

2. Ressources en mémoire partagée

Par défaut, docker alloue 64 Mo de mémoire partagée (shared memory) par conteneur. Cette limite n'est pas optimale pour des usages tels que les bases de données, Apache Spark, Kafka, les navigateurs web… Il est conseillé d'ajuster cette taille, afin de limiter les accès stockage / réseau, et d'améliorer les performances.

Concernant les moteurs PostgreSQL, il est recommandé de définir la taille de la mémoire partagée à environ un quart de la mémoire RAM totale.

```
[docker@Rocky ~]$  docker run --name sh-memory --shm-size=256m -m 1g
--memory-swap      1g      -e      POSTGRES_PASSWORD=mysecretpassword
dhi.io/postgres:18.3
[docker@Rocky ~]$  docker exec -ti sh-memory df -h /dev/shm
```

3. Ressources processeur

```
[docker@Rocky ~]$  docker update --cpuset-cpus 3 Container
[docker@Rocky ~]$  docker update --cpuset-cpus 0,2 Container
[docker@Rocky ~]$  docker update --cpuset-cpus 2-3 Container
[docker@Rocky ~]$  docker update --cpus 1.5 Container
[docker@Rocky ~]$  docker update --cpus 0.01 Container
```

cpuset-cpus désigne spécifiquement des vCUPs prioritaires
cpus permet d'allouer un nombre (ou une fraction) de processeurs, dans la limite de ceux disponibles.



Comment saturer les CPUs avec une simple commande ping (en root) ?
En Bash :

```
for ((i=1 ; i<=$(nproc) ; i++)) ; do ping -i . 000001 localhost & >
/dev/null ; done
```

En Bourne shell :

```
i=1; while [ $i -le $(nproc) ]; do ping -i .000001 localhost >
/dev/null & i=$(expr $i + 1); done
```

```
pkill -9 ping
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

D'autre paramètres permettent de peaufiner la gestion de la mémoire et les ressources des processeurs : https://docs.docker.com/config/containers/resource_constraints/

4. Ressources PIDs

Le nombre maximum de process par défaut avec Docker est de… 4096 ! Limiter ce nombre nous préserve potentiellement des risques de "Fork-bomb", en fonction du nombre de conteneurs lancés en parallèle... Dans un contexte microservice, le nombre de process nécessaires pour un conteneur est forcément limité.

Quelle est la limite actuelle de PIDs que gère votre Kernel ?

```
[docker@Rocky ~]$  sysctl kernel.pid_max
[docker@Rocky ~]$  cat /proc/sys/kernel/pid_max
```

Quel est le dernier PID attribué ?

```
[docker@Rocky ~]$  ps axeo pid --sort pid | tail -1
```

PS : le kernel réutilise les PIDs libérés

Comment écrouler une machine en une seule commande via un Fork-bomb ?

```
:(){ :|:& };:
```

L'équivalent en script bash, plus compréhensible, est le suivant :

```
#!/bin/bash
bomb()
{
   bomb | bomb & # Récursion de forks en arrière-plan. Croissance
exponentielle du nombre de processes...
}
Bomb
```

- L'objectif n'est pas de brider les conteneurs en RAM, mais de protéger les ressources globales de ses conteneurs voisins… et de l'hôte ! Autant un conteneur Nginx peut se contenter de 16 Mo, autant l'installation ou la mise à jour de packages vont planter. Certains conteneurs SGBD ou applicatifs peuvent nécessiter plus que 1 Go.
- Des conteneurs, tels que des bases de données, nécessitent plusieurs PIDs. Attention à ne pas les sous-estimer…

Le paramètre `pids` restreint le nombre maximal de process dans un conteneur.

5. Ressources stockage

- Le système de fichier doit être formaté afin de pouvoir prendre en charge les quotas (xfs, zfs, btrfs …)

Copyright © 2026, Pierre ROYER - https://pierreau.fr

- Le driver pour le stockage doit être btrfs, overlay2, ou zfs
- L'option 'pquota' doit être activée dans le fichier /etc/fstab

```
[docker@Rocky ~]$ docker run -dti --name fsquota --storage-opt size=100M httpd
```

6. Docker compose

```
[docker@Rocky ~]$ vi compose.yaml
```

```
services:
  ...
    memswap_limit: 512M
    deploy:
      resources:
        reservations:
          cpus: '0.2'
          memory: 256M
        limits:
          cpus: '0.3'
          memory: 512M
          pids: 64
    shm_size: 128mb
    storage_opt:
      size: '300M'
```

`resources` : espace garanti
`limits` : espace maximum

7. Ressources réseau

Elles ne sont pas encore nativement implémentées sur Docker, cependant il est tout à fait possible d'utiliser les commandes Linux standard :

Dans le conteneur :
- Installer le package Traffic control (iproute2)
- Modifier la bande passante : `tc qdisc add dev eth0 root tbf rate 100kbit burst 32kbit latency 50ms`

Pour les tests :
- Installer le package iperf
- Tester la bande passante sur l'un des sites (https://iperf.fr/iperf-servers.php) : `iperf -c ping.online.net`

8. Métriques

```
[docker@Rocky ~]$ systemd-cgtop --order=memory
```

D. Audits

Après ces bonnes pratiques, il est temps d'auditer notre environnement :

Copyright © 2026, Pierre ROYER - https://pierreau.fr

1. Trivy

Scan de vulnérabilités (CVE) :

```
[docker@Rocky ~]$ docker run --name trivy --rm -v /var/run/docker.sock:/var/run/docker.sock -v /tmp/Trivy:/root/.cache/ ghcr.io/aquasecurity/trivy --table-mode summary image debian:latest
```

2. bench-security

https://github.com/docker/docker-bench-security

E. Front-end

Plusieurs solutions sont à mettre en œuvre afin de limiter les intrusions. Effectivement, en tant que micro-services, les conteneurs ne sont pas censés incorporer un service de pare-feu. De même, intégrer un pare-feu avec Docker engine ou Swarm est pour l'instant difficilement réalisable dans un environnement par nature éphémère (cattle no pets).

Dockerhub propose des images afin de déployer des reverse proxy.

Il est à intégrer dans une DMZ réelle, ou au moins dans un réseau isolé, dans un conteneur dédié. Ses objectifs sont multiples :

- Gérer et renouveler les certificats SSL
- S'identifier, s'authentifier (simple ou double facteurs)
- Rediriger les URL inscrites dans le DNS, uniquement vers les réseaux Front et les bons conteneurs.
- Contrôler et sécuriser les flux
- Surveiller le trafic
- …

1. Solutions

- Traefik :
  - https://traefik.io/traefik/
  - https://hub.docker.com/_/traefik/
- SWAG (Secure Web Application Gateway)
  - https://docs.linuxserver.io/general/swag
  - https://hub.docker.com/r/linuxserver/swag
- https://caddyserver.com/

2. Entêtes HTTP

Avoir un serveur en HTTPS ne garantit pas un niveau de sécurité optimal (cf. https://securityheaders.com/).

Il faut donc sécuriser ses conteneurs web (Apache / Nginx) avec au moins ces paramètres :

Nginx :

```
...
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
server {
    listen 443 ssl http2 default_server;
    listen [::]:443 ssl http2 default_server;
    ...
    add_header  Strict-Transport-Security  "max-age=31536000;  includeSubDomains"
always;
    add_header X-Frame-Options "sameorigin";
    add_header X-Content-Type-Options nosniff;
    add_header Content-Security-Policy "default-src 'self';";
    add_header Referrer-Policy "origin";
    add_header           Permissions-Policy           "geolocation=(),midi=(),sync-
xhr=(),microphone=(),camera=(),magnetometer=(),gyroscope=(),fullscreen=(self),paym
ent=()"
...
    location / {
        ...
        proxy_pass http://10.11.12.13:56789;
}
...
```

Apache :

```
server {
    ...
    Header always set X-Frame-Options "sameorigin"...
    ...
}
```

3. Fail2ban

Cet outil permet de bannir les tentatives d'authentifications infructueuses pour divers protocoles (HTTPS, SSH...). En option, des alertes peuvent être envoyées par mail.

- https://www.fail2ban.org

```
[docker@Rocky ~]$  dnf install -y fail2ban
[docker@Rocky      ~]$              cp       /etc/fail2ban/jail.conf
/etc/fail2ban/jail.local
[docker@Rocky ~]$  systemctl start fail2ban
```

Exemple de configuration :

```
[docker@Rocky ~]$  vi /etc/fail2ban/jail.local
```

```
[sshd]
enabled = true
port    = ssh
logpath = %(sshd_log)s
backend = systemd
maxretry = 5
bantime = 1d

[proxmox]
enabled = true
port = https,http,8006
filter = proxmox
backend = systemd
maxretry = 3
findtime = 7d
bantime = 3h
```

```
[docker@Rocky ~]$  fail2ban-client reload

[docker@Rocky ~]$  fail2ban-client status
Status
|- Number of jail:      2
`- Jail list:   proxmox, sshd

[docker@Rocky ~]$  fail2ban-client get sshd banned -i
367
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
[docker@Rocky ~]$  fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 1
|  |- Total failed:     3890
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 23
   |- Total banned:     367
   `- Banned IP list:   103.55.36.134 50.236.203.254 ...

[docker@Rocky ~]$  fail2ban-client banned
[{'sshd': ['103.55.36.134', '50.236.203.254', ...]}, {'proxmox': []}]

[docker@Rocky ~]$  fail2ban-client set proxmox unbanip 192.168.1.71
```

4. Crowdsec

Cet outil permet aussi de se protéger des multiples tentatives de connexion sur vos environnements. Il dispose d'une interface web qui permet d'analyser les blocages d'adresses IP sur plusieurs protocoles (SSH, HTTPS…).

Site : https://www.crowdsec.net/
Conteneur : https://hub.docker.com/r/crowdsecurity/crowdsec

Par défaut, Croudsec n'active pas de filtrage ; il faut installer des « bounces ». Par exemple, le filtrage SSH peut s'installer ce cette manière :

```
root@Debian:~ # VERSION=v0.0.31
root@Debian:~  #  wget  https://github.com/crowdsecurity/cs-firewall-bouncer/releases/download/$VERSION/crowdsec-firewall-bouncer-linux-amd64.tgz
root@Debian:~ # tar xzf crowdsec-firewall-bouncer.tgz
root@Debian:~ # cd crowdsec-firewall-bouncer-$VERSION
root@Debian:~ # ./install.sh
root@Debian:~ # systemctl status crowdsec-firewall-bouncer
root@Debian:~ # cscli bouncers list
root@Debian:~ # cscli scenarios list
root@Debian:~ # cscli alert list
root@Debian:~ # cscli collections list
root@Debian:~ # cscli decisions list
root@Debian:~ # cscli parsers list
```

Mise en place de l'interface web :

```
root@Debian:~ # cscli dashboard setup --listen 0.0.0.0
root@Debian:~ # cscli dashboard start
```

Autoriser une IP :

```
root@Debian:~ # cscli decisions delete -i <your ip>
```

Bannir des IP :

```
root@Debian:~ # cscli decisions add --ip 172.69.32.87 --duration 24h
root@Debian:~ # cscli decesions add --range 172.69.0.0/16
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XVIII. 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 -t <container>
```

Ports mappés

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

Evènements

```
[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
[docker@Rocky ~]$  docker events --filter container=netdata --since '2026-07-30T13:00:00' --until '2026-07-30T15:00:00'
```

https://docs.docker.com/reference/cli/docker/system/events/

Etat des process

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

Journaux

```
[docker@Rocky ~]$  docker logs -t --tail 30 -f --since 12h <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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

- .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
```

Nettoyage des images, volumes… :

```
[docker@Rocky ~]$ docker system prune --filter "until=24h"
```



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

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

B. Outils GUI

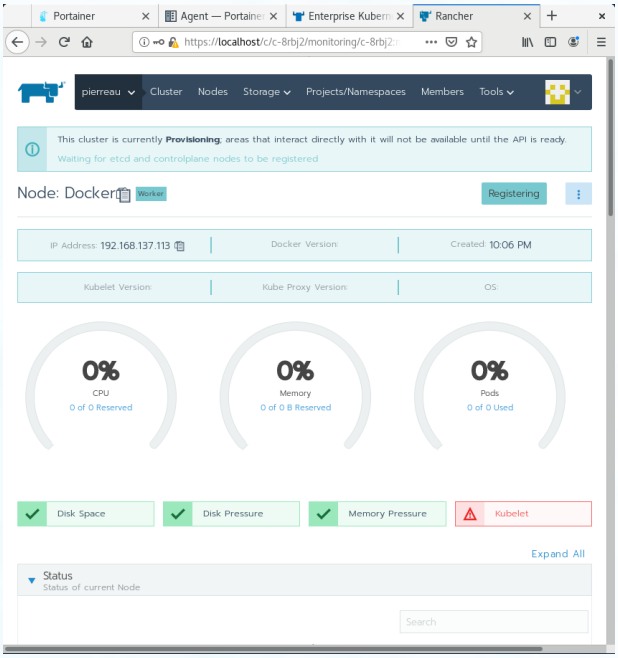
En plus de Portainer, d'autres environnements graphiques peuvent être installés afin de contrôler l'environnement des conteneurs :

1. Rancher

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

http://rancher.com/

Copyright © 2026, Pierre ROYER - https://pierreau.fr



2. OpenNebula

https://opennebula.io/

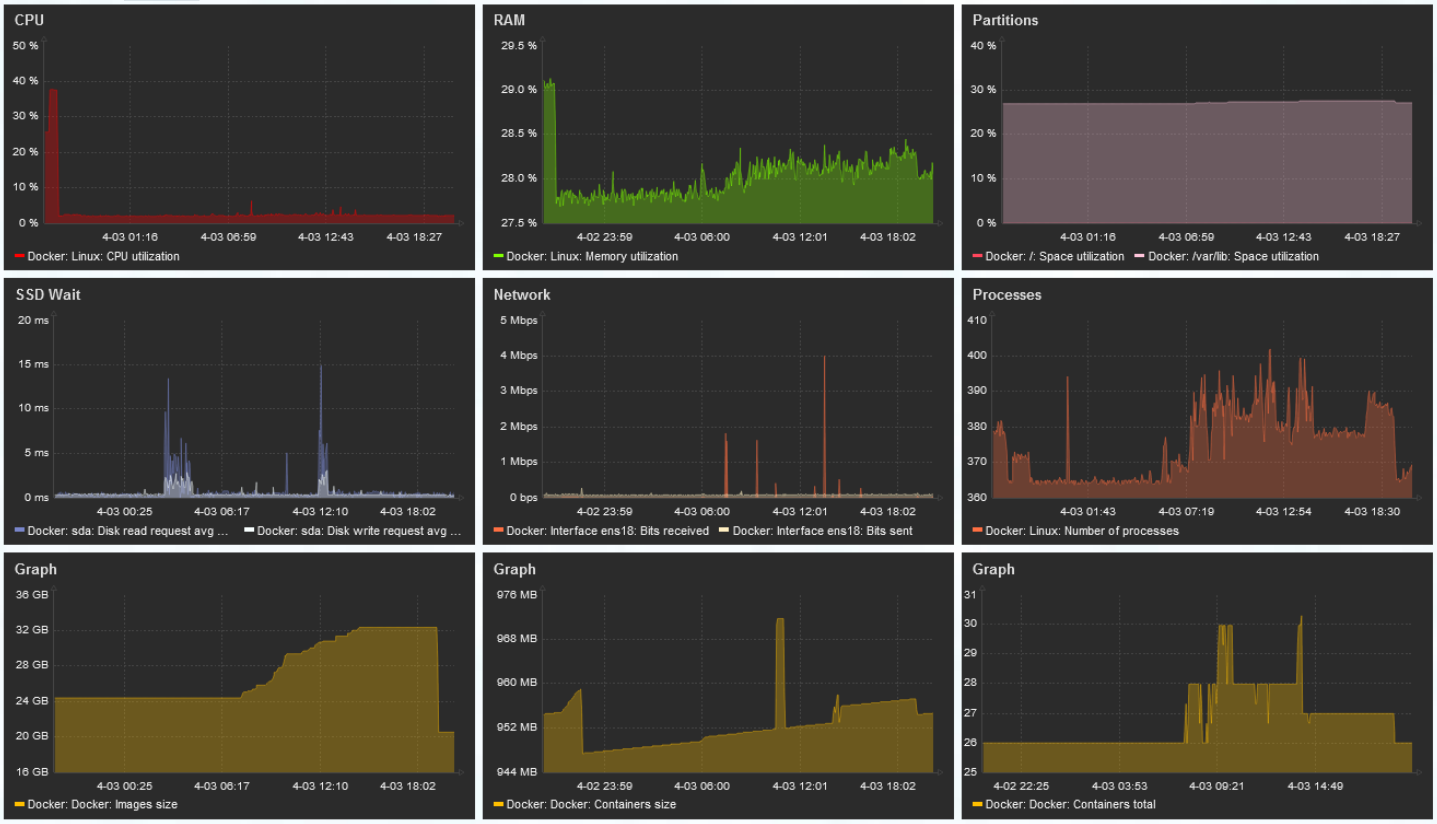
Gère les clusters avec Kubernetes

Copyright © 2026, Pierre ROYER - https://pierreau.fr

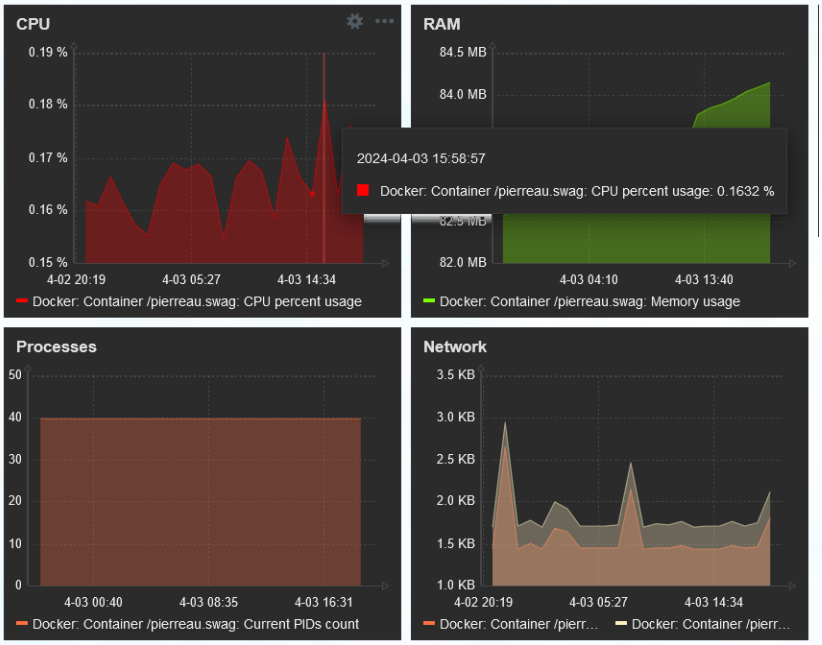
C. Zabbix

Bien qu'il soit compliqué de superviser des conteneurs qui peuvent se créer et s'arrêter à la volée, et en-dehors des orchestrateurs, des logiciels comme Zabbix est capable de superviser des conteneurs (avec l'agent2).

Supervision de l'état de Docker



Supervision d'un conteneur



Zabbix peut aussi s'installer avec docker compose. Pour vous former sur Zabbix, vous pouvez me consulter.

Copyright © 2026, Pierre ROYER - https://pierreau.fr

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) :

- 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 des normes ISO/CIE 27001-27005
- Savoir gérer les risques cyber (ISO 27032)

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

- Adopter les bonnes pratiques du site Cyber malveillance :

Exemples :
- Evaluer le niveau de cybersécurité de votre site Internet
- Le modèle « Zero Trust »

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XX. EXERCICES

Chapitre optionnel (selon la durée de la formation et du syllabus)

Dans la continuité de l'automatisation de notre serveur web personnel, appuyez-vous sur le Dockerfile généré précédemment, en y appliquant les bonnes pratiques et les exigences DevSecOps / RGPD, avec ces trois principes :

A. Security by default

Article 32 du GRPD

1. SSH

- Créer un conteneur Apache
- Créer un utilisateur « docker » (compte de service dédié aux connexions SSH) dans ce conteneur
- Lui affecter un mot de passe
- Installer et activer le service SSH dans le conteneur
- Vérifier la connexion SSH depuis le Host : **ssh -4 docker@*172.17.0.x***
- Générer une paire de clé avec l'algorithme ECDSA :

```
[docker@Rocky ~]$  ssh-keygen -t ecdsa -b 521
[docker@Rocky ~]$  ssh-copy-id -i ~/.ssh/id_ecdsa.pub docker@172.17.0.x
[docker@Rocky ~]$  ssh docker@172.17.0.x
```

- Transférer la clé publique dans le conteneur
- Valider la connexion SSH sans mot de passe depuis l'hôte



Le choix personnel de l'algorithme ECDSA (Elliptic Curve Cryptography) par rapport au plus commun RSA tient de ces faits :

- Une signature ECDSA sur 256 bits est aussi robuste que celle de RSA sur 3072 bits
- ECDSA est plus rapide à générer, et demande mois de mémoire vive
- ECDSA a été introduit dans la spécification de TLS v1.2 en 2008, alors que RSA a été normalisé pour SSL / TLS en 1994

2. SSL

Avec les packages **openssl mod_ssl** ,

Générer un certificat SSL et l'inclure dans Apache

Exemple à réadapter :

```
[root@container ~]# cd /etc/pki/tls/certs/

[root@container ~]# openssl req -x509 -days 365 -nodes -newkey ec -pkeyopt
ec_paramgen_curve:prime256v1 -keyout pierreau.fr.key -out pierreau.fr.crt

Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:Pays de la Loire
```

Copyright © 2026, Pierre ROYER - https://pierreau.fr

```
Locality Name (eg, city) [Default City]:Nantes
Organization Name (eg, company) [Default Company Ltd]:Pierreau
Organizational Unit Name (eg, section) []:DSI
Common Name (eg, your name or your server's hostname) []:pierreau.fr
Email Address []:noreply@pierreau.fr
```

```
[root@container ~]# vi /etc/httpd/conf.d/ssl.conf
```

```
Listen 443 https
<VirtualHost _default_:443>
...
   ServerName pierreau.fr

   SSLEngine On

# Désactivation des protocoles obsolètes
   SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1

   SSLCertificateFile /etc/pki/tls/certs/pierreau.fr.crt
   SSLCertificateKeyFile /etc/pki/tls/certs/pierreau.fr.key
...
</VirtualHost>
```

```
[root@container ~]# systemctl reload httpd
```

Tester la connexion HTTPS (publier le port 443 dans le conteneur).

B. Privacy by design

Article 25 du RGPD

L'objectif est de créer une habilitation basique aux ressources du conteneur, via un login / mot de passe. Cet accès est géré par le serveur Apache
A l'aide de la commande **htpasswd** et d'un fichier `.htaccess` , protéger l'accès au site avec un login / mot de passe.

C. Accountability

Le but étant de démontrer le respect des règles et processus relatifs à la protection des données (responsabilisation / DPO). Afin de démontrer quoi que ce soit, il nous faut des preuves, ou une traçabilité…

- Pérenniser les logs de Apache dans un volume (ou un fichier)
- Dans un autre conteneur, installer https://goaccess.io/
- Les données de GoAccess proviennent du volume généré par le conteneur Apache. Ce qui nous garantit une traçabilité sur le long terme, hors du cycle de vie des images…

Copyright © 2026, Pierre ROYER - https://pierreau.fr

XXI. Annexe

A. Outils de conteneurisation

1. En production

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/

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…

k3s

Développé par Rancher. Support possible.

Microk8s

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

Copyright © 2026, Pierre ROYER - https://pierreau.fr

2. Hors production

Docker Desktop

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

Minikube



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

Copyright © 2026, Pierre ROYER - https://pierreau.fr