Docker - introduction

jeudi 26 janvier 2023 14:26

Formateur : Loup FORMENT

Type 1 : natif [modifier | modifier le code]

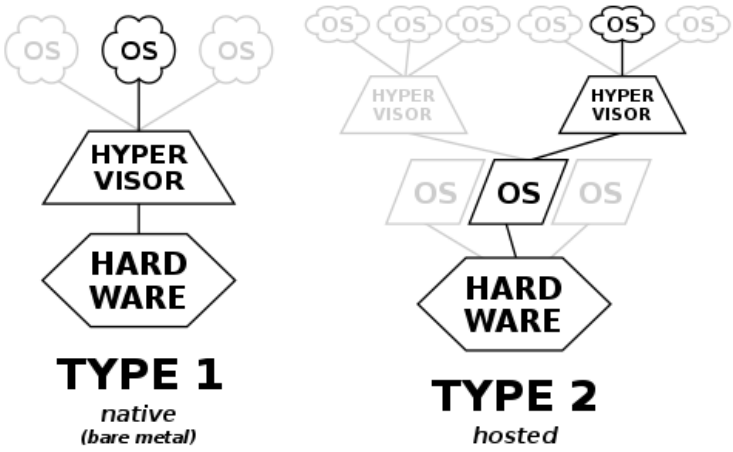
Un hyperviseur de Type 1, natif, voire « *bare metal* » (littéralement « métal nu »), est un logiciel qui s'exécute directement sur une plateforme matérielle ; cette plateforme est alors considérée comme *outil de contrôle* de système d'exploitation. Un système d'exploitation secondaire peut, de ce fait, être exécuté au-dessus du matériel.
L'hyperviseur type 1 est un noyau hôte allégé et optimisé. Sur des processeurs ayant les instructions de virtualisation matérielle (AMD-V et Intel VT) l'hyperviseur n'a plus à émuler les anneaux de protection et le fonctionnement s'en trouve accéléré.

Un hyperviseur de type 1 classique est CP, développé par IBM dans les années 60 et ancêtre de z/VM. Des exemples d'hyperviseurs plus récents sont Xen, Oracle VM, Microsoft Hyper-V, ESXI Server de VMware, l'hyperviseur LPAR de IBM (PR/SM), PolyXene [archive] de Bertin IT [archive] , l'hyperviseur Logical Domains de SUN (sorti en 2005)... Une légère variation consiste à intégrer l'hyperviseur dans le micrologiciel (firmware) de la plateforme. C'est ce qui a été fait dans le cas de l'hyperviseur Virtage d'Hitachi. Les machines virtuelles utilisant un noyau Linux KVM, qui transforment un noyau Linux complet en hyperviseur, sont également considérées comme hyperviseurs de type 1.

Type 2 : hosted [modifier | modifier le code]

Un hyperviseur de Type 2 est un logiciel qui s'exécute à l'intérieur d'un autre système d'exploitation. Un système d'exploitation invité s'exécutera donc en troisième niveau au-dessus du matériel. Les systèmes d'exploitation invités n'ayant pas conscience d'être virtualisés, ils n'ont pas besoin d'être adaptés. Quelques exemples de tels hyperviseurs sont VMware Workstation, VMware Fusion, l'hyperviseur open source QEMU, Virtual PC, Virtual Server, VirtualBox d'Oracle, de même que Parallels Workstation de SWsoft et Parallels Desktop.

Le terme hyperviseur prend sa source dans la réimplémentation par IBM de CP-67 pour le système d'exploitation System/370 sorti en 1972 sous le nom VM/370. Le terme appel hyperviseur ou hypervisor call, ou hypercall, fait référence à l'interface de paravirtualisation, par laquelle un système d'exploitation « invité » accède directement à des services à travers le logiciel de contrôle de niveau élevé (le terme *superviseur* fait référence au noyau du système d'exploitation qui sur les mainframes IBM s'exécute en *mode Superviseur*).



Logiciel de virtualisation sous LInux -> **Qemu**

QEMU

Article Discussion

QEMU est un logiciel libre de machine virtuelle, pouvant émuler un processeur et, plus généralement, une architecture différente si besoin. Il permet d'exécuter un ou plusieurs systèmes d'exploitation via les hyperviseurs KVM et Xen, ou seulement des binaires, dans l'environnement d'un système d'exploitation déjà installé sur la machine.

Exemple d'exécution de VM sous **QEMU**

```
:: Exécution des crochets (« hooks ») de p
ost-transaction…
(1/7) Creating system user accounts...
```

```
Creating group 'qemu' with GID 968.
Creating user 'qemu' (QEMU user) with UID
968 and GID 968.
(2/7) Reloading system manager configurati
on...
(3/7) Creating temporary files...
(4/7) Reloading device manager configurati
on...
(5/7) Arming ConditionNeedsUpdate...
(6/7) Updating icon theme caches...
(7/7) Updating the desktop file MIME type
cache...
[darksasuke@chocolat python_cgi ]$ cd
[darksasuke@chocolat ~ ]$ qemu-img create
-f raw image_file 8G
Formatting 'image_file', fmt=raw size=8589
934592
[darksasuke@chocolat ~ ]$ ls image_file
image_file
[darksasuke@chocolat ~ ]$ qemu-system-x86_
64 -cdrom Téléchargements/archlinux-2023.0
1.01-x86_64.iso -boot order=d -drive file=
image_file,format=raw
VNC server running on ::1:5900
```

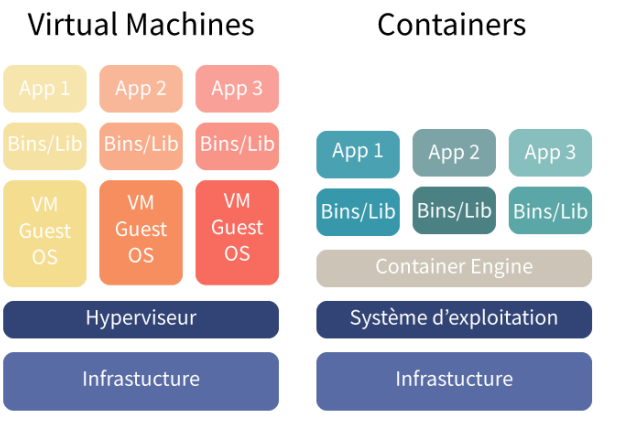
Docker (logiciel)

Article Discussion

Pour les articles homonymes, voir Docker (homonymie).

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

Selon la firme de recherche sur l'industrie 451 Research, « Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc.[2]



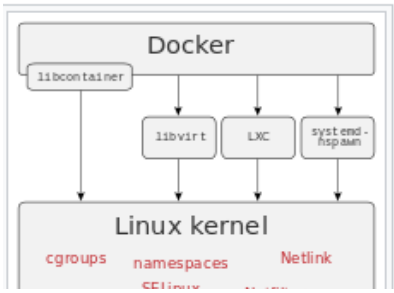


Schéma des interfaces de Docker

"Une autre différence importante avec les machines virtuelles est qu'un conteneur **ne réserve pas**
la quantité de CPU, RAM et disque attribuée auprès du système hôte.
Ainsi, nous pouvons allouer 16 Go de RAM à notre conteneur, mais si
celui-ci n'utilise que 2 Go, le reste ne sera pas verrouillé."

https://www.docker.com/

https://wiki.archlinux.org/title/docker

https://doc.ubuntu-fr.org/docker

https://docs.docker.com/

Installation **Docker**

```
vel@vel-Precision-3650-Tower:~$ sudo apt install docker.io
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  fonts-dejavu gir1.2-keybinder-3.0 kded5 keditbookmarks kio konsole-kpart kpackagelauncherqml
  kpackagetool5 libflashrom1 libftdi1-2 libhfstospell11 libkeybinder-3.0-0 libkf5archive5
  libkf5attica5 libkf5bookmarks-data libkf5bookmarks5 libkf5completion-data libkf5completion5
  libkf5crash5 libkf5declarative-data libkf5declarative5 libkf5doctools5 libkf5globalaccel-bin
  libkf5globalaccel-data libkf5globalaccel5 libkf5globalaccelprivate5 libkf5iconthemes-bin
  libkf5iconthemes-data libkf5iconthemes5 libkf5itemviews-data libkf5itemviews5
  libkf5jobwidgets-data libkf5jobwidgets5 libkf5kiocore5 libkf5kiogui5 libkf5kiontlm5
  libkf5kiowidgets5 libkf5kirigami2-5 libkf5newstuff-data libkf5newstuff5 libkf5newstuffcore5
  libkf5notifyconfig-data libkf5notifyconfig5 libkf5package-data libkf5package5 libkf5parts-data
  libkf5parts-plugins libkf5parts5 libkf5pty-data libkf5pty5 libkf5quickaddons5 libkf5solid5
  libkf5solid5-data libkf5sonnet5-data libkf5sonnetcore5 libkf5sonnetui5 libkf5syndication5abi1
```

```
vel@vel-Precision-3650-Tower:~$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/home/vel/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides
                           DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal")
                           (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/home/vel/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/home/vel/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/home/vel/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
```

```
vel@vel-Precision-3650-Tower:~$ systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-01-26 14:52:26 CET; 5min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 27112 (dockerd)
      Tasks: 21
     Memory: 31.0M
        CPU: 329ms
     CGroup: /system.slice/docker.service
             └─27112 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.526927628+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.526938865+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.526946590+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.617969996+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.824493820+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.890768097+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.955684546+01:00">
janv. 26 14:52:25 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:25.955882052+01:00">
janv. 26 14:52:26 vel-Precision-3650-Tower systemd[1]: Started Docker Application Container Engine.
janv. 26 14:52:26 vel-Precision-3650-Tower dockerd[27112]: time="2023-01-26T14:52:26.019741895+01:00">
lines 1-22/22 (END)
```

https://hub.docker.com/search?q=

Pour récupérer une image sur notre système -> **docker pull**


```
vel@vel-Precision-3650-Tower:~$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8740c948ffd4: Pull complete
d2c0556a17c5: Pull complete
c8b9881f2c6a: Pull complete
693c3ffa8f43: Pull complete
8316c5e80e6d: Pull complete
b2fe3577faa4: Pull complete
Digest: sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Pour voir toute nos images : **sudo docker images**

```
vel@vel-Precision-3650-Tower:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    a99a39d070bf   2 weeks ago   142MB
```

```
vel@vel-Precision-3650-Tower:~$ sudo docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
8921db27df28: Pull complete
Digest: sha256:f271e74b17ced29b915d351685fd4644785c6d1559dd1f2d4189a5e851ef753a
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
vel@vel-Precision-3650-Tower:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    a99a39d070bf   2 weeks ago   142MB
alpine       latest    042a816809aa   2 weeks ago   7.05MB
```

Démarrer un conteneur -> **docker run**

```
vel@vel-Precision-3650-Tower:~$ sudo docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/01/26 14:08:22 [notice] 1#1: using the "epoll" event method
2023/01/26 14:08:22 [notice] 1#1: nginx/1.23.3
2023/01/26 14:08:22 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/01/26 14:08:22 [notice] 1#1: OS: Linux 5.15.0-57-generic
2023/01/26 14:08:22 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/01/26 14:08:22 [notice] 1#1: start worker processes
2023/01/26 14:08:22 [notice] 1#1: start worker process 29
2023/01/26 14:08:22 [notice] 1#1: start worker process 30
2023/01/26 14:08:22 [notice] 1#1: start worker process 31
2023/01/26 14:08:22 [notice] 1#1: start worker process 32
2023/01/26 14:08:22 [notice] 1#1: start worker process 33
2023/01/26 14:08:22 [notice] 1#1: start worker process 34
2023/01/26 14:08:22 [notice] 1#1: start worker process 35
2023/01/26 14:08:22 [notice] 1#1: start worker process 36
2023/01/26 14:08:22 [notice] 1#1: start worker process 37
2023/01/26 14:08:22 [notice] 1#1: start worker process 38
2023/01/26 14:08:22 [notice] 1#1: start worker process 39
2023/01/26 14:08:22 [notice] 1#1: start worker process 40
2023/01/26 14:08:22 [notice] 1#1: start worker process 41
2023/01/26 14:08:22 [notice] 1#1: start worker process 42
2023/01/26 14:08:22 [notice] 1#1: start worker process 43
2023/01/26 14:08:22 [notice] 1#1: start worker process 44
```

Voir les conteneurs lancé : **sudo docker ps**

```
vel@vel-Precision-3650-Tower:~$ sudo docker ps
[sudo] Mot de passe de vel :
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
5329040a50bc   nginx     "/docker-entrypoint.…"   58 seconds ago   Up 57 seconds   80/tcp    nostalgic
```

```
_wright
vel@vel-Precision-3650-Tower:~$
```

Stop conteneur : **docker stop "nom du conteneur"**

```
vel@vel-Precision-3650-Tower:~$ sudo docker stop nostalgic_wright
nostalgic_wright
vel@vel-Precision-3650-Tower:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Pour voir les conteneurs stoppé : **docker ps -a**

```
vel@vel-Precision-3650-Tower:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                     PORTS
 NAMES
5329040a50bc   nginx     "/docker-entrypoint.…"   5 minutes ago   Exited (0) 3 seconds ago
 nostalgic_wright
```

Pour relancer conteneur stoppé : **docker restart "nom du conteneur"**

```
vel@vel-Precision-3650-Tower:~$ sudo docker restart nostalgic_wright
nostalgic_wright
vel@vel-Precision-3650-Tower:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
5329040a50bc   nginx     "/docker-entrypoint.…"   4 minutes ago   Up 3 seconds   80/tcp    nostalgic_w
right
```

Si on a déjà lancé le conteneur, il faudra utiliser la commande **docker start "nom du container'** pour le redémarrer, pas **run**

Pour supprimer un container : **docker rm "nom du container"**

```
vel@vel-Precision-3650-Tower:~$ sudo docker rm nostalgic_wright
nostalgic_wright
vel@vel-Precision-3650-Tower:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAME
vel@vel-Precision-3650-Tower:~$
```

Une fois les container supprimé, il faut aussi supprimer les images si nous n'en avons plus besoin : **sudo docker rmi**

```
vel@vel-Precision-3650-Tower:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    a99a39d070bf   2 weeks ago   142MB
alpine       latest    042a816809aa   2 weeks ago   7.05MB
vel@vel-Precision-3650-Tower:~$ sudo docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Deleted: sha256:a99a39d070bfd1cb60fe65c45dea3a33764dc00a9546bf8dc46cb5a11b1b50e9
Deleted: sha256:937f740376014d8e951ae4f0ff0b0ca64fd958c176510fc0a86dd1b491226a71
Deleted: sha256:a73369e624d6b4fbea1e6046a2f3cf3c7a15c6e0e0012659f410380d8dc70594
Deleted: sha256:bc2dd4cf50f065c3e884872cd30eb888a71f6ab545d8bec196a7662bfbd033a3
Deleted: sha256:c9372c5c9249380700f7af0441de14f14c361b5561ecc876da46f58a7ad71b7b
Deleted: sha256:deea3af05bc845b217a415b240422c19933218422b6d60414c2caeff976e5430
Deleted: sha256:67a4178b7d47beb6a1f697a593bd0c6841c67eb0da00f2badefb05fd30671490
vel@vel-Precision-3650-Tower:~$ sudo docker rmi alpine
Untagged: alpine:latest
Untagged: alpine@sha256:f271e74b17ced29b915d351685fd4644785c6d1559dd1f2d4189a5e851ef753a
Deleted: sha256:042a816809aac8d0f7d7cacac7965782ee2ecac3f21bcf9f24b1de1a7387b769
Deleted: sha256:8e012198eea15b2554b07014081c85fec4967a1b9cc4b65bd9a4bce3ae1c0c88
vel@vel-Precision-3650-Tower:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

Le **pull de l'image** n'est pas forcément obligatoire :

```
[darksasuke@chocolat ~ ]$ sudo docker run nginx -d
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8740c948ffd4: Pull complete
d2c0556a17c5: Pull complete
c8b9881f2c6a: Pull complete
693c3ffa8f43: Pull complete
8316c5e80e6d: Pull complete
b2fe3577faa4: Pull complete
Digest: sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: 47: exec: -d: not found
[darksasuke@chocolat ~ ]$ sudo docker run -d nginx
b8efe53352255d7fb93ca6b83134d78fc1623d87a0e971aaccc12bfe92872c5a
[darksasuke@chocolat ~ ]$
```

Un **docker run nginx** va allez télécharger l'image et la déployer

```
[darksasuke@chocolat ~ ]$ sudo docker stop $(sudo docker ps -q)
```

```
[darksasuke@chocolat ~ ]$ sudo docker rm -f $(sudo docker ps -
```

Voir les différents network de docker : **sudo docker network ls**

```
vel@vel-Precision-3650-Tower:~$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
21935ce3e38f   bridge    bridge    local
b367b8af042e   host      host      local
83003606673b   none      null      local
```

Voir les informations sur le réseau : **sudo docker network inspect "nom du network"**

```
vel@vel-Precision-3650-Tower:~$ sudo docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "21935ce3e38fc1ec5f54eac6abc7c74d28ed37e4b53bd00ed95be9918a5da6fa",
        "Created": "2023-01-26T14:52:25.824513082+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "b8d0277f508843e3fdfc2a7f401a343789a3941c497b0ec52b0560e94cbf7948": {
                "Name": "pensive_chebyshev",
                "EndpointID": "f7190636a75365784d932c891537b7919e044f3e790c44fcc4d6b74a5baa615b",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Cette commande afficher le fichier JSON du network, ici le bridge -> On y retrouve l'ip de notre container

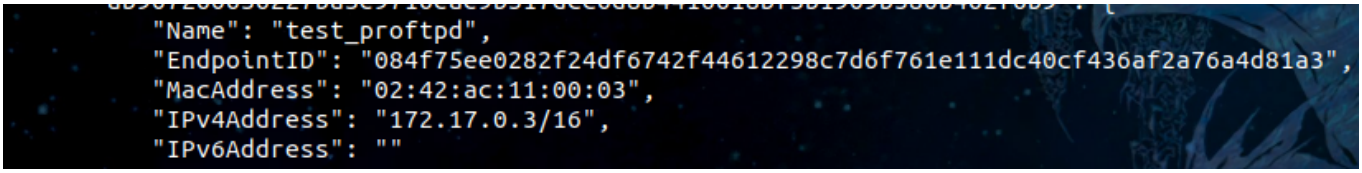
Voir les logs concernant un container : **sudo docker logs "nom du container"**

```
vel@vel-Precision-3650-Tower:~$ sudo docker logs cranky_noyce
** This container will not run without setting for PASV_ADDRESS **
** This container will not run without setting for PASV_ADDRESS **
```

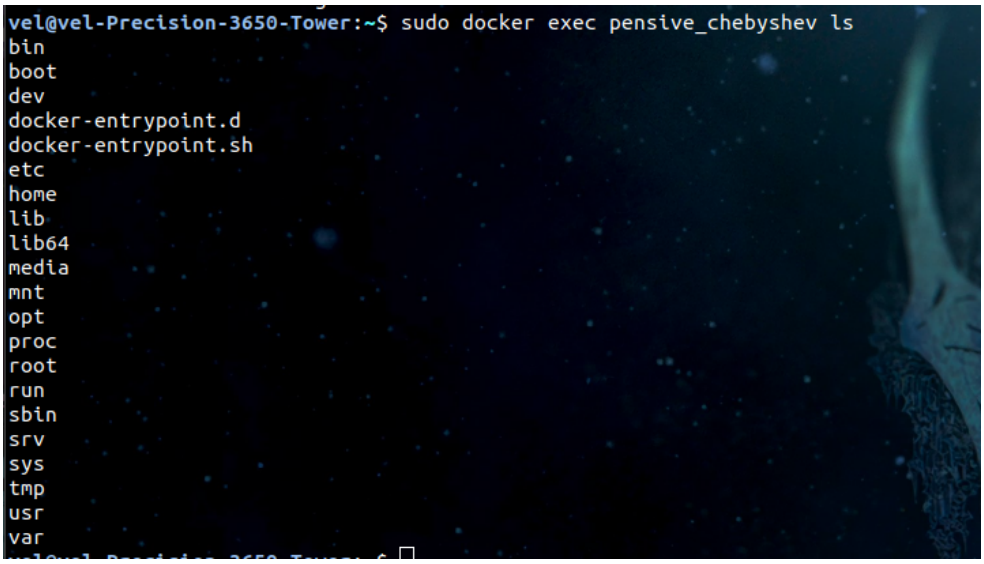
```
[darksasuke@chocolat ~ ]$ sudo docker run -d -e "PASV_ADDRESS=172.17.0.3" --name ftpd i
ntlinux/proftpd
```

Pour forcer une adresse IP sur un container au moment de run : argument **–e "PASV_ADDRESS=xxxx.xxxx.xxxx.xxxx"**

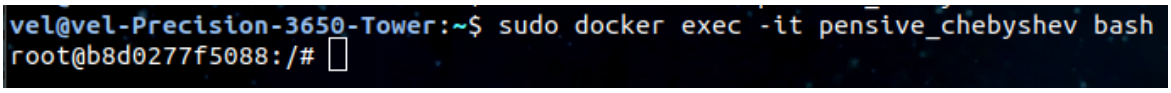
```
    },
    "ConfigOnly": false,
    "Containers": {
        "ab907260636227ba3e9716eae9b317dcc6d8b4416618bf3b1909b580b462f6b9": {
```



Pour exécuter une commande sur un container : **sudo container exec "nom_container" "nom_commande"**



Pour rentrer dans le container (chrooter dans le container) = **sudo docker exec –it "nom_conainter" "nom_commande"**

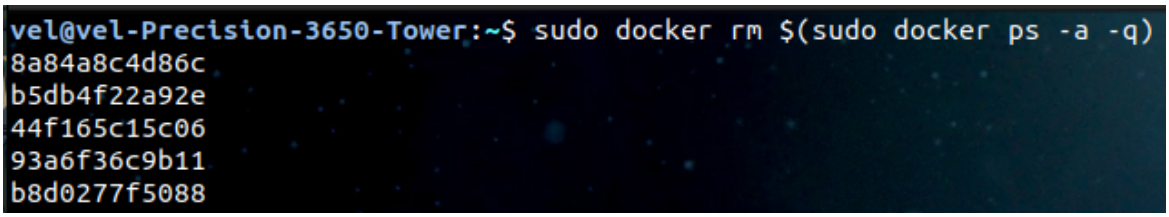


On rentre dans notre conteneur avec la commande bash active

Faire le ménage dans les conteneurs/images :

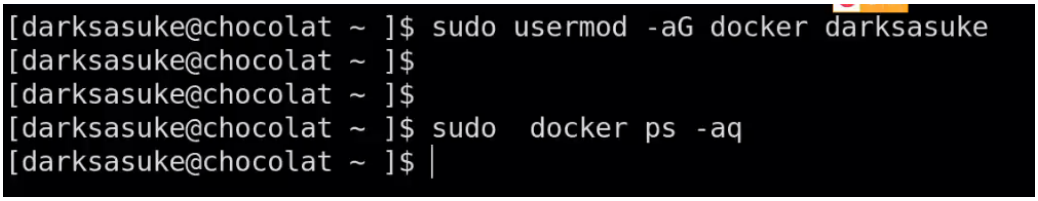
https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes-fr

Supprimer tous les conteneurs sur une infrastructure : **sudo docker rm $(sudo docker ps –a –q)**



Si notre utilisateur n'est pas dans le groupe docker, il faudra utiliser sudo pour exécuter chaque commande : nous pouvons rajouter notre utilisateur au groupe docker --> **sudo usermod –aG docker "user"**

NE PAS OUBLIER –aG : -G seul supprime tout les groupes secondaire de l'utilisateur



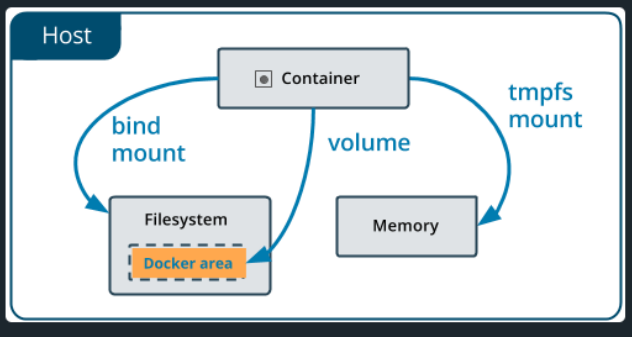
Volume Docker

https://docs.docker.com/storage/volumes/

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container.
- Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

In addition, volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.



Choose the -v or --mount flag

In general, `--mount` is more explicit and verbose. The biggest difference is that the `-v` syntax combines all the options together in one field, while the `--mount` syntax separates them. Here is a comparison of the syntax for each flag.

If you need to specify volume driver options, you must use `--mount`.

- `-v` **or** `--volume`: Consists of three fields, separated by colon characters (`:`). The fields must be in the correct order, and the meaning of each field is not immediately obvious.
  - In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine. For anonymous volumes, the first field is omitted.
  - The second field is the path where the file or directory are mounted in the container.
  - The third field is optional, and is a comma-separated list of options, such as `ro`. These options are discussed below.
- `--mount`: Consists of multiple key-value pairs, separated by commas and each consisting of a `<key>=<value>` tuple. The `--mount` syntax is more verbose than `-v` or `--volume`, but the order of the keys is not significant, and the value of the flag is easier to understand.
  - The `type` of the mount, which can be `bind`, `volume`, or `tmpfs`. This topic discusses volumes, so the type is always `volume`.
  - The `source` of the mount. For named volumes, this is the name of the volume. For anonymous volumes, this field is omitted. May be specified as `source` or `src`.
  - The `destination` takes as its value the path where the file or directory is mounted in the container. May be specified as `destination`, `dst`, or `target`.
  - The `readonly` option, if present, causes the bind mount to be mounted into the container as read-only. May be specified as `readonly` or `ro`.
  - The `volume-opt` option, which can be specified more than once, takes a key-value pair consisting of the option name and its value.

Create and manage volumes

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

Create a volume:

```
$ docker volume create my-vol
```

List volumes:

```
$ docker volume ls

local               my-vol
```

Inspect a volume:

```
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

Remove a volume:

```
$ docker volume rm my-vol
```

Start a container with a volume

If you start a container with a volume that doesn't yet exist, Docker creates the volume for you. The following example mounts the volume `myvol2` into `/app/` in the container.

The `-v` and `--mount` examples below produce the same result. You can't run them both unless you remove the `devtest` container and the `myvol2` volume after running the first one.

`--mount` `-v`

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2,target=/app \
  nginx:latest
```

Use `docker inspect devtest` to verify that the volume was created and mounted correctly. Look for the `Mounts` section:

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "myvol2",
        "Source": "/var/lib/docker/volumes/myvol2/_data",
        "Destination": "/app",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

```
[darksasuke@chocolat ~ ]$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          3         0         573.6MB   573.6MB (100
%)
Containers      0         0         0B        0B
Local Volumes   9         0         0B        0B
Build Cache     0         0         0B        0B
```

Dans le container : les fichier **docker-entrypoint** contiennent les variables d'environnement

```
root@0d3c31294cc6:/# ls
bin   dev                docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d  etc                   lib   media  opt  root  sbin  sys  usr
```

Montage du dossier nginx/html pour pouvoir modifier la page HTML depuis notre machine physique :

docker run –d –rm –v "chemin de notre dossier":"chemin du dossier de notre container"

```
[darksasuke@chocolat ~ ]$ docker run -d --rm -v "$(pwd)"/root_nginx:/app ng
inx
```

```
[darksasuke@chocolat ~ ]$ docker run -d --rm -v "$(pwd)"/root_nginx:/usr/sh
are/nginx/html --name test nginx
6068be7e29f65017d081f4303eb054c742bcb87038a735ab4511c1869f36de1a
[darksasuke@chocolat ~ ]$ dockerr ps
bash: dockerr : commande introuvable
[darksasuke@chocolat ~ ]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STAT
US              PORTS     NAMES
6068be7e29f6   nginx     "/docker-entrypoint.…"   5 seconds ago        Up 4
 seconds        80/tcp    test
68ddc729972c   nginx     "/docker-entrypoint.…"   About a minute ago   Up A
bout a minute   80/tcp    boring_kowalevski
2f9636204e51   nginx     "/docker-entrypoint.…"   2 minutes ago        Up 2
 minutes        80/tcp    dreamy_sinoussi
adb2541d5bad   nginx     "/docker-entrypoint.…"   8 minutes ago        Up 8
 minutes        80/tcp    sweet_cori
```

```
  554  docker run -d --rm -v "$(pwd)"/root_nginx:/usr/share/nginx/html --na
me test nginx
  555  dockerr ps
  556  docker ps
  557  docker exec -it 6068be7e29f6 bash
  558  docker network inspect bridge
  559  history
  560  vim root_nginx/index.html
```

Pour créer un container avec un dossier déjà monté : **vol:chemin absolue du dossier**

```
[darksasuke@chocolat ~ ]$ docker run -d --rm -v vol:/usr/share/nginx/html -
-name test2 nginx
afd7be3145d6ea77a5f94676f2aad600225728994aadf9a3b84c61d651ee1cfb
[darksasuke@chocolat ~ ]$
```

```
[darksasuke@chocolat ~ ]$ docker run -d --rm -v vol:/usr/share/nginx/html -
-name test2 nginx
afd7be3145d6ea77a5f94676f2aad600225728994aadf9a3b84c61d651ee1cfb
[darksasuke@chocolat ~ ]$ docker run -d --rm -v vol:/usr/share/nginx/html -
-name test3 nginx
```

How to use this image

Hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Alternatively, a simple `Dockerfile` can be used to generate a new image that includes the necessary content (which is a much cleaner solution than the bind mount above):

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

Place this file in the same directory as your directory of content ("static-html-directory"), run `docker build -t some-content-nginx .`, then start your container:

```
$ docker run --name some-nginx -d some-content-nginx
```

Docker Network

https://docs.docker.com/network/

Network drivers

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- `bridge` : The default network driver. If you don't specify a driver, this is the type of network you are creating. **Bridge networks are usually used when your applications run in standalone containers that need to communicate.** See bridge networks.
- `host` : For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. See use the host network.
- `overlay` : Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.
- `ipvlan` : IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration. See IPvlan networks.
- `macvlan` : Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the `macvlan` driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack. See Macvlan networks.
- `none` : For this container, disable all networking. Usually used in conjunction with a custom network driver. `none` is not available for swarm services. See disable container networking.
- Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors. See the vendor's documentation for installing and using a given network plugin.

Network driver summary

- **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
- **Host networks** are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
- **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- **Macvlan networks** are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
- **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

Création nouveau network :

```
585  docker network create my_net
586  docker network ls
587  docker network inspect my_net
```

Par défaut, un nouveau network est lié au driver du bridge. Si le besoin est autre, il est possible de créer d'autres type de network lié à d'autre types de driver bridges

```
[darksasuke@chocolat ~ ]$ docker run -d --rm --network my_net --name t
est nginx
bcc352981158941e1a2c404f258d4c8e3c7cf02a9fba1e9572691e13b0ac563b
```

Quand on vérifie dans notre network, on voit les deux containers dans sa configuration

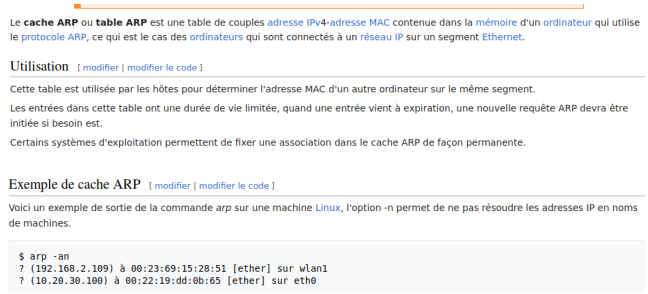
```
                "Name": "test2",
                "EndpointID": "daa3595ad77b6c68275c5f3595bb943c65b14ca
db9798d8408b833b5d6af489f",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
```

```
            "bcc352981158941e1a2c404f258d4c8e3c7cf02a9fba1e9572691e13b
0ac563b": {
                "Name": "test",
                "EndpointID": "7a63141408593b62429d67a3023d412484cf7bd
e1f55cc36923c7afb055c7451",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
```

Test de ping entre les deux container depuis test

```
root@bcc352981158:/# ping 172.18.0.3
PING 172.18.0.3 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: icmp_seq=0 ttl=64 time=0.084 ms
64 bytes from 172.18.0.3: icmp_seq=1 ttl=64 time=0.043 ms
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.051 ms
```

Exemple de table ARP :



Le **cache ARP** ou **table ARP** est une table de couples adresse IPv4-adresse MAC contenue dans la mémoire d'un ordinateur qui utilise le protocole ARP, ce qui est le cas des ordinateurs qui sont connectés à un réseau IP sur un segment Ethernet.

Utilisation [modifier | modifier le code]

Cette table est utilisée par les hôtes pour déterminer l'adresse MAC d'un autre ordinateur sur le même segment.

Les entrées dans cette table ont une durée de vie limitée, quand une entrée vient à expiration, une nouvelle requête ARP devra être initiée si besoin est.

Certains systèmes d'exploitation permettent de fixer une association dans le cache ARP de façon permanente.

Exemple de cache ARP [modifier | modifier le code]

Voici un exemple de sortie de la commande *arp* sur une machine Linux, l'option -n permet de ne pas résoudre les adresses IP en noms de machines.

```
$ arp -an
? (192.168.2.109) à 00:23:69:15:28:51 [ether] sur wlan1
? (10.20.30.100) à 00:22:19:dd:0b:65 [ether] sur eth0
```

```
root@bcc352981158:/# ip n
172.18.0.3 dev eth0 lladdr 02:42:ac:12:00:03 STALE
172.18.0.1 dev eth0 lladdr 02:42:75:50:30:87 STALE
```

Redirection de port :

Pour faire une redirection de port : **-p port_de_la_machine:port_du_docker**

```
[darksasuke@chocolat ~ ]$ docker run -d --rm --network my_net -p 8098:80 --name test2 nginx
```

```
[darksasuke@chocolat ~ ]$ sudo netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8098            0.0.0.0:*               LISTEN      9789/docker-proxy
tcp        0      0 10.125.25.52:40056      34.107.221.82:80        ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:41936      54.149.190.160:443      ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:40374      198.252.206.25:443      ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:47396      52.114.76.233:443       ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:40052      34.107.221.82:80        ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:44656      52.97.219.210:443       ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:36678      52.114.104.10:443       ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:50790      52.113.199.112:443      ESTABLISHED 3168/firefox
tcp        0      0 10.125.25.52:50692      52.113.205.19:443       ESTABLISHED 3168/firefox
tcp6       0      0 :::8098                 :::*                    LISTEN      9812/docker-proxy
```

```
[darksasuke@chocolat ~ ]$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
6b3586f73c18   bridge    bridge    local
316fe046d110   host      host      local
31fb71083d06   my_net    bridge    local
8a1ccde1a97a   none      null      local
```

```
[darksasuke@chocolat ~ ]$ docker network create my_net2 --sub.16.17.0/24
```

Nombre de docker possible dans ce sous-réseau -> 256 –2 (broadcast + routeur) = 254 adresse disponible

Pour définir gateway sur un container : **--gateway "adresse_gateway"**

```
[darksasuke@chocolat ~ ]$ docker network create my_net2 --subnet 15.16.17.0/24 --gate
way 15.16.17.1
4c82855e19eeb18c8dd21000212ca73036e8ca06d9c4d4763fc1ea8a93c75126
```

history commande network :

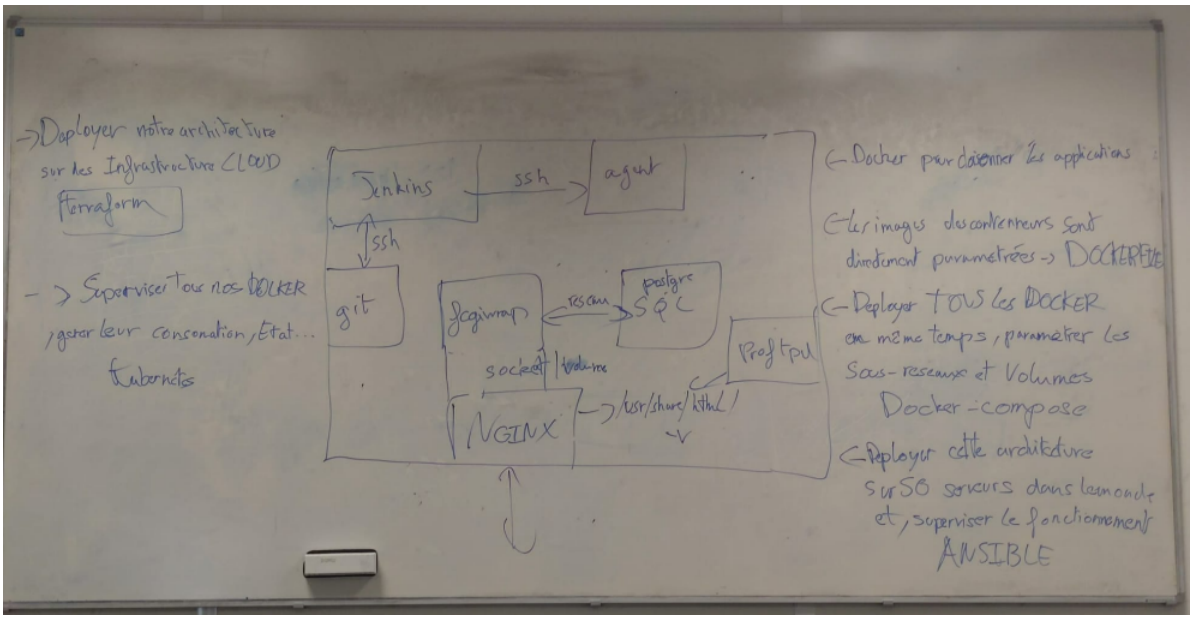
```
626  sudo docker network ls
627  sudo docker network rm app
628  sudo docker network ls
629  sudo docker network inspect my_net
630  docker run -d --rm --network my_net --ip 172.18.0.2 --name test1   nginx
631  docker run -d --rm --network my_net --ip 172.18.0.2 --name test1   nginx
632  sudo docker network inspect my_net
633  docker network create my_net2 --subnet 15.16.17.0/24
634  docker network inspect my_net2
635  docker network rm my_net2
636  docker network create my_net2 --subnet 15.16.17.0/24 --gateway 15.16.17.1/24
637  docker network create my_net2 --subnet 15.16.17.0/24 --gateway 15.16.17.1
638  docker network inspect my_net2
639  docker run -d --rm --network my_net2 --ip 172.18.0.2 --name test1   nginx
640  docker run -d --rm --network my_net2 --ip 15.16.17.18 --name test1   nginx
641  docker network inspect my_net2
642  history
643  docker network inspect my_net2
644  docker run -d --rm --network my_net2 --ip 15.16.17.19 --name test2   nginx
645  docker network inspect my_net2
```

DockerFile :

https://docs.docker.com/engine/reference/builder/

https://docs.docker.com/compose/

Résumé de l'infrastructure du projet Freezer :



Exemple de fichier Docker-Files :

https://github.com/nginxinc/docker-nginx/blob/master/Dockerfile-debian.template

```
108 lines (101 sloc)  5.12 KB                                   Raw  Blame
1  FROM debian:%%DEBIAN_VERSION%%-slim
2
3  LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"
4
5  ENV NGINX_VERSION   %%NGINX_VERSION%%
```

```
ENV PKG_RELEASE     %%PKG_RELEASE%%

RUN set -x \
# create nginx user/group first, to be consistent throughout docker variants
    && addgroup --system --gid 101 nginx \
    && adduser --system --disabled-login --ingroup nginx --no-create-home --home /nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
    && apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 ca-certificates \
    && \
    NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
    NGINX_GPGKEY_PATH=/usr/share/keyrings/nginx-archive-keyring.gpg; \
```

Pour construire une image : **Docker Build**

https://docs.docker.com/engine/reference/commandline/build/

Usage

```
$ docker build [OPTIONS] PATH | URL | -
```

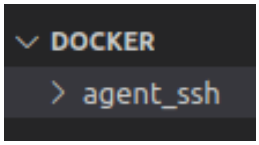
Refer to the options section for an overview of available `OPTIONS` for this command.

Description

The `docker build` command builds Docker images from a Dockerfile and a "context". A build's context is the set of files located in the specified `PATH` or `URL`. The build process can refer to any of the files in the context. For example, your build can use a *COPY* instruction to reference a file in the context.

The `URL` parameter can refer to three kinds of resources: Git repositories, pre-packaged tarball contexts and plain text files.

L'idéale quand on commence à faire des dockerfiles, c'est un dossier qui va stocker tous les dossiers contenant les dockerfiles pour chaque container



Pour créer un container depuis un fichier : **docker build –t "nom_container" -f "nom.fichier" "chemin_fichier"**

-t : donner un nom

-f : donner nom + lire le fichier dockerfiles

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker build -t jenkins_agent -f agent.Docker
file
```

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker build -t jenkins_agent -f agent.Docker
file .
Sending build context to Docker daemon  2.048kB
Step 1/1 : FROM archlinux
 ---> b457d2cc4d4c
Successfully built b457d2cc4d4c
Successfully tagged jenkins_agent:latest
```

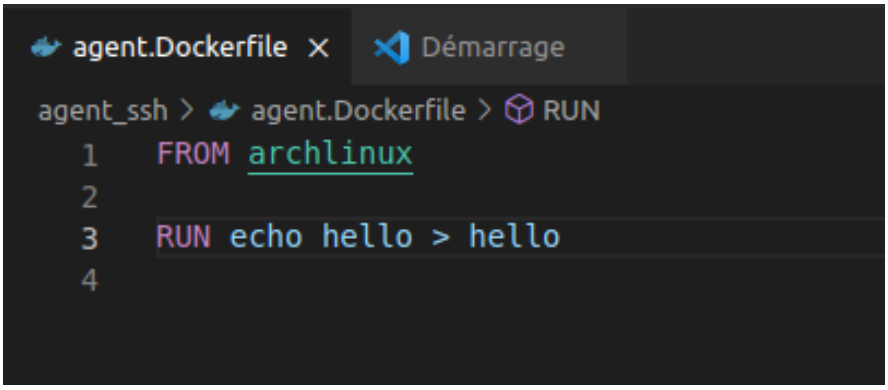
Notre image a bien été créer

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED       SIZE
archlinux               latest    b457d2cc4d4c   3 days ago    417MB
jenkins_agent           latest    b457d2cc4d4c   3 days ago    417MB
nginx                   latest    a99a39d070bf   2 weeks ago   142MB
instantlinux/proftpd    latest    1780a9b366fa   3 weeks ago   14.8MB
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$
```

```
[darksasuke@chocolat agent_ssh ]$ docker run -d --rm jenkins_agent
9d5035def58614a333b9619d590c892cc537345bd6cacecd58523efc493b6454
[darksasuke@chocolat agent_ssh ]$ dokcer ps
bash: dokcer : commande introuvable
[darksasuke@chocolat agent_ssh ]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PO
RTS     NAMES
01923f9255b6   nginx     "/docker-entrypoint.…"   51 minutes ago   Up 51 minutes   80
/tcp    test2
```

```
c8077c46cf54   nginx     "/docker-entrypoint.…"   53 minutes ago   Up 53 minutes   80
/tcp    test1
[darksasuke@chocolat agent_ssh ]$
```

Remplissons le fichier dockerfile :




```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker build -t jenkins_agent -f agent.Docker
file .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM archlinux
 ---> b457d2cc4d4c
Step 2/2 : RUN echo hello > hello
 ---> Running in 0bb96a574183
Removing intermediate container 0bb96a574183
 ---> 95f49e723641
Successfully built 95f49e723641
Successfully tagged jenkins_agent:latest
```

```
[darksasuke@chocolat agent_ssh ]$ docker run --rm -it jenkins_agent bash
[root@744fe84cc893 /]# ls
bin   dev  hello  lib    mnt  proc  run   srv  tmp  var
boot  etc  home   lib64  opt  root  sbin  sys  usr
[root@744fe84cc893 /]#
```

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker run --rm -it jenkins_agent bash
[root@955987cfaf4b /]# ls
bin   dev  hello  lib    mnt  proc  run   srv  tmp  var
boot  etc  home   lib64  opt  root  sbin  sys  usr
[root@955987cfaf4b /]#
```

On cherche à rendre persistant notre container : création d'un script de boucle infinie (juste pour tester)

```
[darksasuke@chocolat agent_ssh ]$ cat boucle.sh
#!/bin/bash


while [ a = a ]
do
        sleep 1
done
```

```
[darksasuke@chocolat agent_ssh ]$ docker run -d  --rm --name agent jenkins_agent
74fd82eab173f27725d30ca28f412aca0e7f4b671968e3c5a6af2383cdc01832
[darksasuke@chocolat agent_ssh ]$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS
          PORTS     NAMES
74fd82eab173   jenkins_agent   "/bin/sh -c 'bash bo…"   23 seconds ago      Up 22 sec
onds              agent
4913d2a7fcf1   b457d2cc4d4c    "bash"                   17 minutes ago      Up 17 min
utes              elegant_mendeleev
01923f9255b6   nginx          "/docker-entrypoint.…"   About an hour ago   Up About
an hour   80/tcp    test2
c8077c46cf54   nginx          "/docker-entrypoint.…"   About an hour ago   Up About
an hour   80/tcp    test1
```

```
 679  vim boucle.sh
 680  ls
```

```
681  cat boucle.sh
682  vim agent.Dockerfile
683  sudo docker ps
684  vim agent.Dockerfile
685  docker build -t jenkins_agent -f agent.Dockerfile .
686  docker ps
687  docker run -d  --rm --name agent jenkins_agent
688  docker ps
```

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker run -d --rm jenkins_agent
e090f7b5c24906a0cfc42f41781f050d0e003a6407ca007b170a5e2e7fe26bc3
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         PORTS     NAMES
e090f7b5c249   jenkins_agent   "/bin/sh -c 'bash bo…"   5 seconds ago   Up 4 seconds             distr
acted_nash
```

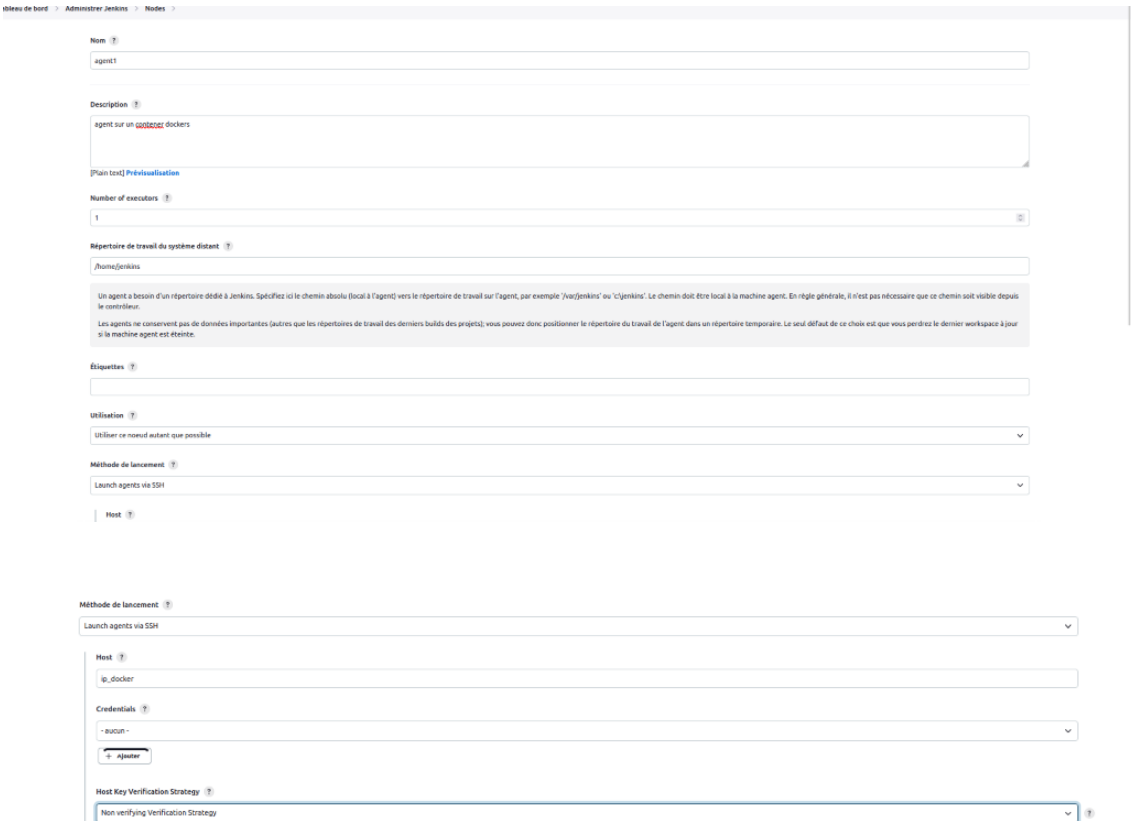
Dans un dockerfiles -> le but du jeu est d'avoir une seule étape RUN -> utilisation des &&
recommandé pour ne pas charger la RAM

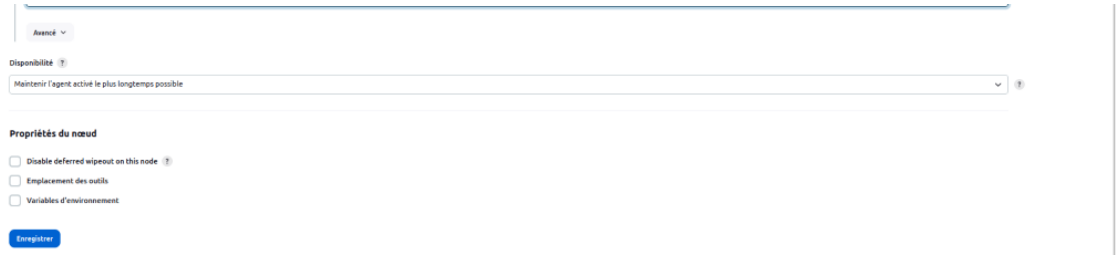
```
RUN set -x \
# create nginx user/group first, to be consistent throughout docker variants
    && addgroup --system --gid 101 nginx \
    && adduser --system --disabled-login --ingroup nginx --no-create-home --home /nonexistent --gecos "nginx user" --shell /bin/false --u
    && apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 ca-certificates \
    && \
```

Nous allons maintenant chercher à faire de notre conteneur un agent Jenkins :

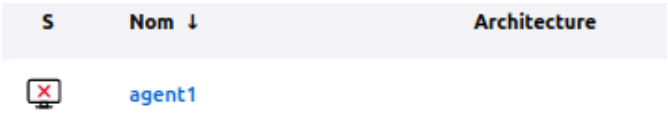
Jenkins: menu de gestion des nodes :

Créer nouveau Nodes :



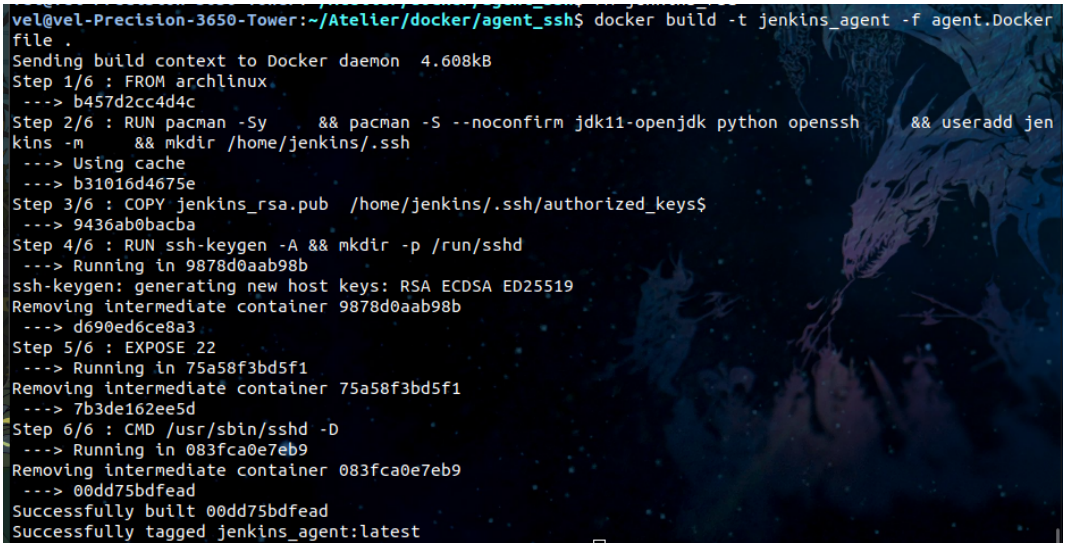


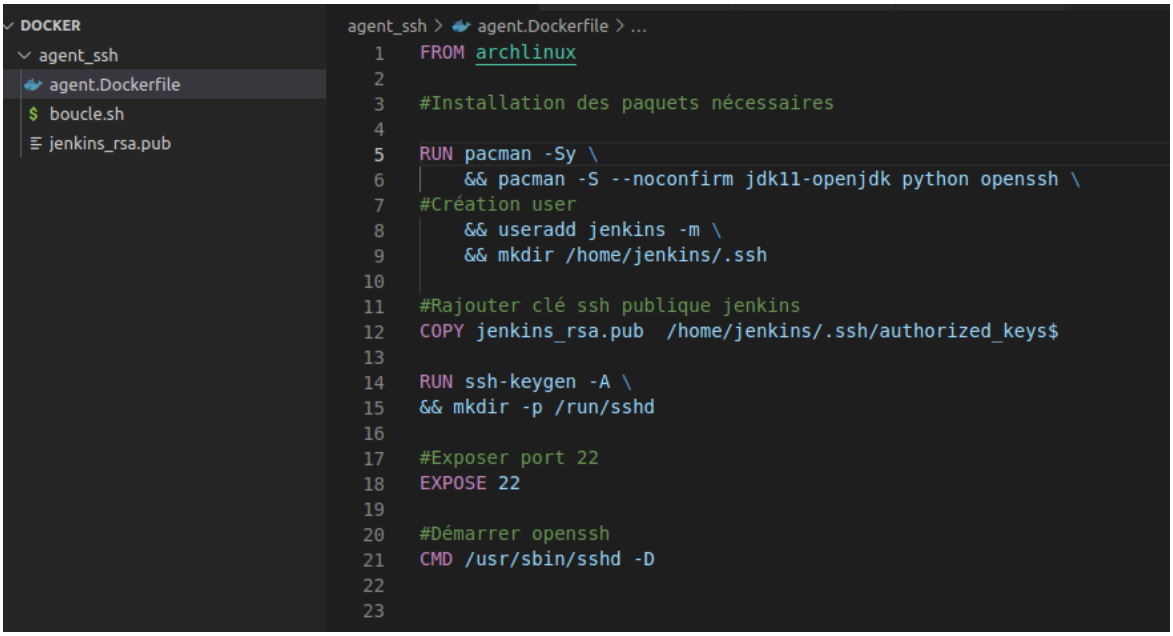
Voici le résultat :

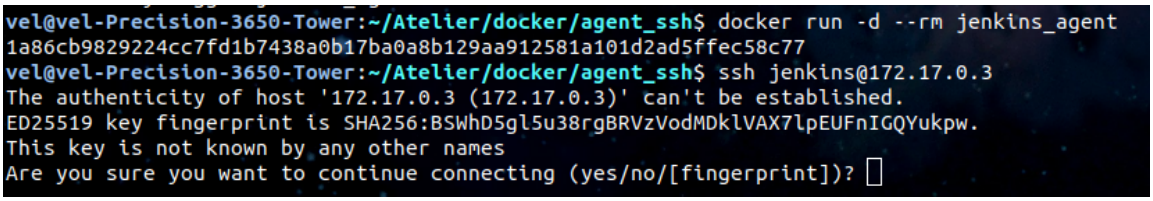


Pour notre serveur : besoin de divers éléments :

Java
SSHD
User : Jenkins –m
Copy : Jenkins_ssh.pub ~/.ssh/authorized_keys
Python
Dernière commande : SSHD
Exposer port 22






vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ docker run -d --rm jenkins_agent
1a86cb9829224cc7fd1b7438a0b17ba0a8b129aa912581a101d2ad5ffec58c77
vel@vel-Precision-3650-Tower:~/Atelier/docker/agent_ssh$ ssh jenkins@172.17.0.3
The authenticity of host '172.17.0.3 (172.17.0.3)' can't be established.
ED25519 key fingerprint is SHA256:BSWhD5gl5u38rgBRVzVodMDklVAX7lpEUFnIGQYukpw.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?