Chapitre 2 - Aller plus loin

Le réseau

Le réseau avec docker network

Docker permet une gestion avancée du réseau, entre un conteneur et la machine hôte, mais aussi entre les conteneurs.

Les réseaux docker peuvent être gérés avec la commande `docker network`.

```
docker network ls # lister les réseaux
docker network create mon_reseau # créer un réseau
docker network rm mon_reseau # supprimer un réseau
```

Il existe plusieurs types de réseaux docker, identifiés avec l'argument `--driver`.

- **Bridge (pont)**
  Utilisé par défaut, c'est un réseau interne à docker.
  Les conteneurs peuvent communiquer entre eux sur ce réseau.
- **Host**
  Le conteneur partage directement le réseau de l'hôte. Il n'est pas isolé et utilise l'adresse IP de l'hôte. Dans ce cas, la redirection de port n'est pas nécessaire.
- **None**
  Aucun réseau, isolation totale.

Docker propose par défaut un réseau pour chacun de ses 3 types.

```
debian@debian:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
e5ee2347d562   bridge    bridge    local
4b13b0e0f0a2   host      host      local
82c013605f89   none      null      local
```

Le réseau par défaut est `bridge`.

Driver Bridge

Ci-dessous un exemple avec l'image `busybox` qui embarque quelques utilitaires réseau, dont la commande `ping`.

```
debian@debian:~$ docker network create --driver=bridge testnetwork
f61e66fb4a3d13dc2713ec7d8d38cea1a68a7de519cd4d96a3fd655e93dcf23d
debian@debian:~$ docker run -tid --name=c1 --network=testnetwork busybox sh
495db35d7ebda8523b9b3b00746fe2b986b7b59a2d9d73d18cffb250d9cb4098
debian@debian:~$ docker run -ti --name=c2 --network=testnetwork busybox sh
/ # ping c1
PING c1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.153 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.119 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.110 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.147 ms
^C
--- c1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.110/0.132/0.153 ms
```

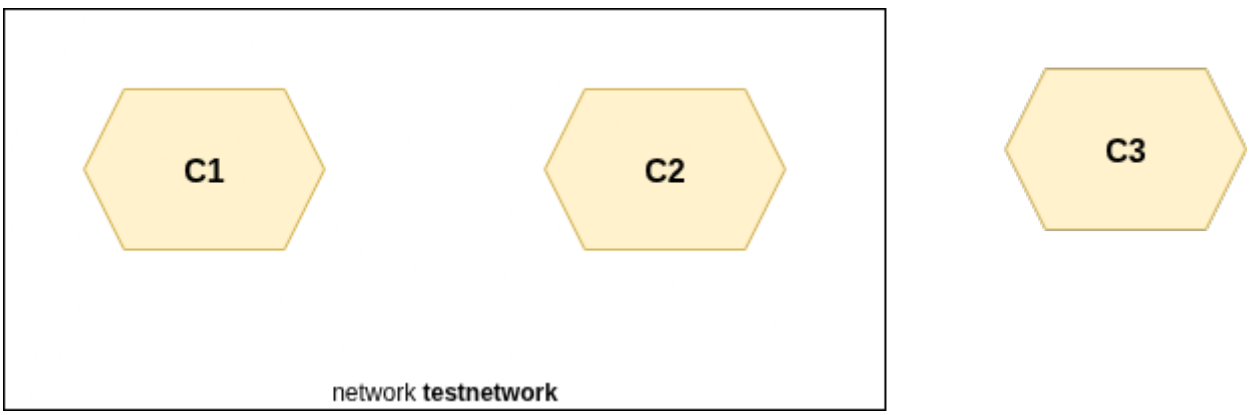
- Un réseau `testnetwork` est créé avec le driver `bridge`.
- Un conteneur `c1` est lancé en arrière-plan, il est attaché au réseau `testnetwork` avec `--network=testnetwork`.
- Un deuxième conteneur `c2` est lancé. Les deux conteneurs peuvent se pinguer.

> Chaque conteneur est visible dans le réseau à partir de son nom défini avec `--name`.

> Le réseau `bridge` par défaut de Docker isole les conteneurs entre-eux. Il est nécessaire de créer un autre réseau avec le driver `bridge` pour répéter l'expérience ci-dessus.

Un conteneur situé en dehors du réseau `testnetwork` ne peut pas pinguer `c1` et `c2` :

```
debian@debian:~$ docker run -ti --name=c3 busybox sh
/ # ping c1
ping: bad address 'c1'
/ # ping c2
ping: bad address 'c2'
```



Driver Host

Un conteneur lancé avec le driver `host` n'est pas isolé du réseau de l'hôte. Les ports ouverts sur le conteneur sont ouverts sur l'hôte.

Par exemple, un conteneur lancé à partir de l'image strm/helloworld-http écoutera à partir du port `80` de l'hôte, même sans redirection avec `--port`.

```
debian@debian:~$ docker run -ti --network=host strm/helloworld-http
Serving HTTP on 0.0.0.0 port 80 ...
```

> Le nom d'hôte du conteneur est affiché. Il est hérité depuis la machine hôte.

Gestion des réseaux

Il est possible de connecter un conteneur à un réseau existant avec `docker network connect`.

```
debian@debian:~$ docker network create --driver=bridge reseau2
562ad4a6e8b3105f619bd0deb30b89d5aaddde71babd6422ad912ccfa327687d
debian@debian:~$ docker run -tid --name=c5 ubuntu
f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158
debian@debian:~$ docker network connect reseau2 c5
```

Le détail d'un réseau peut être affiché avec la commande `docker network inspect`.

```
debian@debian:~$ docker network inspect reseau2
[
    {
        "Name": "reseau2",
        "Id": "562ad4a6e8b3105f619bd0deb30b89d5aaddde71babd6422ad912ccfa327687d",
        "Created": "2025-08-29T22:34:59.617521484+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158": {
                "Name": "c5",
                "EndpointID": "f71251f1d3c4d9be09376f687898d22a984017b91f6da6c8fe75fa65dedba007",
                "MacAddress": "5a:76:dd:aa:62:15",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Un réseau `bridge` embarque une configuration IP, de la même manière qu'un réseau local.
Le réseau est défini par un `subnet` et il possède une IP `gateway` attribuée à la machine hôte.

Dans l'exemple ci-dessus, le réseau `reseau2` possède le subnet `172.19.0.0` avec un masque de sous-réseau `255.255.0.0` (ou `/16`).
L'adresse IP de la machine hôte est `172.19.0.1` et celle du conteneur `c5` est `172.19.0.2`.

Administration : commandes utiles

> Les commandes ci-dessous sont directement copiées depuis la documentation de docker :
>
> https://docs.docker.com/reference/cli/docker/

Les usages principaux des commandes ont déjà été abordés jusqu'ici.

docker image

Command	Description
`docker image history`	Show the history of an image
`docker image import`	Import the contents from a tarball to create a filesystem image
`docker image inspect`	**Display detailed information on one or more images**
`docker image load`	Load an image from a tar archive or STDIN
`docker image ls`	List images
`docker image prune`	Remove unused images
`docker image pull`	Download an image from a registry
`docker image push`	Upload an image to a registry
`docker image rm`	Remove one or more images

Command	Description
`docker image save`	Save one or more images to a tar archive (streamed to STDOUT by default)
`docker image tag`	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

```
debian@debian:~$ docker image inspect ubuntu
[
    {
        "Id": "sha256:e0f16e6366fef4e695b9f8788819849d265cde40eb84300c0147a6e5261d2750",
        "RepoTags": [
            "ubuntu:24.04",
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfe1d2c0322aaf20f9"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2025-07-30T06:51:03.091147588Z",
        "DockerVersion": "24.0.7",
        "Author": "",
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 78122494,
        "GraphDriver": {
            "Data": {
                "MergedDir":
"/var/lib/docker/overlay2/8a59ab09262cee8d6004dcd5b978cd668baae2521d83a0621e5d1366fbd864a1/merged",
                "UpperDir":
"/var/lib/docker/overlay2/8a59ab09262cee8d6004dcd5b978cd668baae2521d83a0621e5d1366fbd864a1/diff",
                "WorkDir":
"/var/lib/docker/overlay2/8a59ab09262cee8d6004dcd5b978cd668baae2521d83a0621e5d1366fbd864a1/work"
            },
            "Name": "overlay2"
```

```
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:cd9664b1462ea111a41bdadf65ce077582cdc77e28683a4f6996dd03afcc56f5"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        },
        "Config": {
            "Cmd": [
                "/bin/bash"
            ],
            "Entrypoint": null,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Labels": {
                "org.opencontainers.image.ref.name": "ubuntu",
                "org.opencontainers.image.version": "24.04"
            },
            "OnBuild": null,
            "User": "",
            "Volumes": null,
            "WorkingDir": ""
        }
    }
]
```


docker container

Command	Description
`docker container attach`	Attach local standard input, output, and error streams to a running container
`docker container commit`	Create a new image from a container's changes
`docker container cp`	Copy files/folders between a container and the local filesystem

Command	Description
`docker container create`	Create a new container
`docker container diff`	Inspect changes to files or directories on a container's filesystem
`docker container exec`	Execute a command in a running container
`docker container export`	Export a container's filesystem as a tar archive
`docker container inspect`	Display detailed information on one or more containers
`docker container kill`	Kill one or more running containers
`docker container logs`	Fetch the logs of a container
`docker container ls`	List containers
`docker container pause`	Pause all processes within one or more containers
`docker container port`	List port mappings or a specific mapping for the container
`docker container prune`	Remove all stopped containers
`docker container rename`	Rename a container
`docker container restart`	Restart one or more containers
`docker container rm`	Remove one or more containers
`docker container run`	Create and run a new container from an image
`docker container start`	Start one or more stopped containers
`docker container stats`	Display a live stream of container(s) resource usage statistics
`docker container stop`	Stop one or more running containers
`docker container top`	Display the running processes of a container
`docker container unpause`	Unpause all processes within one or more containers
`docker container update`	Update configuration of one or more containers
`docker container wait`	Block until one or more containers stop, then print their exit codes

```
debian@debian:~$ docker inspect c5
[
    {
```

```
        "Id": "f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158",
        "Created": "2025-08-29T20:35:19.745912071Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3405,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-08-29T20:35:19.948819333Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:e0f16e6366fef4e695b9f8788819849d265cde40eb84300c0147a6e5261d2750",
        "ResolvConfPath":
"/var/lib/docker/containers/f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158/r
esolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158/h
ostname",
        "HostsPath":
"/var/lib/docker/containers/f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158/h
osts",
        "LogPath":
"/var/lib/docker/containers/f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158/f
5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158-json.log",
        "Name": "/c5",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
```

```
"Binds": null,
"ContainerIDFile": "",
"LogConfig": {
    "Type": "json-file",
    "Config": {}
},
"NetworkMode": "bridge",
"PortBindings": {},
"RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
},
"AutoRemove": false,
"VolumeDriver": "",
"VolumesFrom": null,
"ConsoleSize": [
    54,
    235
],
"CapAdd": null,
"CapDrop": null,
"CgroupnsMode": "private",
"Dns": [],
"DnsOptions": [],
"DnsSearch": [],
"ExtraHosts": null,
"GroupAdd": null,
"IpcMode": "private",
"Cgroup": "",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": false,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsernsMode": "",
"ShmSize": 67108864,
```

```
"Runtime": "runc",
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": [],
"BlkioDeviceReadBps": [],
"BlkioDeviceWriteBps": [],
"BlkioDeviceReadIOps": [],
"BlkioDeviceWriteIOps": [],
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": [],
"DeviceCgroupRules": null,
"DeviceRequests": null,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,
"OomKillDisable": null,
"PidsLimit": null,
"Ulimits": [],
"CpuCount": 0,
"CpuPercent": 0,
"IOMaximumIOps": 0,
"IOMaximumBandwidth": 0,
"MaskedPaths": [
    "/proc/asound",
    "/proc/acpi",
    "/proc/interrupts",
    "/proc/kcore",
    "/proc/keys",
    "/proc/latency_stats",
    "/proc/timer_list",
```

```
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware",
                "/sys/devices/virtual/powercap"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "ID": "f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158",
                "LowerDir":
"/var/lib/docker/overlay2/9c831cc72a1726631cb4c9e776e29e8192a4793b9172c2fabf2226fdfb8babb9-
init/diff:/var/lib/docker/overlay2/8a59ab09262cee8d6004dcd5b978cd668baae2521d83a0621e5d1366fbd
864a1/diff",
                "MergedDir":
"/var/lib/docker/overlay2/9c831cc72a1726631cb4c9e776e29e8192a4793b9172c2fabf2226fdfb8babb9/mer
ged",
                "UpperDir":
"/var/lib/docker/overlay2/9c831cc72a1726631cb4c9e776e29e8192a4793b9172c2fabf2226fdfb8babb9/dif
f",
                "WorkDir":
"/var/lib/docker/overlay2/9c831cc72a1726631cb4c9e776e29e8192a4793b9172c2fabf2226fdfb8babb9/wor
k"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "f5f93155c9e0",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
```

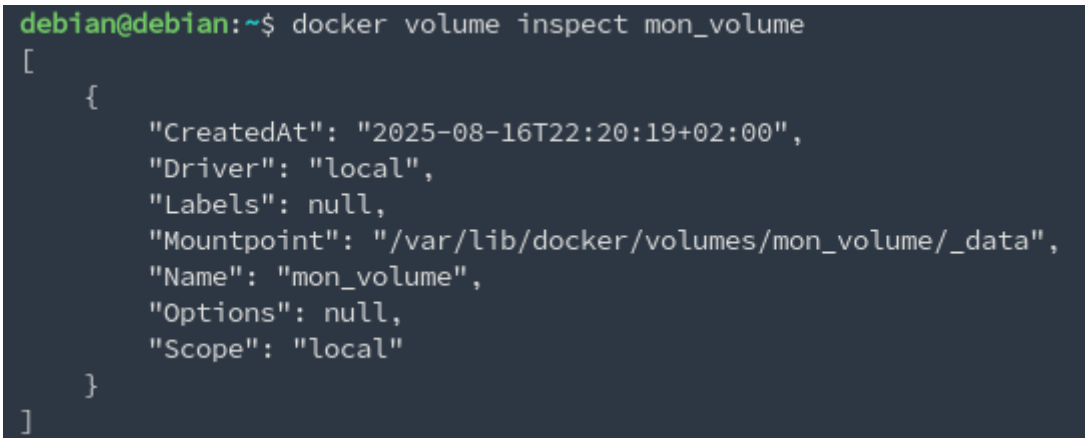
```
        "AttachStdout": false,
        "AttachStderr": false,
        "Tty": true,
        "OpenStdin": true,
        "StdinOnce": false,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
        ],
        "Cmd": [
            "/bin/bash"
        ],
        "Image": "ubuntu",
        "Volumes": null,
        "WorkingDir": "",
        "Entrypoint": null,
        "OnBuild": null,
        "Labels": {
            "org.opencontainers.image.ref.name": "ubuntu",
            "org.opencontainers.image.version": "24.04"
        }
    },
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID": "34132282ba3e544d46aec112f27ed949a571b114ebc2fee664fc034776e06be7",
        "SandboxKey": "/var/run/docker/netns/34132282ba3e",
        "Ports": {},
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "c239ab855b930537d28d6ebe5343f7eb9241eca8c7e086fe9894aa19c5f7cce7",
        "Gateway": "172.17.0.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "MacAddress": "56:4e:03:5c:c5:d4",
```

```
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "56:4e:03:5c:c5:d4",
                    "DriverOpts": null,
                    "GwPriority": 0,
                    "NetworkID":
"e5ee2347d56211a8f10602c4ebc07b27a15b2f9507af47f3c7c16bdbcf18fc6c",
                    "EndpointID":
"c239ab855b930537d28d6ebe5343f7eb9241eca8c7e086fe9894aa19c5f7cce7",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                },
                "reseau2": {
                    "IPAMConfig": {},
                    "Links": null,
                    "Aliases": [],
                    "MacAddress": "5a:76:dd:aa:62:15",
                    "DriverOpts": {},
                    "GwPriority": 0,
                    "NetworkID":
"562ad4a6e8b3105f619bd0deb30b89d5aaddde71babd6422ad912ccfa327687d",
                    "EndpointID":
"f71251f1d3c4d9be09376f687898d22a984017b91f6da6c8fe75fa65dedba007",
                    "Gateway": "172.19.0.1",
                    "IPAddress": "172.19.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": [
                        "c5",
```

```
                    "f5f93155c9e0"
                ]
            }
        }
    }
]
```


docker volume

Command	Description
`docker volume create`	Create a volume
`docker volume inspect`	**Display detailed information on one or more volumes**
`docker volume ls`	List volumes
`docker volume prune`	Remove unused local volumes
`docker volume rm`	Remove one or more volumes
`docker volume update`	Update a volume (cluster volumes only)



```
debian@debian:~$ docker volume inspect mon_volume
[
    {
        "CreatedAt": "2025-08-16T22:20:19+02:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mon_volume/_data",
        "Name": "mon_volume",
        "Options": null,
        "Scope": "local"
    }
]
```


docker network

Command	Description
`docker network connect`	Connect a container to a network
`docker network create`	Create a network
`docker network disconnect`	Disconnect a container from a network
`docker network inspect`	Display detailed information on one or more networks

Command	Description
`docker network ls`	List networks
`docker network prune`	Remove all unused networks
`docker network rm`	Remove one or more networks

```
debian@debian:~$ docker network inspect reseau2
[
    {
        "Name": "reseau2",
        "Id": "562ad4a6e8b3105f619bd0deb30b89d5aaddde71babd6422ad912ccfa327687d",
        "Created": "2025-08-29T22:34:59.617521484+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "f5f93155c9e02fc6a56d40b6d1bf5d0f3cd5c73106c72882f8878038398fe158": {
                "Name": "c5",
                "EndpointID": "f71251f1d3c4d9be09376f687898d22a984017b91f6da6c8fe75fa65dedba007",
                "MacAddress": "5a:76:dd:aa:62:15",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

docker stats

La commande `docker stats` affiche les informations essentielles des conteneurs actuellement lancés. Par défaut, la commande s'actualise toutes les 2 secondes.

```
debian@debian:~$ docker stats --no-stream
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O   PIDS
62d44597ccdd   test                0.00%     924KiB / 966.6MiB     0.09%     866B / 126B     0B / 0B     1
f5f93155c9e0   c5                  0.00%     916KiB / 966.6MiB     0.09%     3.02kB / 252B   0B / 0B     1
053c40ea4bd0   interesting_lamarr  0.02%     6.875MiB / 966.6MiB   0.71%     0B / 0B         0B / 0B     2
495db35d7ebd   c1                  0.00%     456KiB / 966.6MiB     0.05%     2.34kB / 602B   0B / 0B     1
```

docker system

Command	Description
`docker system df`	Show docker disk usage
`docker system events`	Get real time events from the server
`docker system info`	Display system-wide information
`docker system prune`	Remove unused data

```
debian@debian:~$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - unused build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
3b026e3dc434088a22d25729539ce93869c1119423c15bfb269121caa50feab8
65cf1cfa9b76d2174ab4c80f2f45c9e726abd95c343380b55aed60fc07a2ae7f

Total reclaimed space: 76B
debian@debian:~$
```

Administration avec Portainer

Interface graphique Portainer

Portainer est une interface web qui permet de gérer facilement des environnements Docker. Elle simplifie l'administration des conteneurs, images, volumes et réseaux sans avoir à utiliser la ligne de commande.

PORTAINER.iO

Portainer est une solution commerciale basée sur un outil open-source.Ci-dessous un extrait du site docs.portainer.io :

> **Portainer Community Edition (CE)** is our foundation. With over half a million regular users, CE is a powerful, open source toolset that allows you to easily build and manage containers in Docker, Docker Swarm, Kubernetes and Azure ACI.
>
> **Portainer Business Edition (BE)** is our commercial offering. With features geared towards businesses and larger organizations such as Role-Based Access Control, registry management, and dedicated support , Portainer BE is a powerful toolset that allows you to easily build and manage containers in Docker, Docker Swarm, Kubernetes, Podman and Azure ACI.

Nous utiliserons **Portainer Community Edition (CE)**.

Installation

> La procédure d'installation détaillée est disponible sur docs.portainer.io.

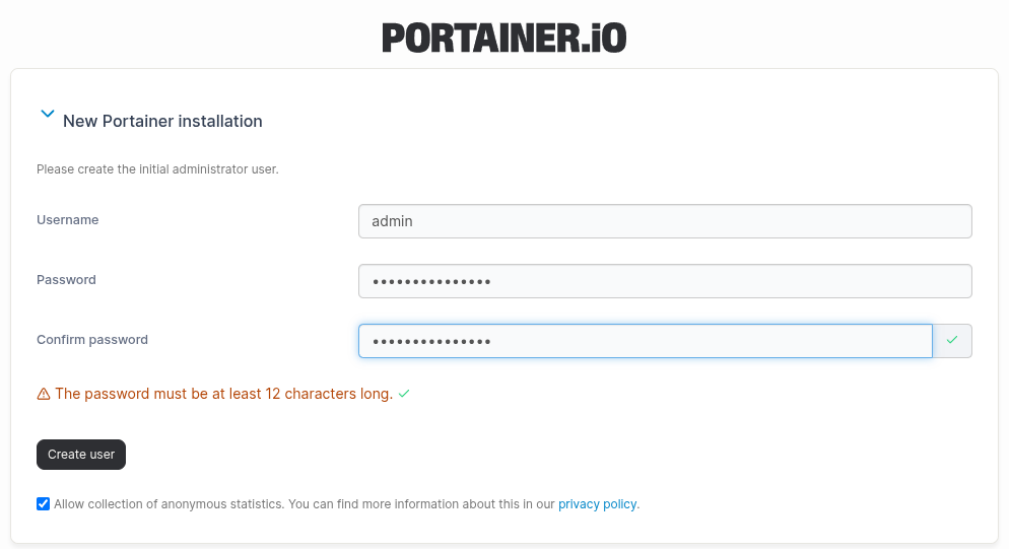
L'installation de Portainer s'effectue simplement avec une image docker :

```
docker volume create portainer_data
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always \
-v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-
ce:lts
```

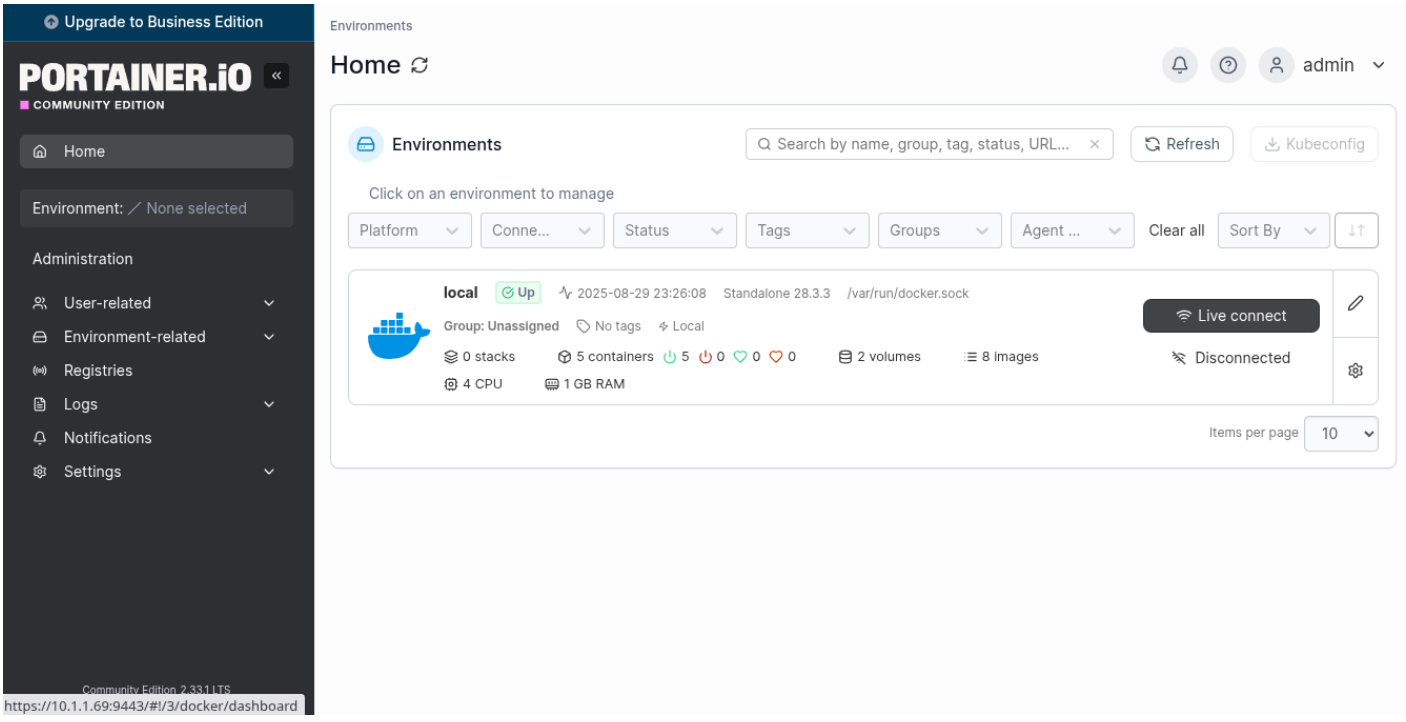
Portainer est accessible à l'adresse : https://127.0.0.1:9443/.

> Remplacez 127.0.0.1 par l'IP de votre hôte docker si nécessaire.

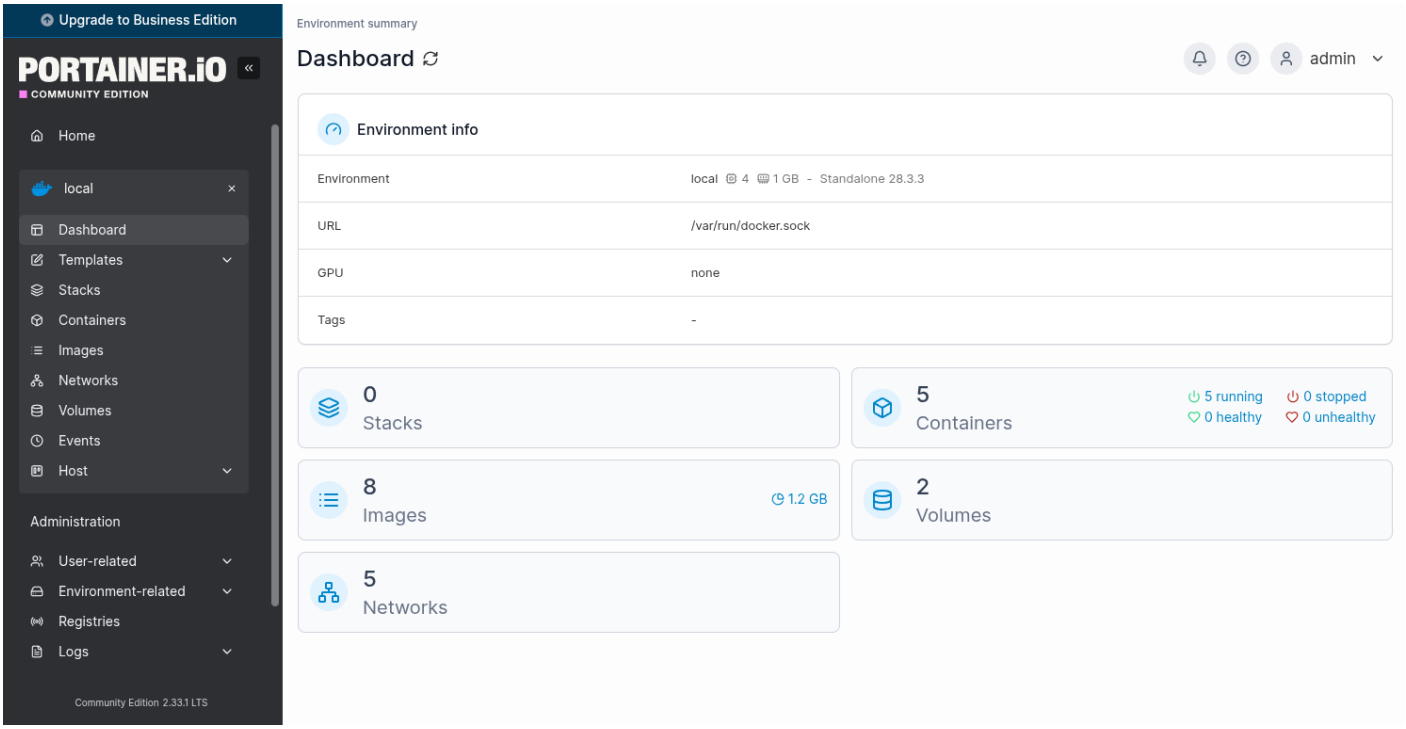
La création d'un utilisateur est obligatoire avant la première connexion.



Il faut ensuite sélectionner l'environnement local.



L'administration des containers, images, volumes, network est disponible.



Construction d'image avec Dockerfile

Dockerfile et layers

Les images Docker fonctionnent par couches successives d'instructions, **les layers**.

Un **Dockerfile** est un fichier texte qui décrit les différentes couches d'une image Docker.

Chaque couche permet d'ajouter des actions à l'image, par exemple :

- Installation de dépendances
- Configuration d'un environnement
- Copie de fichiers depuis la machine hôte
- Définition de la commande lancée au démarrage du conteneur
- Etc.

> Le dockerfile est comparable à une recette de cuisine : chaque instruction du Dockerfile représente une étape dans la préparation de l'environnement logiciel souhaité.

Son utilisation offre certains avantages :

- **Automatisation** : chaque build est reproductible et scripté.
- **Traçabilité** : le fichier peut être versionné avec Git.
- **Portabilité** : un Dockerfile peut être utilisé sur n'importe quelle machine.
- **Modularité** : il permet de construire des images à partir d'autres images, facilitant la réutilisation.

Ci-dessous un exemple de Dockerfile :

```
FROM debian:trixie # point de départ : une image existante
RUN apt-get update -y # commande à éxécuter
RUN apt-get install fastfetch -y # une autre commande, pour une nouvelle couche
ENTRYPOINT ["fastfetch"] # point d'entrée au lancement du conteneur
```

Les instructions d'un Dockerfile

> N'hésitez pas à consulter la documentation de Docker à ce sujet : Dockerfile reference.

`FROM` - Définir l'image de base

```
FROM ubuntu:24.04
```

C'est le point de départ de l'image : une autre image de base officielle ou personnalisée.

`LABEL` - Ajouter des métadonnées

```
LABEL Maintainer="Thibaud FRICHET"
LABEL Description="Description de l'image"
```

Les métadonnées de l'image s'affichent avec `docker image inspect`.

`WORKDIR` - Définir le répertoire de travail

```
WORKDIR /var/www/html
```

Les instructions `RUN`, `CMD` et `ENTRYPOINT` s'exécutent dans le répertoire de travail.

`COPY` - Copier des fichiers locaux dans l'image

```
# copie fichier.txt à partir du répertoire courant de la machine hôte
# dans le répertoire courant de l'image (défini avec WORKDIR)
COPY fichier.txt ./

# copie le répertoire sources depuis la machine hôte vers le répertoire /var/www de l'image
COPY /home/thibaud/projet/sources/ /var/www/
```

`RUN` - Exécuter des commandes pendant la construction

```
# installation et lancement de Redis
RUN apt-get update -y
RUN apt-get install redis-server -y
RUN service redis-server start
```

Les commandes `RUN` s’exécutent pendant la construction de l'image, pas au démarrage d'un conteneur.

> Les commandes doivent se terminer sans saisie utilisateur.

`EXPOSE` - Indiquer les ports utilisés par l'application

```
EXPOSE 80
EXPOSE 443
```

Ce sont les ports d'écoute du conteneur, ils peuvent être redirigés avec `docker run -p 90:80` à l'exécution.

`CMD` - Définir les commandes par défaut à exécuter

L'instruction CMD accepte un tableau en paramètre.

```
CMD ["date", "echo Bonjour"] # affiche la date courante, puis "Bonjour"
```

Contrairement à `RUN`, l'instruction `CMD` est exécutée au démarrage du conteneur, pas pendant la construction.

`ENTRYPOINT` - Définir le point d'entrée du conteneur

```
ENTRYPOINT ["service nginx start"] # lance le serveur web nginx
```

`CMD` et `ENTRYPOINT` définissent les commandes exécutées au démarrage d'un conteneur, mais avec un comportement différent :

- `CMD` indique la commande avec des arguments par défaut qui peuvent être remplacés lors de l’exécution avec docker run.
- `ENTRYPOINT` fixe le point d’entrée du conteneur, rendant son comportement plus rigide.

Lorsqu’ils sont utilisés ensemble, `ENTRYPOINT` définit le programme à exécuter et `CMD` les arguments par défaut.

> Dans la majorité des cas, vous pouvez utiliser l'un ou l'autre uniquement.

Bonnes pratiques

- Utilisez des images légères. Exemples : `python:3.10-slim`, `nginx-stable-alpine`, etc.

- Utilisez des versions fixes dans vos dépendances. Évitez les tags `-latest` par exemple.

- Minimisez le nombre de couches, par exemple avec plusieurs commandes sur une ligne `RUN` et `&&`.
  ```
  RUN apt-get update -y && apt-get install redis-server
  ```

- Utilisez `.dockerignore` pour exclure les fichiers inutiles, de la même manière qu'un `.gitignore`.
  Le contenu du répertoire courant de la machine hôte est envoyé à Docker lors de la construction d'une image. Docker appelle cela le `context`.

- Respectez la convention de nommage d'un dockerfile :
  - Par défaut : `Dockerfile`
  - Avec un préfixe : `dev.Dockerfile`, `prod.Dockerfile`, `app-v3.Dockerfile`

La construction de l'image

La construction d'une image docker s'effectue avec `docker build`.

Le paramètre `-t`, `--tag` indique le nom de l'image et sa version. Le paramètre `-f`, `--file` le nom du fichier dockerfile, par défaut `Dockerfile`.

Un répertoire de contexte doit être indiqué, par exemple le répertoire courant avec `.`.

```
docker build -t mon_appli:1.0.0 -f monappli.Dockerfile .
```

```
debian@debian:~/docker_build_exemple$ cat Dockerfile
FROM debian:trixie
RUN apt-get update -y && apt-get install fastfetch -y
ENTRYPOINT ["fastfetch"]
debian@debian:~/docker_build_exemple$ docker build -t mon_appli:1.0.0 .
[+] Building 10.0s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 135B
 => [internal] load metadata for docker.io/library/debian:trixie
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/debian:trixie@sha256:6d87375016340817ac2391e670971725a9981cfc24e221c47734681ed0f6c0f5
 => [2/2] RUN apt-get update -y && apt-get install fastfetch -y
 => exporting to image
 => => exporting layers
 => => writing image sha256:080ef57bfee7674fd7793b71e26d6c42f8bdcc2463e585a087e7bf921d7bf585
 => => naming to docker.io/library/mon_appli:1.0.0
```

```
debian@debian:~/docker_build_exemple$ docker image ls
REPOSITORY                TAG          IMAGE ID       CREATED              SIZE
mon_appli                 1.0.0        080ef57bfee7   About a minute ago   144MB
portainer/portainer-ce    lts          79b66b8e24d3   5 days ago           186MB
debian                    bookworm     3d2058890b96   3 weeks ago          117MB
debian                    latest       047bd8d81940   3 weeks ago          120MB
hello-world               latest       1b44b5a3e06a   3 weeks ago          10.1kB
php                       8.4-apache   27a77f7fe238   3 weeks ago          511MB
ubuntu                    24.04        e0f16e6366fe   4 weeks ago          78.1MB
ubuntu                    latest       e0f16e6366fe   4 weeks ago          78.1MB
busybox                   latest       0ed463b26dae   11 months ago        4.43MB
strm/helloworld-http      latest       29acbf1818de   7 years ago          158MB
debian@debian:~/docker_build_exemple$ docker run -ti mon_appli:1.0.0
        _,met$$$$$gg.          root@4f1071111007
     ,g$$$$$$$$$$$$$$$P.       ----------------
   ,g$$P""       """Y$$.".     OS: Debian GNU/Linux 13 (trixie) x86_64
  ,$$P'              `$$$.     Host: KVM/QEMU Standard PC (i440FX + PIIX, 1996) (pc-i440fx-2.2)
',$$P       ,ggs.     `$$b:   Kernel: Linux 6.12.41+deb13-amd64
`d$$'     ,$P"'   .    $$$    Uptime: 1 day, 30 mins
 $$P      d$'     ,    $$P    Packages: 80 (dpkg)
 $$:      $$.   -    ,d$$'    Shell: bash 5.2.37
 $$;      Y$b._   _,d$P'      Display (Virtual-1): 1024x768
 Y$$.    `.`"Y$$$$P"'         Terminal: xterm
 `$$b      "-.__              Terminal Font: fixed (8.0pt)
  `Y$$b                       CPU: AMD Ryzen Embedded V1500B (4) @ 2.20 GHz
   `Y$$.                      GPU: Unknown Device 1111 (VGA compatible)
     `$$b.                    Memory: 354.06 MiB / 966.56 MiB (37%)
       `Y$$b.                 Swap: 0 B / 2.00 GiB (0%)
         `"Y$b._              Disk (/): 2.86 GiB / 95.90 GiB (3%) - overlay
             `"""             Local IP (eth0): 172.17.0.3/16
                              Locale: C

                              
debian@debian:~/docker_build_exemple$
```

> Par la suite, une image Docker peut être exportée dans un fichier ou uploadée sur un registre privé par exemple.

TP : Création d'une image Docker

Objectif

L'objectif est de **construire une image** qui embarque un script Python accessible en web.

> Aucune connaissance de Python n'est requise.

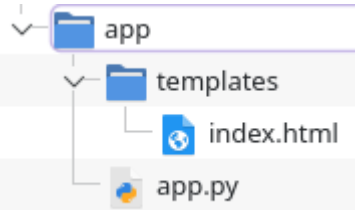
Consignes

1 - Téléchargement de l'application

Téléchargez le fichier zip suivant et dézippez-le.

https://formation-tfrichet-assets.s3.fr-par.scw.cloud/docker-tp-2/docker-tp-2.zip

Vous obtenez un dossier `app`.



2 - Création du Dockerfile

Créez et buildez un Dockerfile en respectant les consignes ci-dessous.

- L'image de départ est `python:3.10-slim`.
- Le répertoire de travail est `/app`.

- Pendant la construction de l'image, les commandes doivent être exécutées :

```
apt-get update && apt-get install -y procps
pip install flask
```

- L'application (dossier `app` téléchargé) doit être copiée dans le répertoire de travail de l'image.
- L'image écoute sur le port `5000`.
- Les commandes suivantes sont exécutées au démarrage d'un conteneur : `python` et `app.py`.
- Nommez l'image `tp2`.

3 - Exécution d'un conteneur

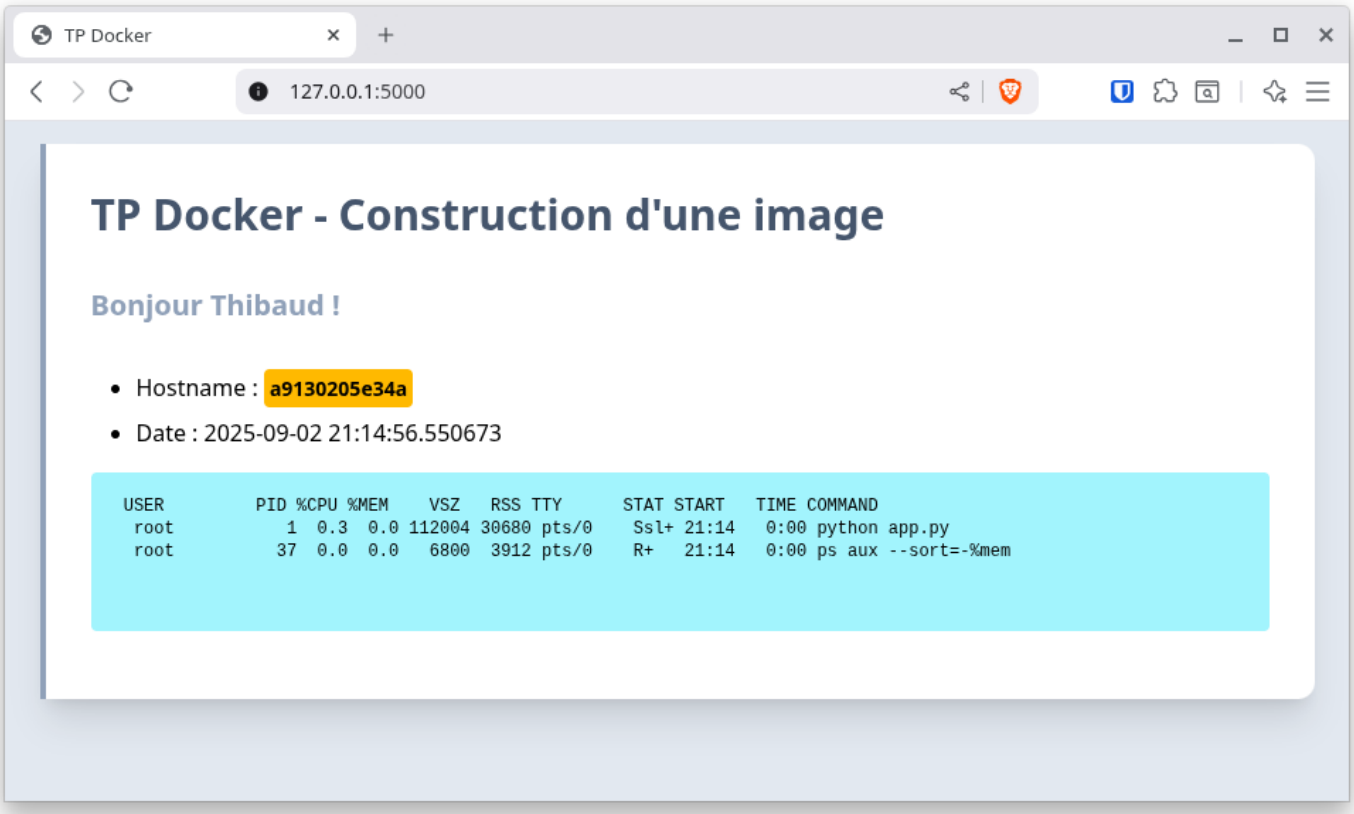
Exécutez un conteneur avec la commande ci-dessous. N'oubliez pas de renseigner votre nom.

```
docker run -ti --rm -p 5000:5000 -e NAME=VOTRE_NOM_ICI tp2
```

Rendez-vous sur http://127.0.0.1:5000.

Résultat attendu

> Le contenu du Dockerfile et la capture d'écran ci-dessous sont attendus pour valider le TP.


TP Docker
127.0.0.1:5000
TP Docker - Construction d'une image
Bonjour Thibaud !
Hostname : a9130205e34a
Date : 2025-09-02 21:14:56.550673
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.3 0.0 112004 30680 pts/0 Ssl+ 21:14 0:00 python app.py
root 37 0.0 0.0 6800 3912 pts/0 R+ 21:14 0:00 ps aux --sort=-%mem