

Le réseau (Networks)

Introduction

Source : <https://docs.docker.com/network/>

La mise en réseau de conteneurs fait référence à la capacité des conteneurs à :

- ✓ se connecter et à communiquer entre eux,
- ✓ ou avec d'autres processus non Docker.

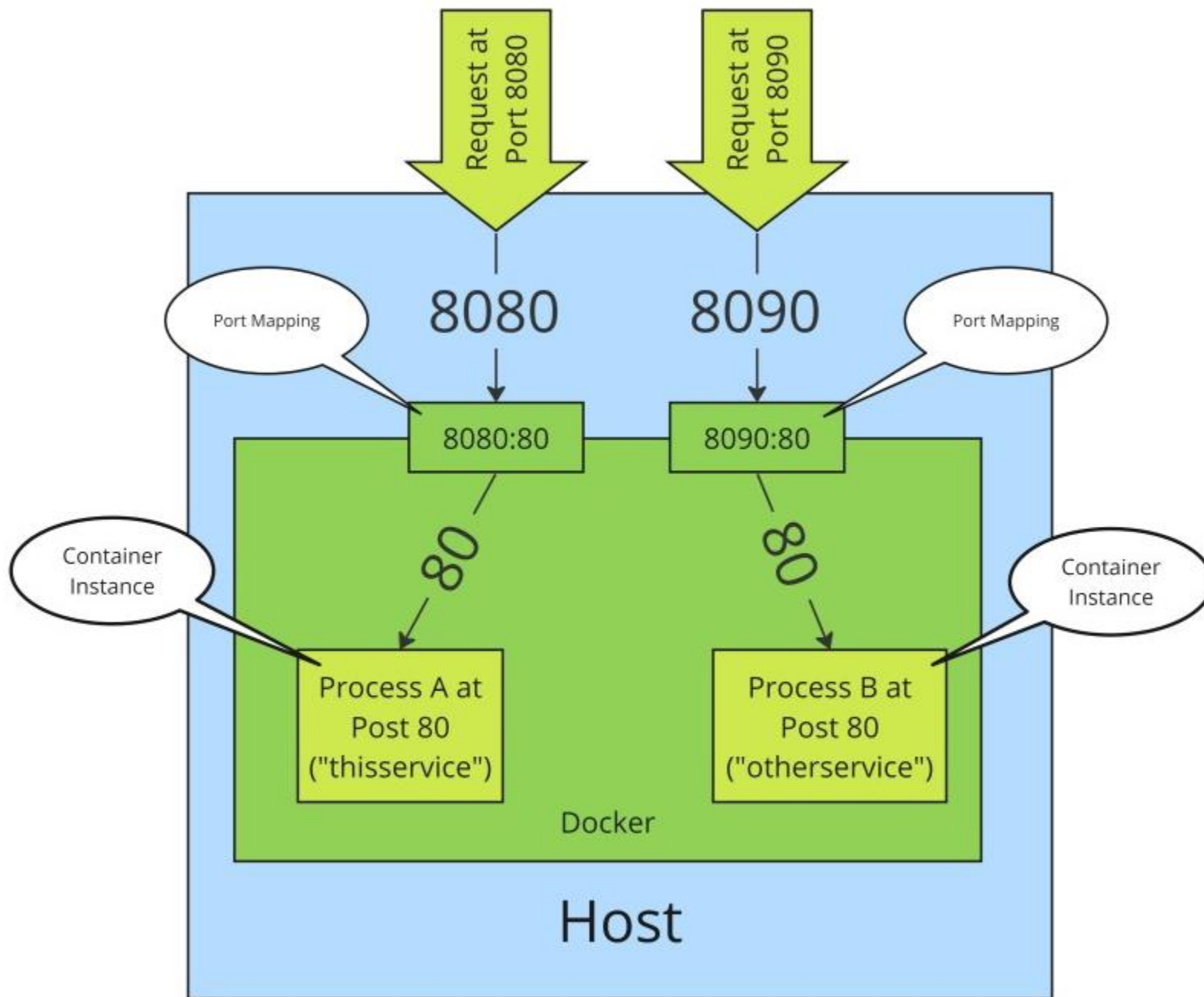
Par défaut, lorsque vous créez ou exécutez un conteneur à l'aide de docker create ou docker run :

- le conteneur n'expose aucun de ses ports au monde extérieur.

Utilisez l'indicateur --publish ou -p pour rendre un port disponible aux services en dehors de Docker :

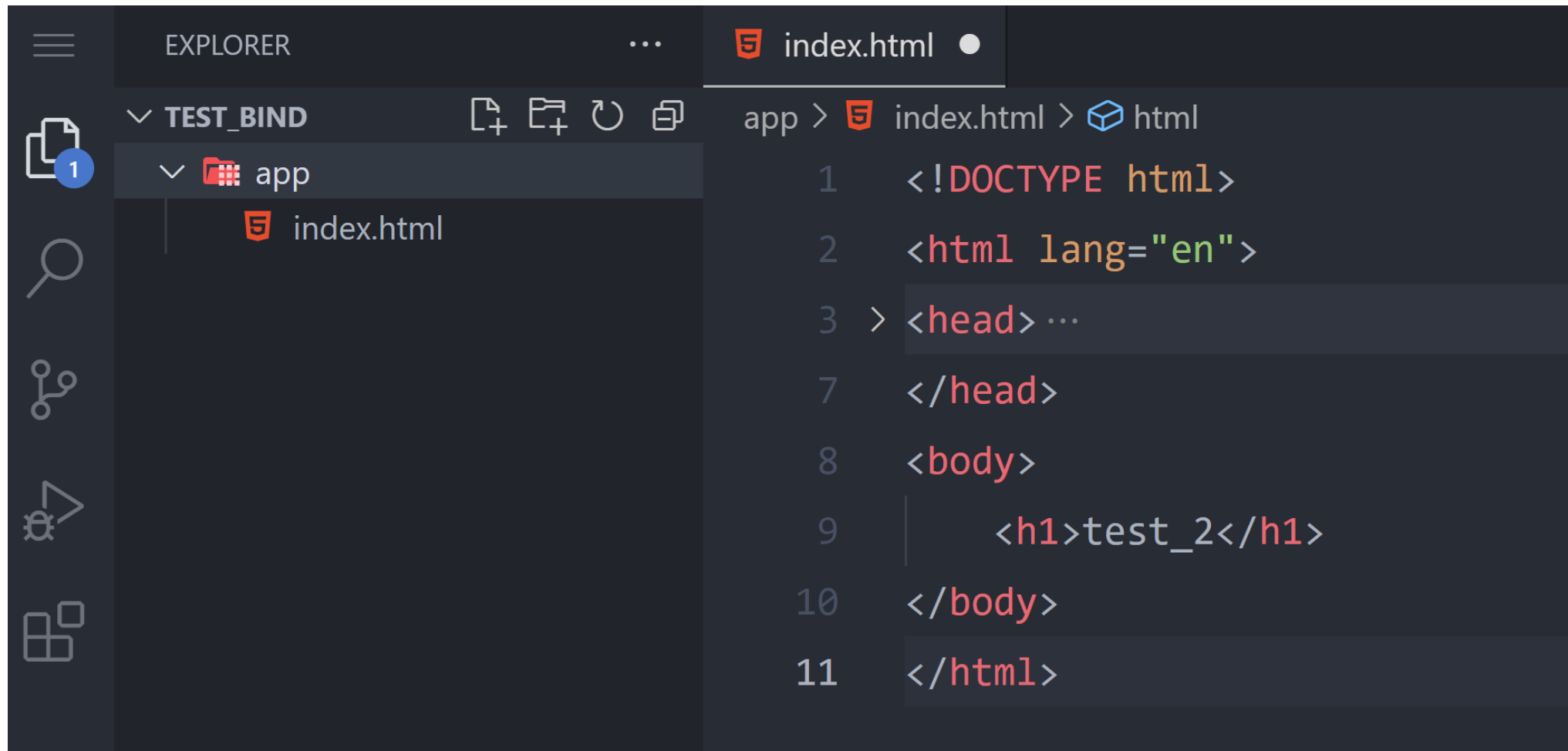
- ✓ Cela crée une règle de pare-feu dans l'hôte,

mappant un port de conteneur à un port de l'hôte Docker vers le monde extérieur.



Cas concret :

Préparons un projet en local



```
EXPLORER
TEST_BIND
├── app
│   └── index.html
└── ...

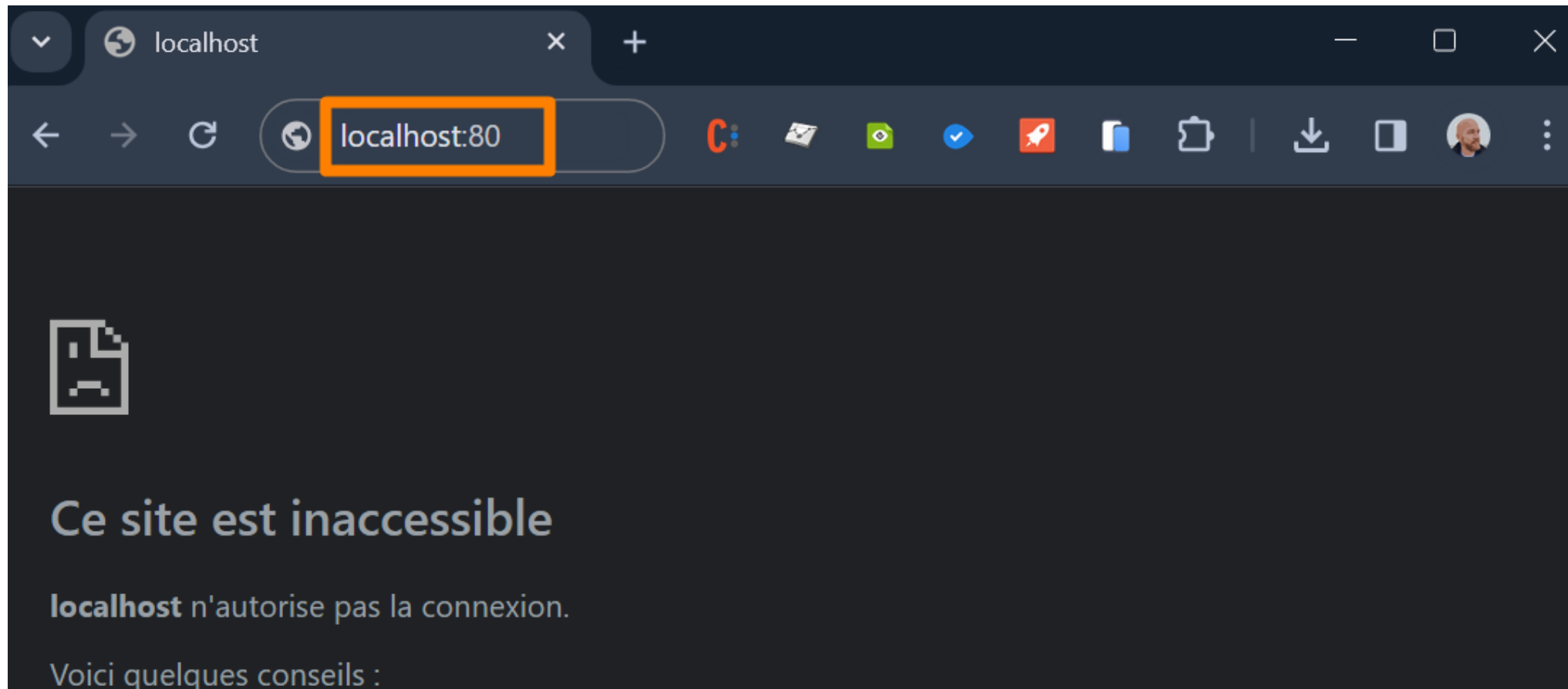
index.html
1  <!DOCTYPE html>
2  <html lang="en">
3  > <head> ...
7  </head>
8  <body>
9  |   <h1>test_2</h1>
10 </body>
11 </html>
```

Cas concret :

Tentons de le binder sur un conteneur sans mapper le port 80

```
docker run -d --name web --mount type=bind,src=${PWD}/app,target=/usr/share/nginx/html nginx:latest
```

Le port par défaut étant 80, il devrait être accessible ici : <http://localhost/>

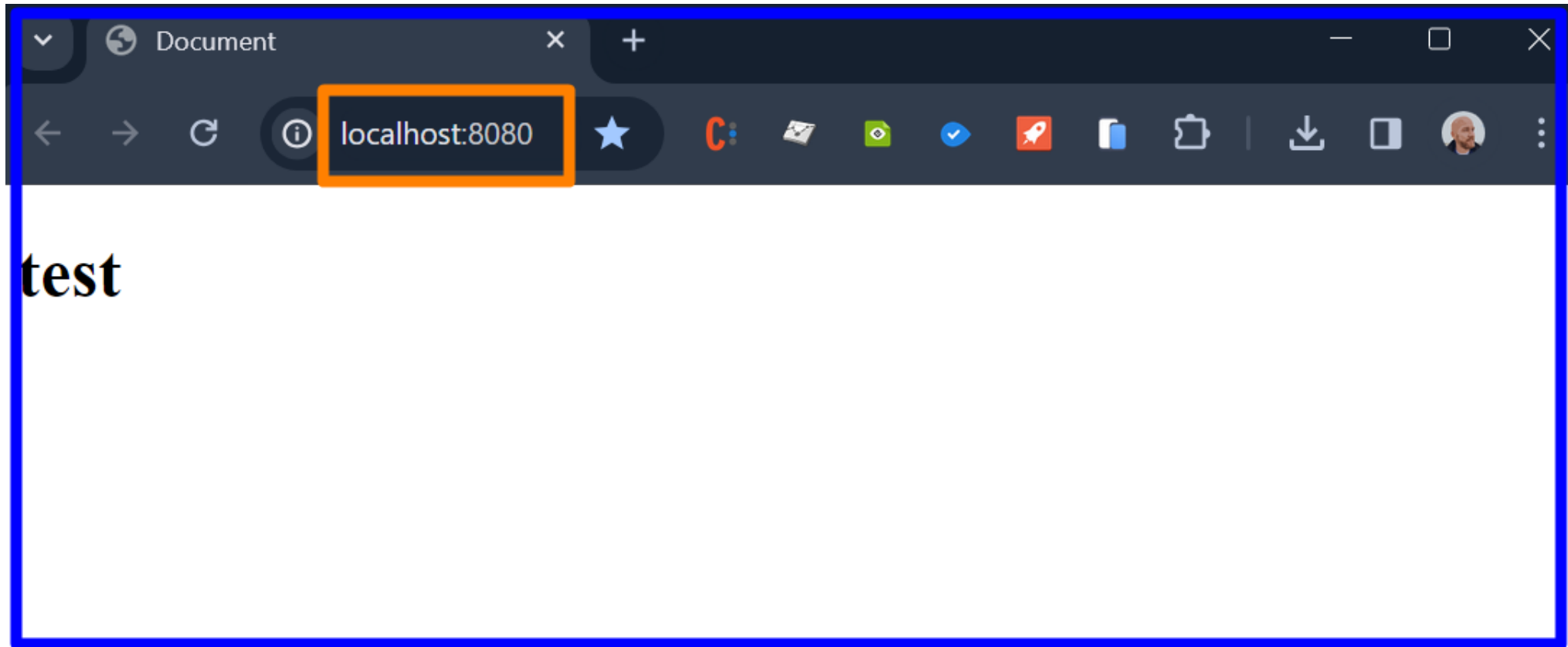


Cas concret :

Tentons maintenant de le binder sur un conteneur avec le mappage

```
docker run -d -p 8080:80 --name web --mount type=bind,src=${PWD}/app,target=/usr/share/nginx/html nginx:latest
```

Le conteneur est maintenant accessible ici : <http://localhost:8080>



Présentation des pilotes réseau

Le sous-système réseau de Docker est 'pluggable', à l'aide de pilotes (driver).

Plusieurs driver existent par défaut et fournissent les fonctionnalités réseau de base :

- ✓ Bridge,
- ✓ Host,
- ✓ Overlay,
- ✓ Ipvlan,
- ✓ Macvlan,
- ✓ None,
- ✓ Plugins réseau.

Les principaux :

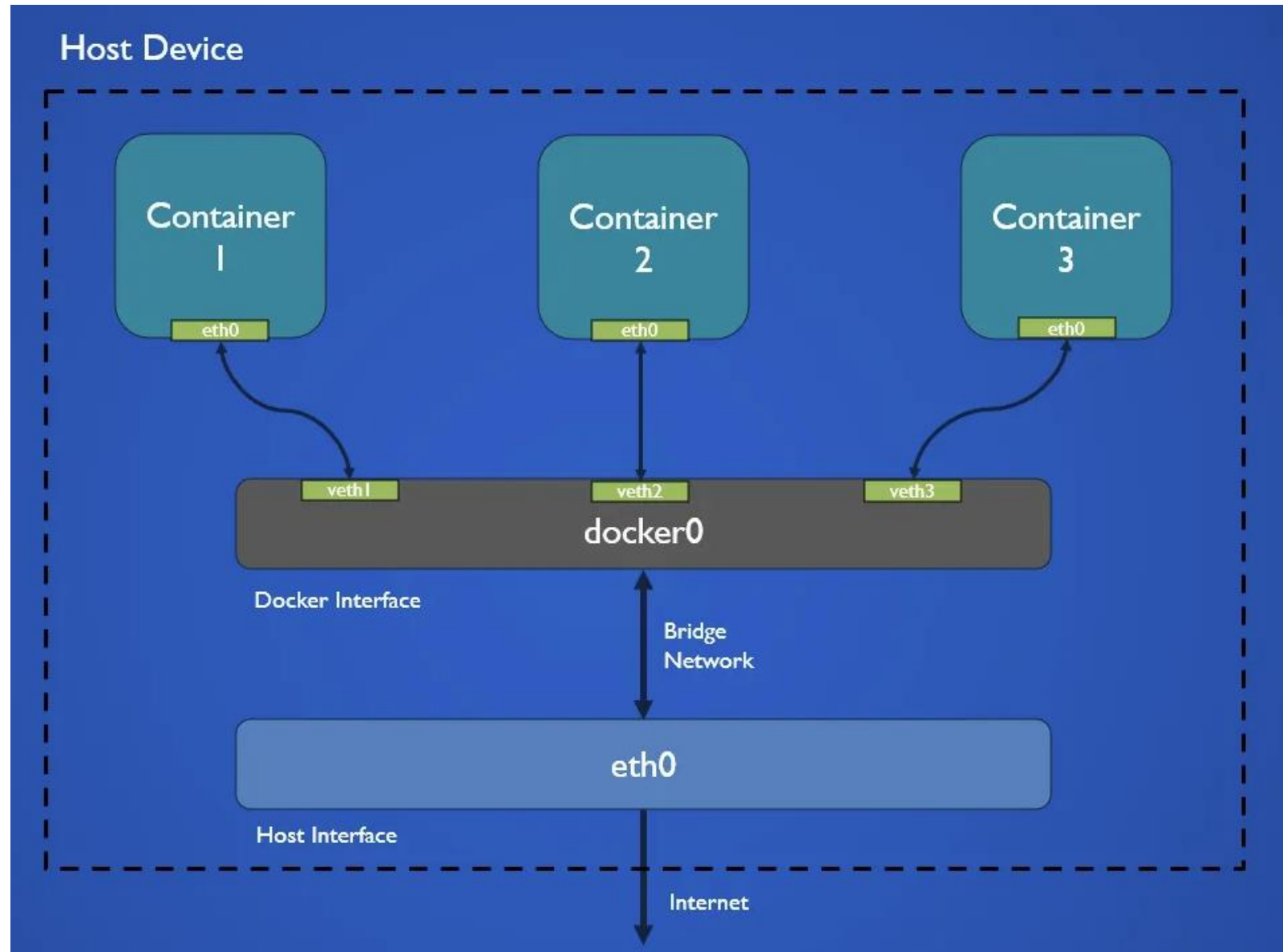
✓ bridge :

le pilote réseau par défaut, si vous ne spécifiez pas de pilote, il s'agit du type de réseau que vous créez.

Les réseaux de pont sont couramment utilisés lorsque votre application s'exécute dans un conteneur qui doit communiquer avec d'autres conteneurs sur le même hôte.

Les principaux :

✓ bridge :



Les principaux :

✓ host :

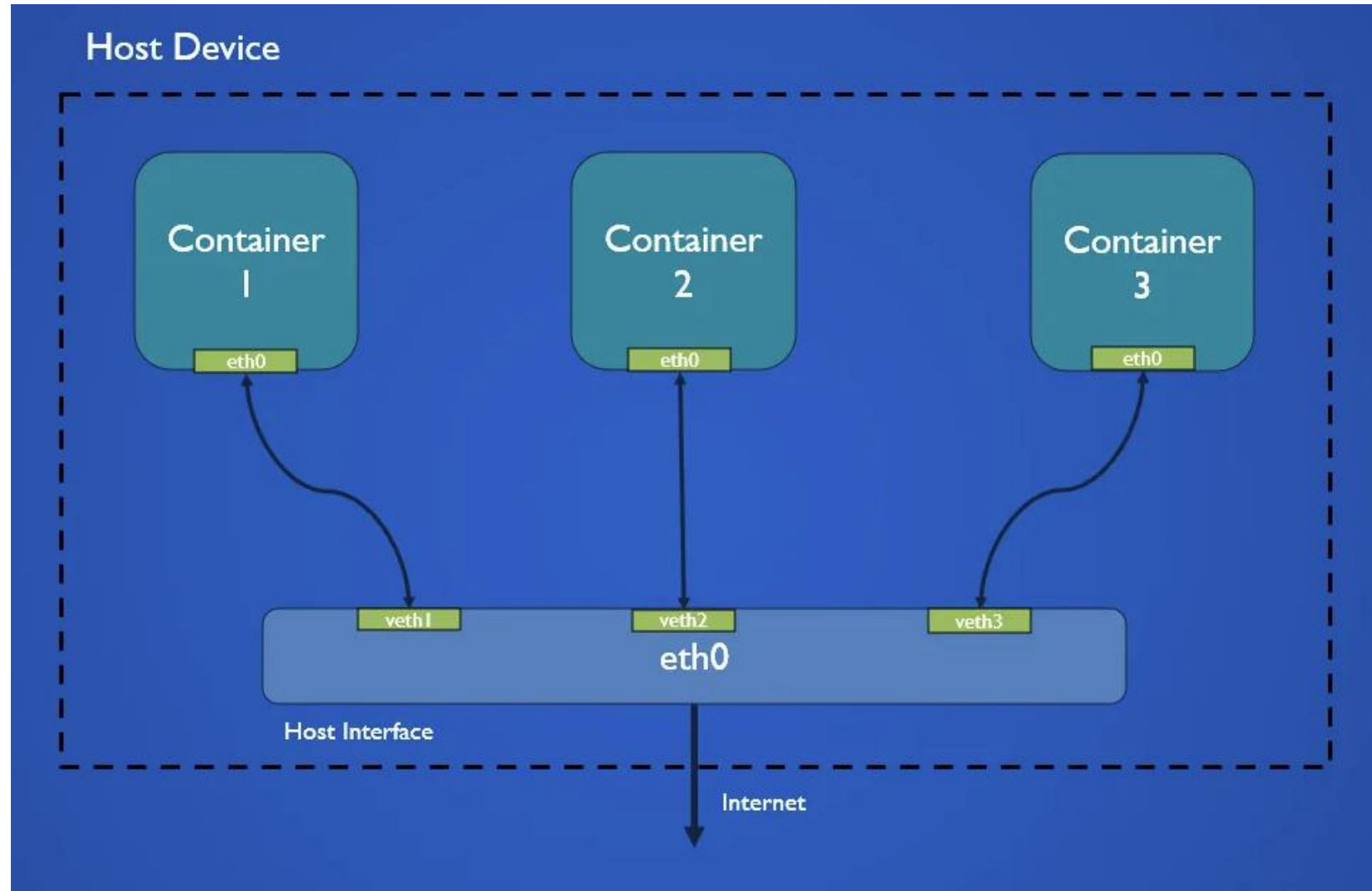
Supprime l'isolation réseau entre le conteneur et l'hôte Docker et utilisez directement le réseau de l'hôte.

Le pilote host permet à un conteneur de se lier directement à l'interface réseau du système hôte.

La pile réseau des conteneurs n'est pas isolée de l'hôte dans ce mode, le conteneur utilise le même espace de noms réseau que le système hôte.

Les principaux :

✓ host :



Les principaux :

✓ none :

isole complètement un conteneur de l'hôte et des autres conteneurs.

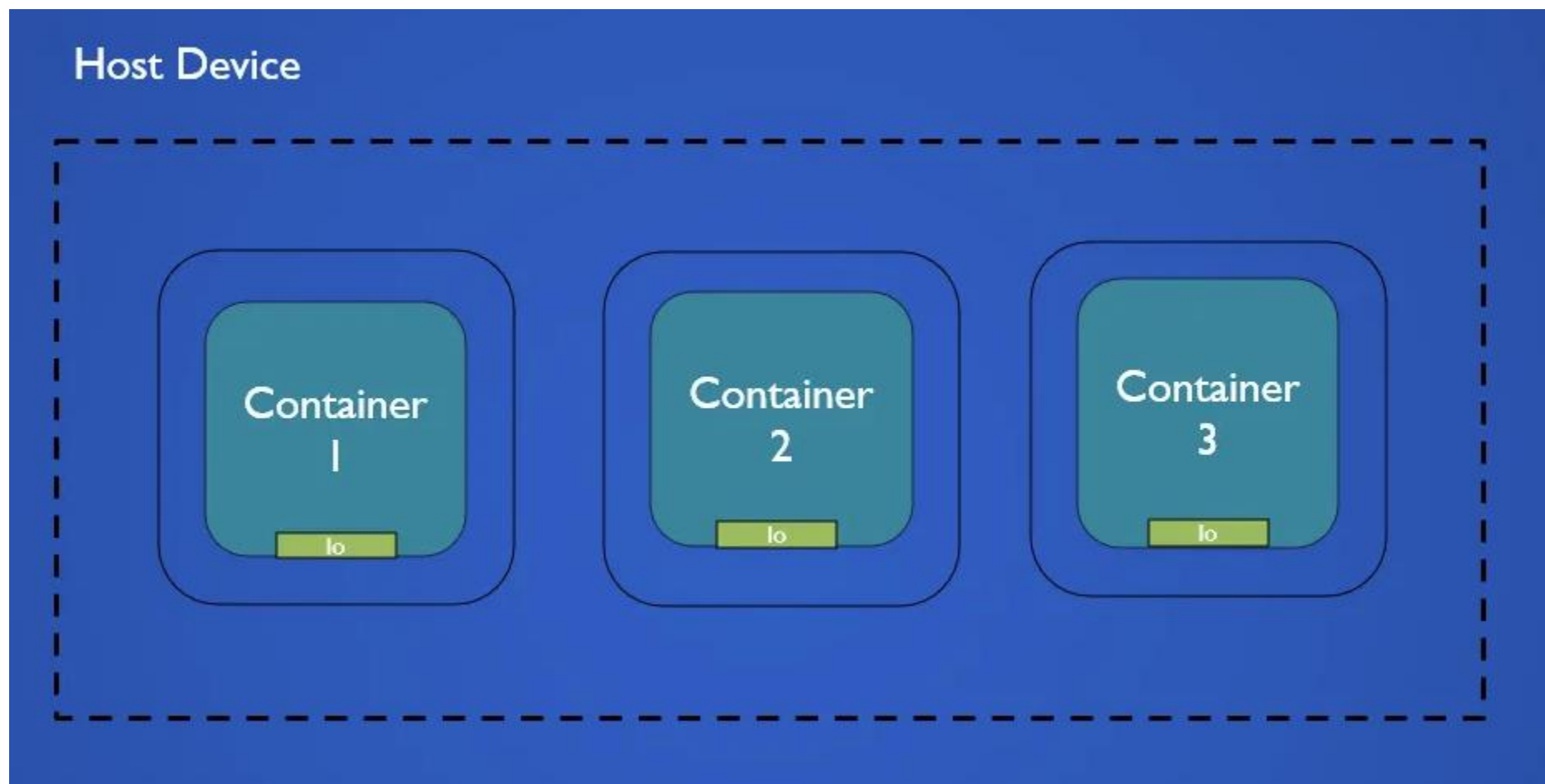
Le pilote none est utilisé lorsque nous souhaitons exécuter des conteneurs de manière complètement isolée les uns des autres et du monde extérieur.

Les conteneurs créés dans ce mode n'ont pas de pile réseau, ils contiennent uniquement une interface de bouclage.

Le conteneur n'a aucun moyen de se connecter à l'interface réseau du système hôte ou d'accéder à Internet.

Les principaux :

✓ none :



Gestion des networks

Lister les réseaux :

```
PS D:\Test_bind> docker network ls
```

| NETWORK ID | NAME | DRIVER | SCOPE |
|--------------|--------|--------|-------|
| c0cc9612cddd | bridge | bridge | local |
| 90edf8889d69 | host | host | local |
| 92407f652703 | none | null | local |

```
PS D:\Test_bind> |
```

Existents par défaut

Créer un nouveau réseau :

Un nouveau réseau peut être créé à l'aide de cette commande :

```
docker network create -d <driver> <network-name>|
```


L'indicateur -d est utilisé pour spécifier le pilote à utiliser pour créer le réseau (par défaut : bridge)

Créer un nouveau réseau :

```
PS D:\Test_bind> docker network create docker-101
f51e86cdafafb3ea1bbf83c71bc29cd82f79572e+3f351550f73ff8de247ff7
PS D:\Test_bind> docker network ls
```

| NETWORK ID | NAME | DRIVER | SCOPE |
|--------------|------------|--------|-------|
| c0cc9612cddd | bridge | bridge | local |
| f51e86cdafaf | docker-101 | bridge | local |
| 90edf8889d69 | host | host | local |
| 92407f652703 | none | null | local |

```
PS D:\Test_bind> |
```



Créer un nouveau réseau :

Lors de la création d'un réseau personnalisé :

- ✓ d'autres paramètres liés au réseau peuvent également être configurés.

Ces options sont plus avancées et ne sont pas requises pour la plupart des cas d'utilisation.

Créer un nouveau réseau :

Pour afficher ces options, la commande --help peut être utilisée

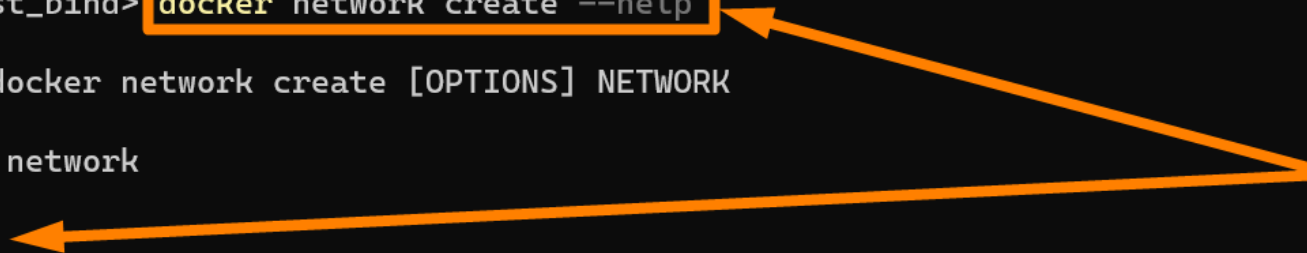
```
PS D:\Test_bind> docker network create --help
```

Usage: docker network create [OPTIONS] NETWORK

Create a network

Options:

| | |
|----------------------|---|
| --attachable | Enable manual container attachment |
| --aux-address map | Auxiliary IPv4 or IPv6 addresses used by Network driver (default map[]) |
| --config-from string | The network from which to copy the configuration |
| --config-only | Create a configuration only network |
| -d, --driver string | Driver to manage the Network (default "bridge") |
| --gateway strings | IPv4 or IPv6 Gateway for the master subnet |
| --ingress | Create swarm routing-mesh network |
| --internal | Restrict external access to the network |
| --ip-range strings | Allocate container ip from a sub-range |
| --ipam-driver string | IP Address Management Driver (default "default") |
| --ipam-opt map | Set IPAM driver specific options (default map[]) |
| --ipv6 | Enable IPv6 networking |
| --label list | Set metadata on a network |
| -o, --opt map | Set driver specific options (default map[]) |
| --scope string | Control the network's scope |
| --subnet strings | Subnet in CIDR format that represents a network segment |



Ajouter un conteneur au réseau :

Un conteneur peut être attaché à un réseau à l'aide de l'option `--net` dans la commande d'exécution du conteneur.

```
PS D:\Test_bind> docker run --rm --name alpine --net docker-101 -it alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c926b61bad3b: Pull complete
Digest: sha256:34871e7290500828b39e22294660bee86d966bc0017544e848dd9a255cdf59e0
Status: Downloaded newer image for alpine:latest
/ # |
```


Ajouter un conteneur au réseau :

```
Status: Downloaded newer image for alpine:latest
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
38: eth0@if39: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
/ #
```

Le conteneur s'est vu attribuer automatiquement une adresse IP 172.18.0.2 à l'aide du serveur DHCP du réseau Docker virtuel.

Ajouter un conteneur au réseau :

Nous pouvons confirmer ces informations en inspectant la configuration du container.

```
docker container inspect alpine
```

Résultat :

```
IPv4Gateway": "172.18.0.1",  
  "MacAddress": "",  
  "Networks": {  
    "docker-101": {  
      "IPAMConfig": null,  
      "Links": null,  
      "Aliases": [  
        "083d53fa22a8"  
      ],  
      "NetworkID": "f51e86cdafafbf3ea1bbf83c71bc29cd82f79572ef3f351550673ff8de247ff7",  
      "EndpointID": "3e88fca94db2b62b20a5931f2b5fc1b5f0f2b2d895254b13b0f8041fda9da892",  
      "Gateway": "172.18.0.1",  
      "IPAddress": "172.18.0.2",  
      "IPPrefixLen": 16,  
      "IPv6Gateway": "",  
      "GlobalIPv6Address": "",  
      "GlobalIPv6PrefixLen": 0,  
      "MacAddress": "02:42:ac:12:00:02",  
      "DriverOpts": null  
    }  
  }  
}
```

Communication entre conteneurs

Les conteneurs qui se trouvent sur le même réseau peuvent communiquer entre eux en utilisant leur nom d'hôte.

- ✓ Par défaut, le nom d'hôte est le nom attribué au conteneur lors de sa création.

En utilisant l'indicateur `--network-alias` :

- un nom d'hôte personnalisé différent du nom du conteneur peut être défini.

Créons deux conteneurs alpine :

Consignes : *création dans 2 terminaux différents.*

- ✓ tous deux connectés au réseau docker-101 que nous avons créé auparavant.
- ✓ Un conteneur s'appelle alpine
- ✓ et le deuxième conteneur s'appelle alpine2.

Alpine2 se voit également attribuer un nom d'hôte personnalisé alpin-host

Windows PowerShell

PS D:\Test_bind> docker run --rm --name alpine --net docker-101 -it alpine / #

PS C:\Users\conta> docker run --rm --name alpine2 --net docker-101 --network-alias alpine-host -it alpine / #

PS C:\Users\conta> docker ps -a

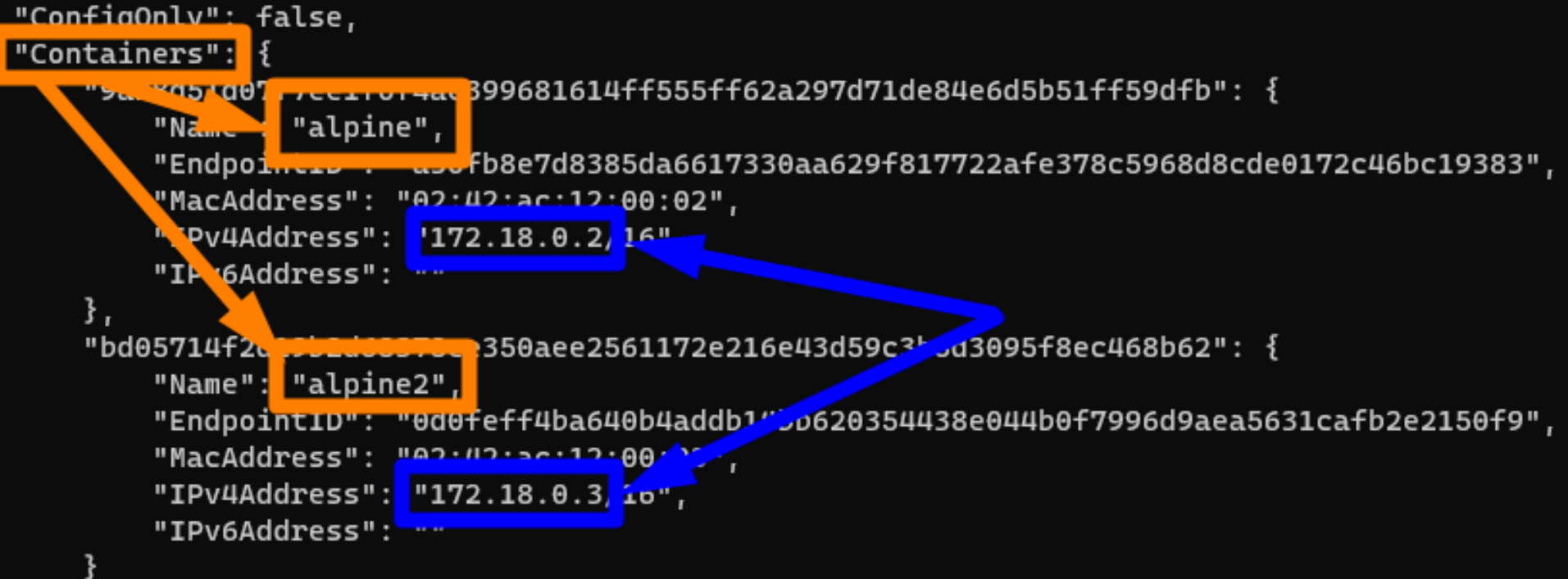
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------|-----------|--------------------|-------------------|-------|---------|
| bd05714f2d19 | alpine | "/bin/sh" | 14 seconds ago | Up 13 seconds | | alpine2 |
| 9a28d51d07f7 | alpine | "/bin/sh" | About a minute ago | Up About a minute | | alpine |

PS C:\Users\conta> |

Nous pouvons inspecter le réseau docker-101 pour voir qu'en effet les deux conteneurs l'utilisent :

```
docker network inspect docker-101
```

```
  "ConfigOnly": false,
  "Containers": {
    "9a8d51d0717cc17014ac399681614ff555ff62a297d71de84e6d5b51ff59dfb": {
      "Name": "alpine",
      "EndpointID": "450fb8e7d8385da6617330aa629f817722afe378c5968d8cde0172c46bc19383",
      "MacAddress": "02:42:ac:12:00:02",
      "IPv4Address": "172.18.0.2/16",
      "IPv6Address": ""
    },
    "bd05714f201b1d003701350aee2561172e216e43d59c315d3095f8ec468b62": {
      "Name": "alpine2",
      "EndpointID": "0d0feff4ba640b4addb115b620354438e044b0f7996d9aea5631cafb2e2150f9",
      "MacAddress": "02:42:ac:12:00:02",
      "IPv4Address": "172.18.0.3/16",
      "IPv6Address": ""
    }
  }
}
```



Depuis apline, lorsque nous pingons alpine-host :

✓ nous pouvons voir que alpine2 est accessible depuis alpine1.

Si nous faisons la même chose dans la direction opposée, cela fonctionne toujours.

```
PS D:\Test bind> docker run --rm --name alpine --net docker-101 -it alpine / # ping alpine-host -c 4
PING alpine-host (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.606 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.205 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.182 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.190 ms

--- alpine-host ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round trip min/avg/max = 0.182/0.255/0.606 ms
/ # |

PS C:\Users\conta> docker run --rm --name alpine2 --net docker-101 --network-alias alpine-host -it alpine / # ping alpine -c 4
PING alpine (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.151 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.117 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.182 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.224 ms

--- alpine ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
/ # |
```

Des résultats observés, nous pouvons conclure que les conteneurs qui s'exécutent sur le même réseau peuvent accéder les uns aux autres en utilisant leur nom d'hôte sans aucune configuration particulière.

**Connecter – déconnecter
un conteneur en cours d'exécution
au réseau**

Il est possible de connecter et déconnecter un conteneur d'un réseau à l'aide des commandes `network connect` et `network disconnect`.

Créons 3 conteneurs `alpine1`, `alpine2` et `alpine3` en connectant directement les 2 premiers sur notre réseau personnalisé `docker-101` :

```
PS D:\Test_bind> docker run -d --name alpine1 --net docker-101 -it alpine
0b7a2aff2bba69581602dec5b2f63986979de916084212e0e6df629e365f73b9
PS D:\Test_bind> docker run -d --name alpine2 --net docker-101 -it alpine
b12d76790f2f350c654b0a0b061b2e4202745130ce0e63c6bd6a6bbdc324a59c
PS D:\Test_bind> docker run -d --name alpine3 -it alpine
90f0543fd7697587f029eb1cb70b0fb15e25a9ed45b2e0e787d1f32c8084bc2a
PS D:\Test_bind> docker ps -a
```

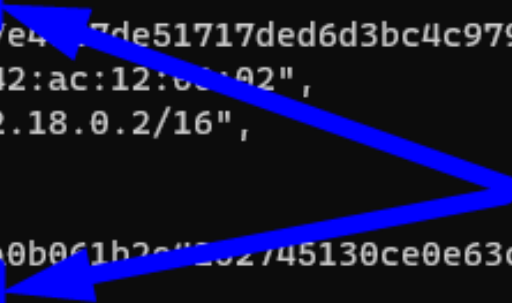
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------|-----------|----------------|---------------|-------|---------|
| 90f0543fd769 | alpine | "/bin/sh" | 7 seconds ago | Up 7 seconds | | alpine3 |
| b12d76790f2f | alpine | "/bin/sh" | 23 seconds ago | Up 23 seconds | | alpine2 |
| 0b7a2aff2bba | alpine | "/bin/sh" | 33 seconds ago | Up 32 seconds | | alpine1 |

```
PS D:\Test_bind> |
```

Inspectons notre réseau personnalisé :

```
docker network inspect docker-101
```

```
"ConfigOnly": false,  
"Containers": {  
  "0b7a2aff2bba69581602dec5b2f63986979de916084212e0e6df629e365f73b9": {  
    "Name": "alpine1",  
    "EndpointID": "9127e417de51717ded6d3bc4c9798938c0684995cfc39dd8f88a2fc735077d3",  
    "MacAddress": "02:42:ac:12:00:02",  
    "IPv4Address": "172.18.0.2/16",  
    "IPv6Address": ""  
  },  
  "b12d76790f2f350c650b0a0b0c1b2c252745130ce0e63c6bd6a6bbdc324a59c": {  
    "Name": "alpine2",  
    "EndpointID": "640ccc1f6eb9b0b32f9fa02c4e03737882cd29e87c777ba23c47fb23da32a9b6",  
    "MacAddress": "02:42:ac:12:00:03",  
    "IPv4Address": "172.18.0.3/16",  
    "IPv6Address": ""  
  }  
},  
"Options": {},
```



Connectons notre conteneur alpine3 à notre réseau personnalisé docker-101 :

```
docker network connect docker-101 alpine3
```

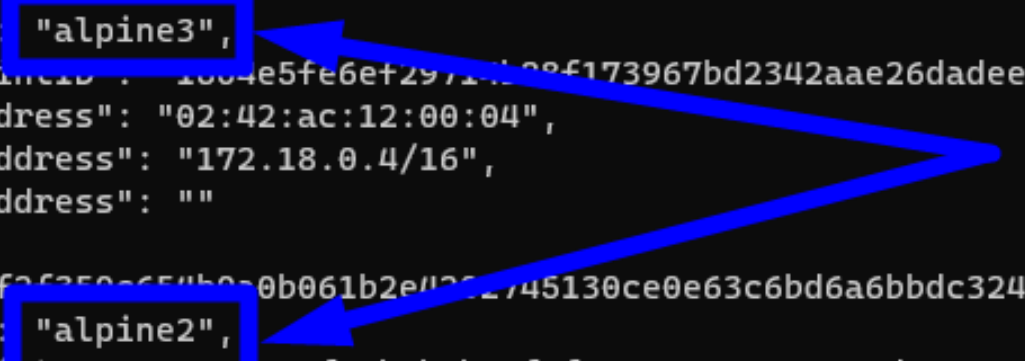
```
"Containers": {  
  "0b7a2aff2bba69581602dec5b2f63986979de916084212e0e6df629e365f73b9": {  
    "Name": "alpine1",  
    "EndpointID": "9127e4e87de32717ded6d3bc4c9798938c0684995cfc39dd8f88a2fc735077d3",  
    "MacAddress": "02:42:ac:12:00:02",  
    "IPv4Address": "172.18.0.2/16",  
    "IPv6Address": ""  
  },  
  "90f0543fd7697587f029eb1cb70b0fb15e25a9ed45b2e0e787d1f32c8084bc2a": {  
    "Name": "alpine3",  
    "EndpointID": "1604e5fe6ef29714b88f173967bd2342aae26dadee58518f5bf9bc5ce95caf3e",  
    "MacAddress": "02:42:ac:12:00:04",  
    "IPv4Address": "172.18.0.4/16",  
    "IPv6Address": ""  
  },  
  "b12d76790f173331854b019b061b5c4202745130ce0e63c6bd6a6bbdc324a59c": {  
    "Name": "alpine2",  
    "EndpointID": "0482cc1f6eb9b0b32f9fa02c4e03737882cd29e87c777ba23c47fb23da32a9b6",  
    "MacAddress": "02:42:ac:12:00:03",  
    "IPv4Address": "172.18.0.3/16",  
    "IPv6Address": ""  
  }  
},  
}
```

A diagram consisting of three blue arrows pointing from the container names 'alpine1', 'alpine3', and 'alpine2' to a single central point on the right side of the image. The arrows originate from the highlighted names in the JSON output and converge towards the center-right.

Déconnectons notre conteneur alpine1 de notre réseau personnalisé docker-101 :

```
docker network disconnect docker-101 alpine1
```

```
"ConfigOnly": false,  
"Containers": {  
  "90f0543fd76075875020eb1cb70b0fb15e25a9ed45b2e0e787d1f32c8084bc2a": {  
    "Name": "alpine3",  
    "EndpointID": "1884e5fe6e+2971108f173967bd2342aae26dadee583f8f5bf9bc5ce95caf3e",  
    "MacAddress": "02:42:ac:12:00:04",  
    "IPv4Address": "172.18.0.4/16",  
    "IPv6Address": ""  
  },  
  "b12d76790f26250265b070b061b2e4202745130ce0e63c6bd6a6bbdc324a59c": {  
    "Name": "alpine2",  
    "EndpointID": "080cc1f6eb9b0b32f9fa02c4e03737882cd29e87c777ba23c47fb23da32a9b6",  
    "MacAddress": "02:42:ac:12:00:03",  
    "IPv4Address": "172.18.0.3/16",  
    "IPv6Address": ""  
  }  
},  
"Options": {},
```



Supprimer un réseau

Une fois que nous avons fini d'utiliser un réseau, il peut être supprimé à l'aide de la commande :

```
PS D:\Test_bind> docker network rm docker-101
Error response from daemon: error while removing network: network docker-101 id f51e86cdafafbf3ea1bbf83c71bc29cd82f79572ef3f351550673ff8de247ff7 has active endpoints
```

Le réseau étant utilisé, la suppression est impossible.

```
"ConfigOnly": false,
"Containers": {
  "90f0543fd7607587f030eb1cb70b0fb15e25a9ed45b2e0e787d1f32c8084bc2a": {
    "Name": "alpine3",
    "EndpointID": "1884e5fe6e2971d1084f173967bd2342aae26dadee583f8f5bf9bc5ce95caf3e",
    "MacAddress": "02:42:ac:12:00:04",
    "IPv4Address": "172.18.0.4/16",
    "IPv6Address": ""
  },
  "b12d76790f2f350c65b070b061b2e0202745130ce0e63c6bd6a6bbdc324a59c": {
    "Name": "alpine2",
    "EndpointID": "3401cc1f6eb9b0b32f9fa02c4e03737882cd29e87c777ba23c47fb23da32a9b6",
    "MacAddress": "02:42:ac:12:00:03",
    "IPv4Address": "172.18.0.3/16",
    "IPv6Address": ""
  }
},
"Options": {},
```

Il faut d'abord vider le réseau :

```
PS D:\Test_bind> docker network disconnect docker-101 alpine2
PS D:\Test_bind> docker network disconnect docker-101 alpine3
```

Vérifions :

```
PS D:\Test_bind> docker network disconnect docker-101 alpine3
PS D:\Test_bind> docker network rm docker-101
docker-101
PS D:\Test_bind> docker network ls
NETWORK ID        Name               DRIVER            SCOPE
c0cc9612cddd      bridge            bridge            local
90edf8889d69      host              host              local
92407f652703      none             null              local
PS D:\Test_bind>
```

docker-101 à
disparu

Tips

Les réseaux personnalisés définis par l'utilisateur offrent une meilleure isolation.

- Tous les conteneurs sans `--network` spécifié sont attachés au bridge par défaut.

- ✓ Cela peut constituer un risque,

L'utilisation d'un réseau défini par l'utilisateur fournit un réseau étendu dans lequel seuls les conteneurs attachés à ce réseau peuvent communiquer.

- En raison des limitations définies par le noyau Linux :

les bridges deviennent instables et les communications entre conteneurs peuvent être interrompues lorsque 1 000 conteneurs ou plus se connectent à un seul réseau.

Questions ??

Prochain chapitre: Le Dockerfile.