Version : **2022.01**

Dernière mise-à-jour : 2022/04/29 07:57

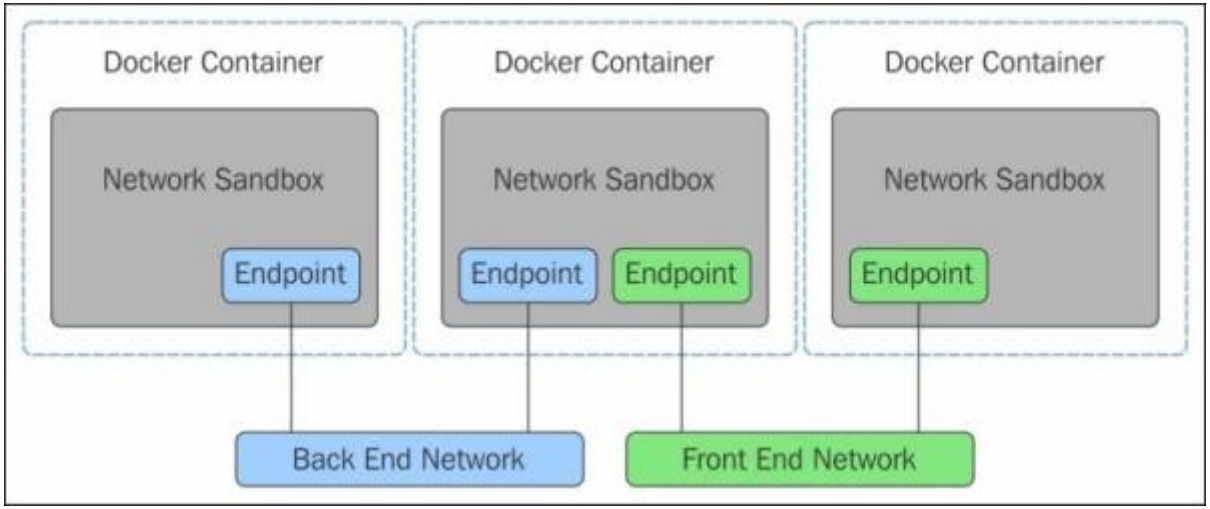
DOF203 - Gestion du Réseau avec Docker

Contenu du Module

L'Approche Réseau Docker

L'approche réseau de Docker est **libnetwork** qui implémente le **C**ontainer **N**etwork **M**odel (CNM). Dans ce modèle on trouve trois composants :

- Sandbox,
  - contient la configuration réseau du conteneur à savoir, la gestion des interfaces, la table de routage et le DNS,
- Endpoint,
  - relie un sandbox à un network,
- Network,
  - un groupe d'endpoints qui communiquent directement.



LAB #1 - Les Réseaux Docker ayant un Scope Local

Docker fournit trois réseaux par défaut :

```
root@debian9:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
495b3db75b0d        bridge              bridge              local
e1ed4de2f947        host                host                local
6bda460c97c6        none                null                local
```

1.1 - Bridge

Ce type de réseau est limité aux conteneurs d'un hôte unique exécutant Docker. Les conteneurs ne peuvent communiquer qu'entre eux et ils ne sont pas accessibles depuis l'extérieur. Pour que les conteneurs sur le réseau puissent communiquer ou être accessibles du monde extérieur, il faut configurer le mappage de port.

Par défaut Docker fonctionne en mode **Pont** ou (*Bridge*) et crée une interface intermédiaire à cet effet appelé **docker0** :

```
root@debian9:~# ip addr show docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:38:f1:e7:ee brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

Démarrez un conteneur dénommé **resotest** à partir d'une image de CentOS :

```
root@debian9:~# docker run -itd --name=resotest centos
2169360fcbfdbd6e68ea969a95edeb6fc42603c23ee42f03ceec286276519855
```

Lancez ensuite la commande **docker network inspect bridge** à partir de la machine virtuelle hôte de Debian_9 :

```
root@debian9:~# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "495b3db75b0d4bfcfc6da7c3e2af5f6addcdc227aa8b69b1e59a998be1819d12",
        "Created": "2017-09-07T07:44:49.942615596+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
```

```
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "2169360fcbfdbd6e68ea969a95edeb6fc42603c23ee42f03ceec286276519855": {
                "Name": "resotest",
                "EndpointID": "fc74e519d69b9a2112be959c92cda22b67671b52efbbd36fadf66097ccbb1271",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "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": {}
    }
]
```

Important : Notez ici que le conteneur **resotest** ne dispose pas de la même adresse que l'interface **docker0** de la machine hôte. Cependant les adresses se trouvent dans le même segment - **172.17.0.0/16** indiqué par la sortie **“Subnet”: “172.17.0.0/16”**.

Vous pouvez déconnecter un conteneur du réseau en utilisant la commande suivante :

```
root@debian9:~# docker network disconnect bridge resotest
root@debian9:~# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "495b3db75b0d4bfcfc6da7c3e2af5f6addcdc227aa8b69b1e59a998be1819d12",
        "Created": "2017-09-07T07:44:49.942615596+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
```

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

Créez maintenant votre propre réseau ponté appelé **my-bridged-network** :

```
root@debian9:~# docker network create -d bridge --subnet 172.25.0.0/16 --gateway 172.25.0.1 my-bridged-network
ceb7ba7493933c55d181bc92b1f799ca07bfe84b168d52a6ac648c1a906093f3
root@debian9:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
495b3db75b0d        bridge              bridge              local
e1ed4de2f947        host                host                local
ceb7ba749393        my-bridged-network  bridge              local
6bda460c97c6        none                null                local
```

Bien évidement, ce réseau est actuellement vide :

```
root@debian9:~# docker network inspect my-bridged-network
[
    {
        "Name": "my-bridged-network",
        "Id": "ceb7ba7493933c55d181bc92b1f799ca07bfe84b168d52a6ac648c1a906093f3",
        "Created": "2017-09-07T10:03:17.063730665+01:00",
```

```
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.25.0.0/16",
                    "Gateway": "172.25.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Lancez maintenant deux conteneurs et consultez les informations concernant le réseau :

```
root@debian9:~# docker run -itd --name=centos1 centos
9f36a628c72b383edfd4dc13ee4e4b2eaf5be0078d780f0334fcb8be0d977d0e

root@debian9:~# docker run -itd --name=centos2 centos
aaed3bc8e404ee1bccd6c87b39de32332940b5391514691fc70188edb17c1d7c
```

```
root@debian9:~# docker inspect --format='{{json .NetworkSettings.Networks}}'  centos1
{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"NetworkID":"495b3db75b0d4bfcfc6da7c3e2af5f6addcdc227aa8b69b1e59a998be1819d12","EndpointID":"d7b87875688b45258fc867b6bb8b0a0592f5c5fa16857fe136e55b87b6698219","Gateway":"172.17.0.1","IPAddress":"172.17.0.3","IPPrefixLen":16,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:11:00:03","DriverOpts":null}}

root@debian9:~# docker inspect --format='{{json .NetworkSettings.Networks}}'  centos2
{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"NetworkID":"495b3db75b0d4bfcfc6da7c3e2af5f6addcdc227aa8b69b1e59a998be1819d12","EndpointID":"2bfe090dccef89495d437d8deba5765996a917544ab7fde28ef5199f4e907eb1","Gateway":"172.17.0.1","IPAddress":"172.17.0.4","IPPrefixLen":16,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:11:00:04","DriverOpts":null}}

root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos1
172.17.0.3

root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos2
172.17.0.4
```

Mettez le conteneur **centos1** dans le réseau **my-bridged-network** :

```
root@debian9:~# docker network connect my-bridged-network centos1

root@debian9:~# docker network inspect my-bridged-network
[
    {
        "Name": "my-bridged-network",
        "Id": "ceb7ba7493933c55d181bc92b1f799ca07bfe84b168d52a6ac648c1a906093f3",
        "Created": "2017-09-07T10:03:17.063730665+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
```

```
            "Config": [
                {
                    "Subnet": "172.25.0.0/16",
                    "Gateway": "172.25.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9f36a628c72b383edfd4dc13ee4e4b2eaf5be0078d780f0334fcb8be0d977d0e": {
                "Name": "centos1",
                "EndpointID": "71e10e4e34ce8c42ef029e302f6ed372357f6fde8fd87fc2cbc1b14c2bdf6bb5",
                "MacAddress": "02:42:ac:19:00:02",
                "IPv4Address": "172.25.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos1
172.17.0.3172.25.0.2
```

Important : Notez que le conteneur **centos1** se trouve dans deux réseaux.

Faites la même chose pour le conteneur **centos2** :

```
root@debian9:~# docker network connect my-bridged-network centos2

root@debian9:~# docker network inspect my-bridged-network
[
    {
        "Name": "my-bridged-network",
        "Id": "ceb7ba7493933c55d181bc92b1f799ca07bfe84b168d52a6ac648c1a906093f3",
        "Created": "2017-09-07T10:03:17.063730665+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.25.0.0/16",
                    "Gateway": "172.25.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9f36a628c72b383edfd4dc13ee4e4b2eaf5be0078d780f0334fcb8be0d977d0e": {
                "Name": "centos1",
                "EndpointID": "71e10e4e34ce8c42ef029e302f6ed372357f6fde8fd87fc2cbc1b14c2bdf6bb5",
```

```
                "MacAddress": "02:42:ac:19:00:02",
                "IPv4Address": "172.25.0.2/16",
                "IPv6Address": ""
            },
            "aaed3bc8e404ee1bccd6c87b39de32332940b5391514691fc70188edb17c1d7c": {
                "Name": "centos2",
                "EndpointID": "34f533622f134b995097f1d3e6ce935158c1e5644201f896b42336738a81819c",
                "MacAddress": "02:42:ac:19:00:03",
                "IPv4Address": "172.25.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos2
172.17.0.4172.25.0.3
```

Connectez-vous au conteneur **centos1** en lançant bash :

```
root@debian9:~# docker exec -it centos1 bash
```

Vérifiez que la connectivité fonctionne :

```
[root@9f36a628c72b /]# ping 172.25.0.3
PING 172.25.0.3 (172.25.0.3) 56(84) bytes of data.
64 bytes from 172.25.0.3: icmp_seq=1 ttl=64 time=0.100 ms
64 bytes from 172.25.0.3: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 172.25.0.3: icmp_seq=3 ttl=64 time=0.050 ms
^C
--- 172.25.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
```

```
rtt min/avg/max/mdev = 0.050/0.066/0.100/0.025 ms
```

Les options possibles au niveau de la gestion du réseau sont vaste. Voici deux exemples supplémentaires.

Il est possible d'ajouter une adresse d'un serveur DNS au lancement d'un conteneur :

```
[root@9f36a628c72b /]# exit
exit
root@debian9:~# docker stop resotest
mongo2
root@debian9:~# docker rm resotest
mongo2
root@debian9:~# docker run -it --name=resotest --dns 8.8.8.8 centos bash
root@735599480b45:/# cat /etc/resolv.conf
search home
nameserver 8.8.8.8
root@735599480b45:/#
```

ou de passer une entrée pour le fichier **/etc/hosts** :

```
root@735599480b45:/# exit
exit
root@debian9:~# docker stop resotest
mongo2
root@debian9:~# docker rm resotest
mongo2
root@debian9:~# docker run -it --name=resotest --add-host mickeymouse:127.0.0.1 centos bash
root@718e7eab814f:/# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
127.0.0.1   mickeymouse
172.17.0.2  718e7eab814f
```

Liens

Le mécanisme des liens entre conteneurs est très puissant et permet d'atteindre un autre conteneur facilement à condition que les deux conteneurs soient dans le même réseau. Créez donc un conteneur dénommé **centos3** qui est lié au conteneur **centos2** qu'il connait aussi sous l'alias **alias** :

```
root@332aa9930f30:/# exit
exit

root@debian9:~# docker run -itd --name centos3 --link centos2:alias centos
6a315259b2946c3bf2bb69f608cbe910d87edaadedb4f805e7a4dbf6af1eb916

root@debian9:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
6a315259b294        centos              "/bin/bash"              33 seconds ago      Up 32 seconds
centos3
332aa9930f30        i2tch/mongodb2      "docker-entrypoint..."   3 minutes ago       Exited (127) 39 seconds ago
mongo2
aaed3bc8e404        centos              "/bin/bash"              16 minutes ago      Up 16 minutes
centos2
9f36a628c72b        centos              "/bin/bash"              16 minutes ago      Up 16 minutes
centos1
2169360fcbfd        centos              "/bin/bash"              20 minutes ago      Up 20 minutes
resotest
ea239635e141        testcache           "more /tmp/moment"       7 hours ago         Exited (0) 7 hours ago
test1
21b0490a93dd        i2tch/mydocker      "/entrypoint.sh my..."   7 hours ago         Exited (137) 6 hours ago
myDocker
bdb4bc0f81de        i2tch/mongodb1      "docker-entrypoint..."   18 hours ago        Created
27017/tcp           mongo1
```

```
f5b45072b831        i2tch/mongodb       "bash"                   19 hours ago        Exited (137) 6 hours ago
mongo
9731a48f126a        nginx               "nginx -g 'daemon ..."   19 hours ago        Exited (0) 6 hours ago
cocky_gates
eacd70596e23        nginx               "nginx -g 'daemon ..."   19 hours ago        Exited (0) 19 hours ago
adoring_yonath
cffb4456e9c4        ubuntu              "/bin/bash"              20 hours ago        Exited (0) 20 hours ago
i2tch

root@debian9:~# docker exec -it centos3 bash

[root@6a315259b294 /]# ping centos2
PING alias (172.17.0.4) 56(84) bytes of data.
64 bytes from alias (172.17.0.4): icmp_seq=1 ttl=64 time=0.116 ms
64 bytes from alias (172.17.0.4): icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from alias (172.17.0.4): icmp_seq=3 ttl=64 time=0.068 ms
64 bytes from alias (172.17.0.4): icmp_seq=4 ttl=64 time=0.070 ms
^C
--- alias ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.068/0.080/0.116/0.023 ms

[root@6a315259b294 /]# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.4  alias aaed3bc8e404 centos2
172.17.0.2  6a315259b294

[root@6a315259b294 /]# exit
exit
```

```
root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos3
172.17.0.2
```

Notez cependant qu le lien est unidirectionnel :

```
root@debian9:~# docker exec -it centos2 bash

[root@aaed3bc8e404 /]# ping centos3
ping: centos3: Name or service not known

[root@aaed3bc8e404 /]# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.071 ms
^C
--- 172.17.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.035/0.052/0.071/0.015 ms

[root@aaed3bc8e404 /]#
```

Dans le cas ci-dessus, **centos2** peut atteindre **centos3** en utilisant l'adresse IP **172.17.0.2** car **centos2** se trouve dans les deux réseaux avec les adresses IP 172.17.0.4 et 172.25.0.3 :

```
[root@aaed3bc8e404 /]# exit
exit
root@debian9:~# docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' centos2
172.17.0.4172.25.0.3
```

1.2 - Host

Ce type de réseau est utilisé dans le cas où le réseau ne doit pas être isolé de l'hôte tout en isolant les autres aspects du conteneur. Les conteneurs utilisent la même interface que l'hôte en prenant la même adresse IP que la machine hôte.

Dans le cas de la machine virtuelle, l'adresse IP de l'interface connectée au réseau local est **10.0.2.60** :

```
root@debian9:~# ip addr show ens18
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:2e:77:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.60/24 brd 10.0.2.255 scope global dynamic ens18
       valid_lft 83772sec preferred_lft 83772sec
    inet6 fe80::a00:27ff:fe2e:7701/64 scope link
       valid_lft forever preferred_lft forever
```

Démarrez un conteneur à partir de l'image **centos** dans un réseau de type **host** :

```
root@debian9:~# docker run -it --rm --network host --name centos3 centos bash
[root@debian9 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:2e:77:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.60/24 brd 10.0.2.255 scope global dynamic ens18
       valid_lft 82102sec preferred_lft 82102sec
    inet6 fe80::a00:27ff:fe2e:7701/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:38:f1:e7:ee brd ff:ff:ff:ff:ff:ff
```

```
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:38ff:fef1:e7ee/64 scope link
       valid_lft forever preferred_lft forever
[root@debian9 /]# hostname
debian9
[root@debian9 /]# exit
```

Le but de ce type de réseau est de permettre l'accès à des services dans le conteneur directement à partir de l'adresse IP de l'hôte Docker. Par exemple, un nginx dans le conteneur pourrait être joint directement sur 10.0.2.60:80 **sans** avoir besoin de passer par l'exposition du port.

Pour cette raison, dans le cas de l'option **-p** utilisé dans la cas du réseau **host**, cette option n'est pas prise en compte et produit l'avertissement **WARNING: Published ports are discarded when using host network mode**. L'utilité majeure donc du réseau **host** se trouve dans le cas où de multiples ports dans le conteneur doivent être joignables.

Important : Notez que le réseau de type **host** ne fonctionne que sous Linux. Il est donc incompatible avec Docker Desktop pour Mac, Docker Desktop pour Windows et Docker EE pour Windows Server.

1.3 - None

Ce type de réseau est utilisé principalement dans le cas de l'utilisation d'un plugin réseau disponible dans le **Docker Hub**.

Il est donc possible de lancer un conteneur totalement étanche grâce au réseau **none** :

```
root@718e7eab814f:/# exit
exit
root@debian9:~# docker stop mongo2
mongo2
root@debian9:~# docker rm mongo2
```

```
mongo2
root@debian9:~# docker run -it --name mongo2 --network none i2tch/mongodb2 bash
root@332aa9930f30:/#
```

1.4 - Lancer Wordpress dans un container

Créez le répertoire ~/wordpress et placez-vous dedans :

```
root@debian9:~# mkdir ~/wordpress && cd ~/wordpress
```

Créez un conteneur dénommé **wordpressdb** à partir de l'image **mariadb:latest** :

```
root@debian9:~/wordpress# docker run -e MYSQL_ROOT_PASSWORD=fenestros -e MYSQL_DATABASE=wordpress --name
wordpressdb -v "$PWD/database":/var/lib/mysql -d mariadb:latest
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
f2b6b4884fc8: Pull complete
26d8bdca4f3e: Pull complete
74f09e820cce: Pull complete
5390f1fe4554: Pull complete
3d3f1706a741: Pull complete
2942f66426ea: Pull complete
97ee11d39c75: Pull complete
590c46ef722b: Pull complete
32eb4b9666e5: Pull complete
fc883f98a064: Pull complete
bb8bee61bc1e: Pull complete
Digest: sha256:6135f5b851e7fe263dcf0edf3480cdab1ab28c4287e867c5d83fbe967412ea14
Status: Downloaded newer image for mariadb:latest
67831dacf002bdc21dc79b0e8483f538235d00ddd2e8aae175ef3ebf189ae14d
```

Vérifiez que le conteneur fonctionne :

```
root@debian9:~/wordpress# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS
NAMES
67831dacf002        mariadb:latest      "docker-entrypoint.s…"   About a minute ago   Up 45 seconds
3306/tcp            wordpressdb
```

Créez un conteneur appellé **wordpress** lié au conteneur wordpressdb :

```
root@debian9:~/wordpress# docker run -e WORDPRESS_DB_USER=root -e WORDPRESS_DB_PASSWORD=fenestros --name
wordpress --link wordpressdb:mysql -p 10.0.2.60:80:80 -v "$PWD/html":/var/www/html -d wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
2a72cbf407d6: Pull complete
273cd543cb15: Pull complete
ec5ac8875de7: Pull complete
9106e19b56c1: Pull complete
ee2f70ac7c7d: Pull complete
7257ad6985e8: Pull complete
18f5c2055da2: Pull complete
85293a6fdd80: Pull complete
9e797eeb0c14: Pull complete
f16178842884: Pull complete
13899c06d3f8: Pull complete
70c27fe4c3c5: Pull complete
d32c8ad2d9d7: Pull complete
07fe445494e6: Pull complete
63b8de7b32fe: Pull complete
e4b721952e22: Pull complete
d9ede6dd6f74: Pull complete
0af4f74bfd92: Pull complete
e4e7c47b969f: Pull complete
69aff47f3112: Pull complete
Digest: sha256:201d004f55669dd2c0884f00fc44145fb0da8cafa465bf22cbaacecaf81138d4
Status: Downloaded newer image for wordpress:latest
```

```
9eb2f7fbfbd25307ed2f463c7eb3bef40bfa556174e68750bb76b8d032546129
```

Vérifiez que le conteneur fonctionne :

```
root@debian9:~/wordpress# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
9eb2f7fbfbd2        wordpress           "docker-entrypoint.s…"   2 minutes ago       Up About a minute
10.0.2.60:80->80/tcp   wordpress
67831dacf002        mariadb:latest      "docker-entrypoint.s…"   9 minutes ago       Up 8 minutes        3306/tcp
wordpressdb
```

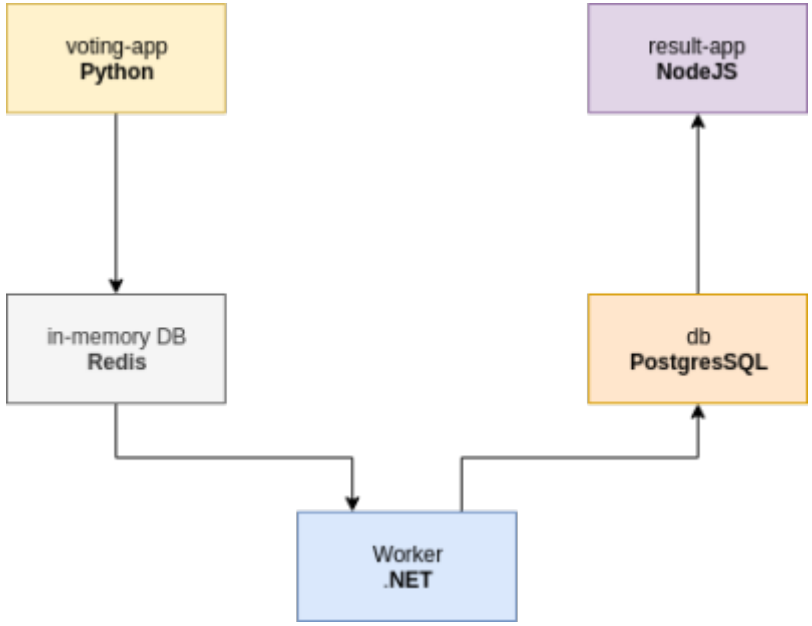
Vérifiez que le Wordpress fonctionne :

```
root@debian9:~/wordpress# lynx --dump http://10.0.2.60
   [1]WordPress
   Select a default language [English (United States)________]

   Continue

References

   1. https://wordpress.org/

root@debian9:~/wordpress# docker inspect wordpress | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.3",
                    "IPAddress": "172.17.0.3",
root@debian9:~/wordpress# lynx --dump http://172.17.0.3
   [1]WordPress
   Select a default language [English (United States)________]

   Continue
```

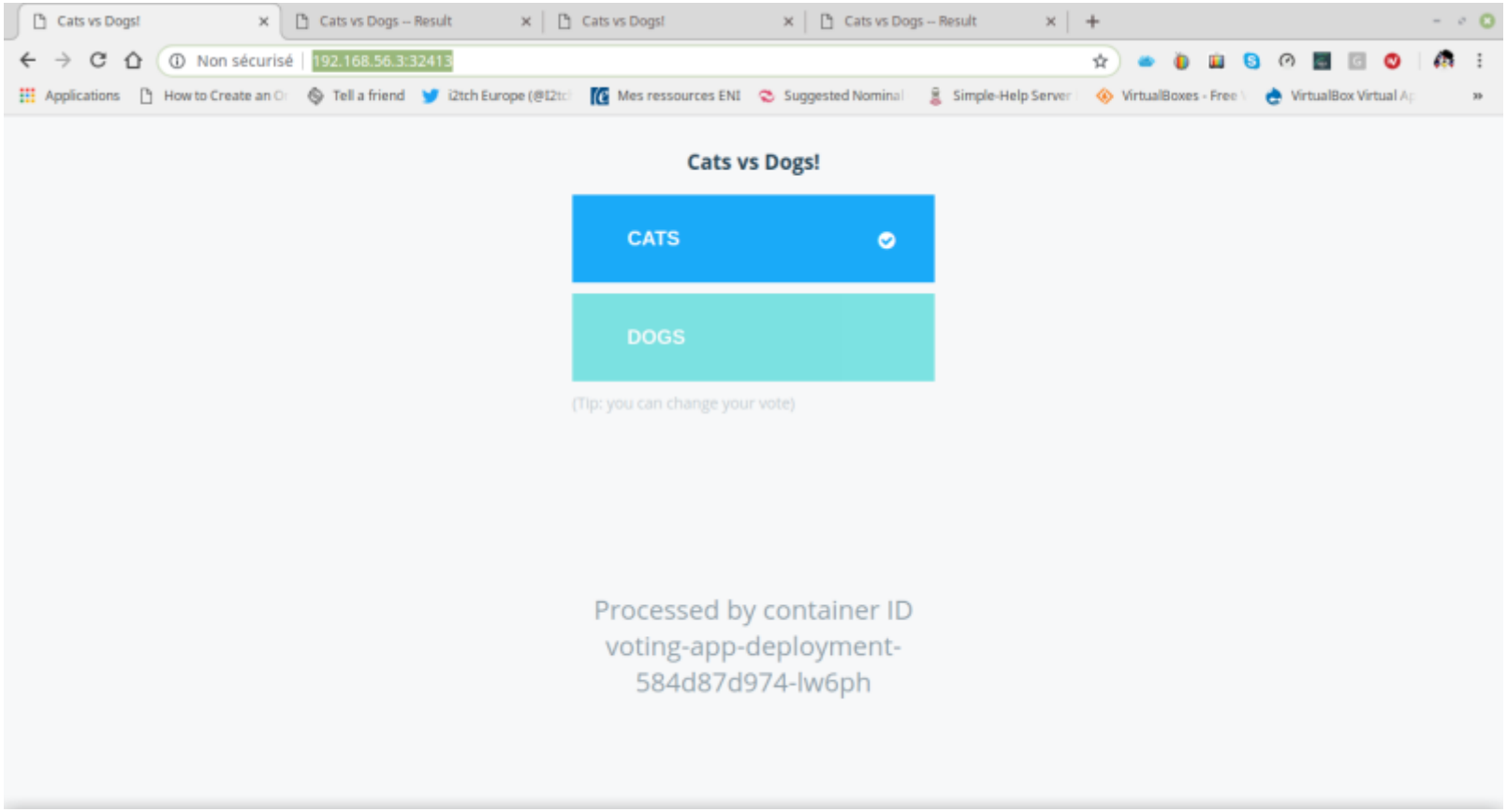
```
References

   1. https://wordpress.org/
```

1.5 - Gestion d'une Architecture de Microservices

Vous allez mettre en place une application simple sous forme de microservices, développé par Docker et appelé **demo-voting-app**, :

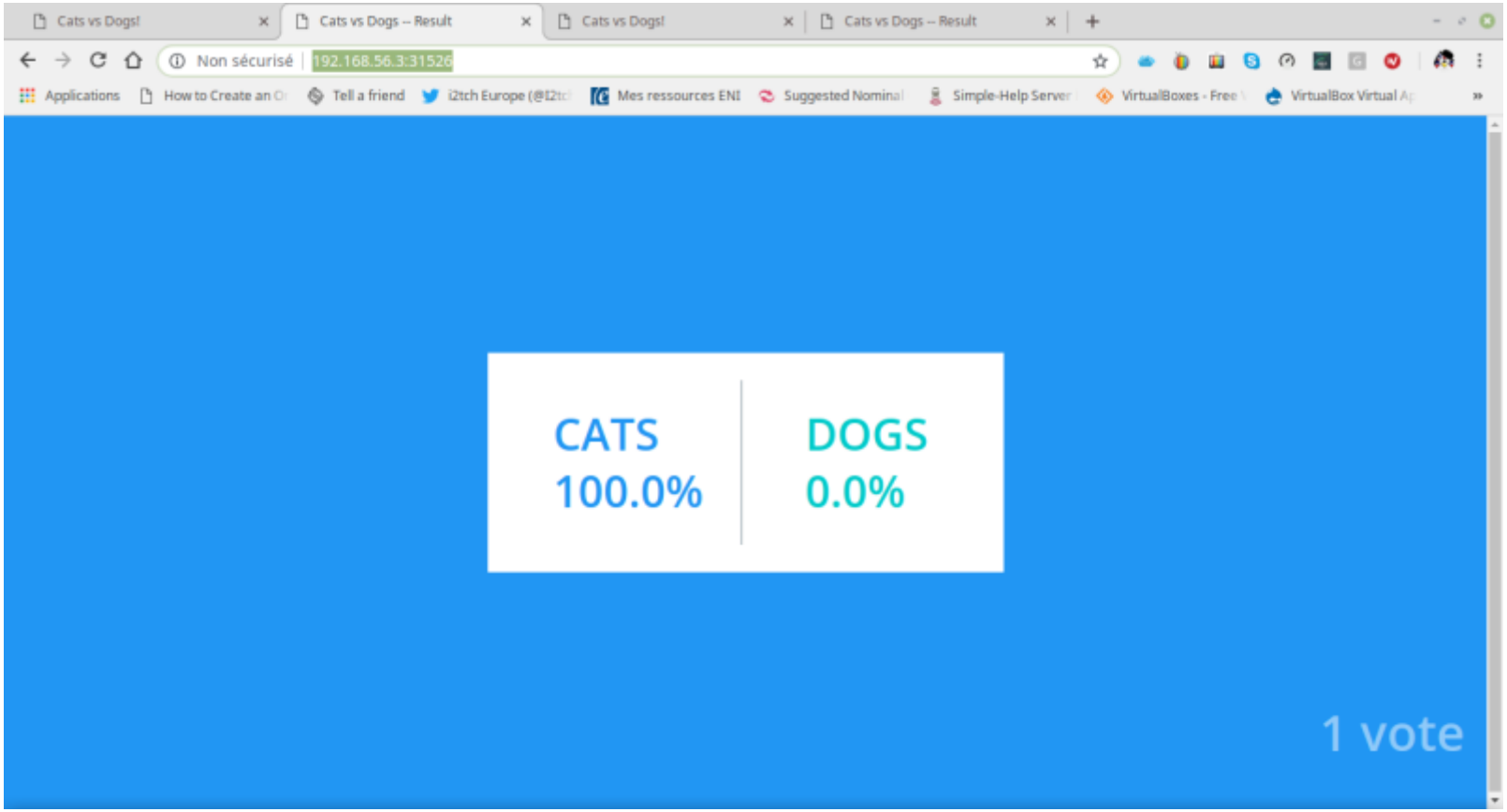


Dans cette application le conteneur **voting-app** permet de voter pour des **chats** ou des **chiens**. Cette application tourne sous Python et fournit une interface HTML :



Lors de la vote, le résultat de celle-ci est stocké dans **Redis** dans une base de données en mémoire. Le résultat est ensuite passé au conteneur **Worker** qui tourne sous .NET et qui met à jour la base de données persistante dans le conteneur **db** qui tourne sous PostgreSQL.

L'application **result-app** qui tourne sous NodeJS lit ensuite la table dans la base de données PostgreSQL et affiche le résultat sous forme HTML :



Cette application peut être mise en place sous docker avec les commandes suivantes :

```
docker run -d --name=redis redis
docker run -d --name=db -e POSTGRES_PASSWORD=postgres -e POSTGRES_USER=postgres postgres:9.4
docker run -d --name=vote -p 5000:80 --link redis:redis dockersamples/examplevotingapp_vote
docker run -d --name=result -p 5001:80 --link db:db dockersamples/examplevotingapp_result
docker run -d --name=worker --link db:db --link redis:redis dockersamples/examplevotingapp_worker
```

Cette solution utilise un réseau de type Bridge. Ce type de réseau est limité aux conteneurs d'un hôte unique exécutant Docker. Les conteneurs ne peuvent communiquer qu'entre eux et ils ne sont pas accessibles depuis l'extérieur. Pour que les conteneurs sur le réseau puissent communiquer ou être accessibles du monde extérieur, il faut configurer le mappage de port.

Ouvrez le navigateur web **Firefox** ou **Chrome** dans **votre** machine et saisissez l'URL selon le tableau ci-dessous :

ID	URL (Notez http: et non https:)
Trainee10	http://compute01.ittraining.network
Trainee11	http://compute02.ittraining.network
Trainee12	http://compute03.ittraining.network
Trainee13	http://compute04.ittraining.network
Trainee14	http://compute05.ittraining.network
Trainee15	http://compute06.ittraining.network
Trainee16	http://compute07.ittraining.network
Trainee17	http://compute08.ittraining.network
Trainee18	http://compute09.ittraining.network
Trainee19	http://compute10.ittraining.network
Trainee20	http://compute01.ittraining.network
Trainee21	http://compute02.ittraining.network
Trainee22	http://compute03.ittraining.network
Trainee23	http://compute04.ittraining.network
Trainee24	http://compute05.ittraining.network
Trainee25	http://compute06.ittraining.network
Trainee26	http://compute07.ittraining.network
Trainee27	http://compute08.ittraining.network
Trainee28	http://compute09.ittraining.network
Trainee29	http://compute10.ittraining.network

Dans la boîte de connexion d'Apache Guacamole, entrez votre ID **traineeXX** et le mot de passe qui vous a été fourni par votre formateur.

Cliquez sur la connexion **TraineeXX_VNC** et testez ensuite votre application en utilisant le navigateur web de la machine virtuelle.

LAB #2 - Gestion du Réseau overlay

En plus des réseaux **bridge**, **host** et **none**, Docker propose deux autres types de réseaux, à savoir **overlay** et **macvlan**. Ce module concerne overlay. Pour plus d'informations concernant le type **macvlan**, consultez le site de la documentation de Docker **ici**.

Comme son nom indique, un réseau overlay est un réseau qui se positionne au-dessus du réseau des hôtes. Lors de la création d'un réseau overlay, celui-ci n'est disponible par défaut qu'aux services swarm. Par contre il est possible de connecter des conteneurs autonomes au réseau overlay si l'option **–attachable** est spécifiée lors de sa création. Ce type d'utilisation du réseau overlay n'est pas recommandé par Docker qui dit que le support de cette fonctionnalité pourrait être retiré.

Le trafic lié à la gestion des services swarm est crypté par défaut avec l'algorithme AES en mode GCM. Afin de crypter le trafic des données liées aux applications il est possible d'utiliser l'option **–opt encrypted** lors de la création du réseau overlay. Dans ce cas, Docker crée des tunnels IPSEC entre chaque nœud qui utilise le même algorithme que le trafic des services swarm. Il y a donc une dégradation des performances à évaluer avant la mise en production. Dans les deux cas les clefs sont modifiées toutes les 12 heures (voir https://www.vaultproject.io/docs/internals/rotation.html)

ATTENTION : Le cryptage des données liées aux applications n'est pas compatible avec Windows™. Lors de la connexion du nœud Windows™ à un réseau overlay crypté, aucune erreur ne sera rapportée. Par contre le nœud sera incapable de communiquer.

Commencez par re-créer un swarm en utilisant les machines virtuelles **manager**, **worker1** et **worker2** :

```
trainee@traineeXX:~$ ssh -l trainee 10.0.2.62
...
root@manager:~# docker swarm leave
Node left the swarm.
root@manager:~# docker swarm init --advertise-addr 10.0.2.62
Swarm initialized: current node (tpn1zsk20sfsfafmk2cvefqjc) is now a manager.

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

    docker swarm join --token SWMTKN-1-23d7n1fkkk9rvlhty106q9390bfpf9daljjguq3s807le6c5qs-
e0s1yqsajvmi7s8t9l9mw48ao 10.0.2.62:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

```
root@manager:~# exit
trainee@manager:~# exit
```

Connectez-vous au **worker1** :

```
trainee@traineeXX:~$ ssh -l trainee 10.0.2.63
...
root@worker1:~# docker swarm leave
Node left the swarm.
root@worker1:~# docker swarm join --token SWMTKN-1-23d7n1fkkk9rvlhty106q9390bfpf9daljjguq3s807le6c5qs-
e0s1yqsajvmi7s8t9l9mw48ao 10.0.2.62:2377
This node joined a swarm as a worker.
root@worker1:~# exit
trainee@worker1:~# exit
```

Connectez-vous au **worker2** :

```
trainee@traineeXX:~$ ssh -l trainee 10.0.2.64
...
root@worker2:~# docker swarm leave
Node left the swarm.
root@worker2:~# docker swarm join --token SWMTKN-1-23d7n1fkkk9rvlhty106q9390bfpf9daljjguq3s807le6c5qs-
e0s1yqsajvmi7s8t9l9mw48ao 10.0.2.62:2377
This node joined a swarm as a worker.
root@worker2:~# exit
trainee@worker2:~# exit
```

Vérifiez l'état du swarm :

```
trainee@traineeXX:~$ ssh -l trainee 10.0.2.62
...
root@manager:~# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
ENGINE VERSION
```

```
b85hxlixbr1mh1txd1hrfe4us *   manager.i2tch.loc   Ready               Active              Leader
19.03.4
4sui75vvdhmet4qvt0zbvzlzl     worker1.i2tch.loc   Ready               Active
19.03.4
lbjtg5o9kw3x6xg7frm07jfuw     worker2.i2tch.loc   Ready               Active
19.03.4
root@manager:~# docker node ls --filter role=manager
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
ENGINE VERSION
b85hxlixbr1mh1txd1hrfe4us *   manager.i2tch.loc   Ready               Active              Leader
19.03.4
root@manager:~# docker node ls --filter role=worker
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
ENGINE VERSION
4sui75vvdhmet4qvt0zbvzlzl     worker1.i2tch.loc   Ready               Active
19.03.4
lbjtg5o9kw3x6xg7frm07jfuw     worker2.i2tch.loc   Ready               Active
19.03.4
```

Vérifiez la présence du réseau overlay **ingress** ainsi que le réseau ponté **docker_gwbridge** :

```
root@manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
4edb7186dcc9        bridge              bridge              local
d4c9b0c9437a        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
de563e30d473        none                null                local
```

Info : Le réseau **docker_gwbridge** relie le réseau **ingress** à l'adaptateur réseau de l'hôte et par conséquent relie le démon Docker aux autres démons Docker qui participent dans swarm.

Best Practice : Docker recommande l'utilisation de réseaux de type overlay différents pour chaque application ou groupe d'applications.

2.2 - Création d'un Réseau overlay

A partir du Manager, créez un réseau de type overlay appelé **nginx-net** :

```
root@manager:~# docker network create -d overlay nginx-net
j57jhtug4kjxp22ai1y664lqr
root@manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
dde514eea83f        bridge              bridge              local
d4c9b0c9437a        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
j57jhtug4kjx        nginx-net           overlay             swarm
de563e30d473        none                null                local
```

2.2 - Création d'un Service

Créez un service nginx qui utilise le réseau **nginx-net** :

```
root@manager:~# docker service create --name my-nginx --publish target=80,published=80 --replicas=5 --network
nginx-net nginx
fpydgix3e1rc1qum72gvwcb7f
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
```

```
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
```

Info : Le service publie le port 80 qui est visible de l'extérieur. Les conteneurs communiquent entre eux sans ouvrir de ports supplémentaires.

Vérifiez que le service fonctionne avant de poursuivre :

```
root@manager:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
fpydgix3e1rc        my-nginx            replicated          5/5                 nginx:latest        *:80->80/tcp
```

Consultez maintenant les détails du service :

```
root@manager:~# docker service inspect my-nginx
[
    {
        "ID": "fpydgix3e1rc1qum72gvwcb7f",
        "Version": {
            "Index": 40
        },
        "CreatedAt": "2019-10-28T06:23:29.17883246Z",
        "UpdatedAt": "2019-10-28T06:23:29.183438696Z",
        "Spec": {
            "Name": "my-nginx",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image":
```

```
"nginx:latest@sha256:922c815aa4df050d4df476e92daed4231f466acc8ee90e0e774951b0fd7195a4",
                    "Init": false,
                    "StopGracePeriod": 10000000000,
                    "DNSConfig": {},
                    "Isolation": "default"
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                    "Condition": "any",
                    "Delay": 5000000000,
                    "MaxAttempts": 0
                },
                "Placement": {
                    "Platforms": [
                        {
                            "Architecture": "amd64",
                            "OS": "linux"
                        },
                        {
                            "OS": "linux"
                        },
                        {
                            "Architecture": "arm64",
                            "OS": "linux"
                        },
                        {
                            "Architecture": "386",
                            "OS": "linux"
                        },
                        {
                            "Architecture": "ppc64le",
```

```
                        "OS": "linux"
                    },
                    {
                        "Architecture": "s390x",
                        "OS": "linux"
                    }
                ]
            },
            "Networks": [
                {
                    "Target": "j57jhtug4kjxp22ai1y664lqr"
                }
            ],
            "ForceUpdate": 0,
            "Runtime": "container"
        },
        "Mode": {
            "Replicated": {
                "Replicas": 5
            }
        },
        "UpdateConfig": {
            "Parallelism": 1,
            "FailureAction": "pause",
            "Monitor": 5000000000,
            "MaxFailureRatio": 0,
            "Order": "stop-first"
        },
        "RollbackConfig": {
            "Parallelism": 1,
            "FailureAction": "pause",
            "Monitor": 5000000000,
            "MaxFailureRatio": 0,
            "Order": "stop-first"
```

```
            },
            "EndpointSpec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishedPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            }
        },
        "Endpoint": {
            "Spec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishedPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            },
            "Ports": [
                {
                    "Protocol": "tcp",
                    "TargetPort": 80,
                    "PublishedPort": 80,
                    "PublishMode": "ingress"
                }
            ],
            "VirtualIPs": [
```

```
                {
                    "NetworkID": "r8htcvc8oxmzy896xvwvv87k5",
                    "Addr": "10.255.0.5/16"
                },
                {
                    "NetworkID": "j57jhtug4kjxp22ai1y664lqr",
                    "Addr": "10.0.0.2/24"
                }
            ]
        }
    }
]
```

Important : Notez ici les informations concernant les ports et les Endpoints utilisés par le service.

2.3 - Déplacer le Service vers un autre Réseau overlay

Consultez le réseau overlay **nginx-net** sur les trois nœuds :

```
root@manager:~# docker inspect nginx-net
[
    {
        "Name": "nginx-net",
        "Id": "j57jhtug4kjxp22ai1y664lqr",
        "Created": "2019-10-28T07:23:29.492986337+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
```

```
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "b2e882e530b10f8fd0b2481f851007f864ce1495bc9fdedcf51a475c0fc03aeb": {
                "Name": "my-nginx.2.bo4q3us1f6m0uwxhqgtau1yg5",
                "EndpointID": "f6f82bcb81ba82191f3988702b0e91f7f5f139c5c88899ad7c95e12ab189e055",
                "MacAddress": "02:42:0a:00:00:04",
                "IPv4Address": "10.0.0.4/24",
                "IPv6Address": ""
            },
            "c0a76b54dad58b0faf80d2f915a10072aa7d726c46036caa3157d22c30dba843": {
                "Name": "my-nginx.4.aqj5vafpqtkc8f4rn4v04x4kn",
                "EndpointID": "813bef65edc4de42d5ec4357013f5b711cd21ce7d1a1c8361c1d989d0d709071",
                "MacAddress": "02:42:0a:00:00:06",
                "IPv4Address": "10.0.0.6/24",
                "IPv6Address": ""
            },
            "lb-nginx-net": {
                "Name": "nginx-net-endpoint",
                "EndpointID": "d087f5fe91481b12ca0b966d01584d143b25c746952bb517441cfad6beba90de",
```

```
                "MacAddress": "02:42:0a:00:00:08",
                "IPv4Address": "10.0.0.8/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "1199cab4a6dd",
                "IP": "10.0.2.62"
            },
            {
                "Name": "69676ae46ab9",
                "IP": "10.0.2.63"
            },
            {
                "Name": "d058d363197d",
                "IP": "10.0.2.64"
            }
        ]
    }
]
```

```
root@worker1:~# docker inspect nginx-net
[
    {
        "Name": "nginx-net",
        "Id": "j57jhtug4kjxp22ai1y664lqr",
        "Created": "2019-10-28T07:23:29.561068917+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
```

```
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "50b205e2ed4ccaaad5adc06c508af235557c89c116c819e367a1d925e9c2b564": {
                "Name": "my-nginx.1.gcz867ezj0y46tsdgoz8j3jz2",
                "EndpointID": "a48a43da98acef2748f42ffa992ba302863ed3c417fa3289cbd3aed0e33e97fa",
                "MacAddress": "02:42:0a:00:00:03",
                "IPv4Address": "10.0.0.3/24",
                "IPv6Address": ""
            },
            "lb-nginx-net": {
                "Name": "nginx-net-endpoint",
                "EndpointID": "54ed15511cdd574cb60d37d39257cbf7b30331b24bb069aadb33b457b2864789",
                "MacAddress": "02:42:0a:00:00:0a",
                "IPv4Address": "10.0.0.10/24",
                "IPv6Address": ""
            }
        },
```

```
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "69676ae46ab9",
                "IP": "10.0.2.63"
            },
            {
                "Name": "d058d363197d",
                "IP": "10.0.2.64"
            },
            {
                "Name": "1199cab4a6dd",
                "IP": "10.0.2.62"
            }
        ]
    }
]
```

```
root@worker2:~# docker inspect nginx-net
[
    {
        "Name": "nginx-net",
        "Id": "j57jhtug4kjxp22ai1y664lqr",
        "Created": "2019-10-28T07:23:29.562818383+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
```

```
            {
                "Subnet": "10.0.0.0/24",
                "Gateway": "10.0.0.1"
            }
        ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "31bcb5e553886cd9b3a6b8e70fe0c2bed92fe081bd0def0c94864631a940cbd6": {
            "Name": "my-nginx.5.t3be85jtp2qlhpmvsl4866s5m",
            "EndpointID": "ffa92f5f3bb7fd2665a8be336ef1e4e2d786790852eb152dac1a2c45f18518ba",
            "MacAddress": "02:42:0a:00:00:07",
            "IPv4Address": "10.0.0.7/24",
            "IPv6Address": ""
        },
        "8e2ce40a6e0d9fb2bc64c264b92164b6ea241a2369d8e6844d00b8952f5729a7": {
            "Name": "my-nginx.3.dma616z2rkbted13zd824fyo2",
            "EndpointID": "99cfb31ce34ccd9b6b15f71c87eddb5f39a84512ec76d215d54bdaaf851d5129",
            "MacAddress": "02:42:0a:00:00:05",
            "IPv4Address": "10.0.0.5/24",
            "IPv6Address": ""
        },
        "lb-nginx-net": {
            "Name": "nginx-net-endpoint",
            "EndpointID": "c0816f6f1e5c046ac1deb8163c5a8cf40765a126bf76b6f10bf1bb708a51dfa1",
            "MacAddress": "02:42:0a:00:00:09",
            "IPv4Address": "10.0.0.9/24",
            "IPv6Address": ""
```

```
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "d058d363197d",
                "IP": "10.0.2.64"
            },
            {
                "Name": "69676ae46ab9",
                "IP": "10.0.2.63"
            },
            {
                "Name": "1199cab4a6dd",
                "IP": "10.0.2.62"
            }
        ]
    }
]
```

Important : Notez que le réseau **nginx-net** a été créé automatiquement sur les deux Workers. Notez aussi le contenu de la section **Peers** qui liste les nœuds ainsi que la section **Containers** qui liste les conteneurs sur chaque nœud qui sont connectés au réseau overlay.

Créez maintenant un deuxième réseau de type overlay, appelé **nginx-net-2** :

```
root@manager:~# docker network create -d overlay nginx-net-2
```

```
aez5huut9hd472qmldzf2tsud
```

Déplacez le service **my-nginx** vers le réseau **nginx-net-2** :

```
root@manager:~# docker service update --network-add nginx-net-2 --network-rm nginx-net my-nginx
my-nginx
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
```

Vérifiez que le service fonctionne avant de poursuivre :

```
root@manager:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
fpydgix3e1rc        my-nginx            replicated          5/5                 nginx:latest        *:80->80/tcp
```

Vérifiez qu'aucun conteneur se trouve dans le réseau **nginx-net** :

```
root@manager:~# docker network inspect nginx-net
[
    {
        "Name": "nginx-net",
        "Id": "j57jhtug4kjxp22ai1y664lqr",
        "Created": "2019-10-28T06:21:18.337578134Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
```

```
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": null,
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": null
    }
]
```

Vérifiez maintenant que les conteneurs se trouvent dans le réseau **nginx-net-2** :

```
root@manager:~# docker network inspect nginx-net-2
[
    {
        "Name": "nginx-net-2",
        "Id": "aez5huut9hd472qmldzf2tsud",
        "Created": "2019-10-28T10:09:54.465105557+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
```

```
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.1.0/24",
                    "Gateway": "10.0.1.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0bf159064e30d5e788a12baca53ee8e9504a2d7300017fb268cb9e90caaea27a": {
                "Name": "my-nginx.2.81pveqac42zesvuulpbiho7k6",
                "EndpointID": "25c9587e76cfca10d17b10fa967186bc73ca6b444cc2689e43a7243f5d1795b2",
                "MacAddress": "02:42:0a:00:01:05",
                "IPv4Address": "10.0.1.5/24",
                "IPv6Address": ""
            },
            "74e656da8c670fca23270078565af164c4d42415f012ff51ccb02395c6d121e9": {
                "Name": "my-nginx.3.mjj1bsguaaewk61dw7yxxjdlu",
                "EndpointID": "2be3c3e0286d3afb5ba47bbd903151a4d337a45743cb30c46595160223e02fba",
                "MacAddress": "02:42:0a:00:01:07",
                "IPv4Address": "10.0.1.7/24",
                "IPv6Address": ""
            },
            "lb-nginx-net-2": {
                "Name": "nginx-net-2-endpoint",
                "EndpointID": "768a4cc926b5c94a20904e5db500dc62b40a063077a49769ccccc007a6cb61ac",
```

```
                "MacAddress": "02:42:0a:00:01:06",
                "IPv4Address": "10.0.1.6/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4098"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "69676ae46ab9",
                "IP": "10.0.2.63"
            },
            {
                "Name": "1199cab4a6dd",
                "IP": "10.0.2.62"
            },
            {
                "Name": "d058d363197d",
                "IP": "10.0.2.64"
            }
        ]
    }
]
```

Supprimez maintenant le service **my-nginx** ainsi que les deux réseaux overlay **nginx-net** et **nginx-net-2** :

```
root@manager:~# docker service rm my-nginx
my-nginx
root@manager:~# docker network rm nginx-net nginx-net-2
nginx-net
nginx-net-2
```

2.4 - DNS container discovery

Le daemon Docker exécute un server DNS embarqué à l'adresse 127.0.0.11 qui permet la résolution des noms dans un réseau personnalisé. Si ce serveur est incapable de faire la résolution, il transfert la requête à tout serveur externe défini dans le conteneur.

Pour que le **DNS container discovery** fonctionne, les ports suivants doivent être ouverts sur les nœuds :

- 2377/tcp
- 7946/tcp
- 7946/udp
- 4789/udp

Créez maintenant le réseau de type overlay **test-net** :

```
root@manager:~# docker network create --driver=overlay --attachable test-net
hrs25w4l951kkickhj6262mjg
```

Important : Notez que le **NETWORK-ID** ici est **hrs25w4l951kkickhj6262mjg**.

Sur le Manager, démarrez un conteneur interactif appelé **alpine1** et qui se connecte au réseau **test-net** :

```
root@manager:~# docker run -it --name alpine1 --network test-net alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
89d9c30c1d48: Pull complete
Digest: sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2bd5a
Status: Downloaded newer image for alpine:latest
/ #
```

Listez les réseaux disponibles sur **Worker1** :

```
root@worker1:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
3fe43b514f9d        bridge              bridge              local
ee22b3e623ca        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que le réseau **test-net** n'a pas été créé.

Démarrez maintenant un conteneur **alpine2** sur **Worker1** :

```
root@worker1:~# docker run -dit --name alpine2 --network test-net alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
89d9c30c1d48: Pull complete
Digest: sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2bd5a
Status: Downloaded newer image for alpine:latest
5734e84cd460cdd33ce90970d98a96837a0305832a86fc4d86be38aecf51b23b
```

Saisissez la commande **docker network ls** sur **Worker1** :

```
root@worker1:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
3fe43b514f9d        bridge              bridge              local
ee22b3e623ca        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
de563e30d473        none                null                local
hrs25w4l951k        test-net            overlay             swarm
```

Important : Notez que le réseau **test-net**, ayant le même **NETWORK ID**, a été automatiquement créé lors de la création du conteneur **alpine2**.

Listez les réseaux disponibles sur **Worker2** :

```
root@worker2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
ff7308310f60        bridge              bridge              local
0ce1d8369c29        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que le réseau **test-net** n'a pas été créé.

Attachez vous au conteneur **alpine2** sur **Worker1** et essayez de contacter le conteneur **alpine1** :

```
root@worker1:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
ce9097b864dc        alpine              "/bin/sh"           23 minutes ago      Up 23 minutes
alpine2
root@worker1:~# docker attach alpine2
/ # ping -c 2 alpine1
PING alpine1 (10.0.2.2): 56 data bytes
64 bytes from 10.0.2.2: seq=0 ttl=64 time=1.874 ms
64 bytes from 10.0.2.2: seq=1 ttl=64 time=1.669 ms
```

```
--- alpine1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.669/1.771/1.874 ms
/ #
```

Retournez dans la VM **Manager** et essayez de contacter le conteneur **alpine2** à partir du conteneur **alpine1** :

```
root@manager:~# docker attach alpine1
/ # ping -c 2 alpine2
PING alpine2 (10.0.0.4): 56 data bytes
64 bytes from 10.0.0.4: seq=0 ttl=64 time=0.666 ms
64 bytes from 10.0.0.4: seq=1 ttl=64 time=1.239 ms

--- alpine2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.666/0.952/1.239 ms
/ #
```

Créez ensuite le conteneur **alpine3** sur le **Worker2** essayez de contacter le conteneur **alpine1** :

```
root@worker2:~# docker run -it --rm --name alpine3 --network test-net alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c9b1b535fdd9: Pull complete
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for alpine:latest
/ # ping -c 2 alpine1
PING alpine1 (10.0.2.2): 56 data bytes
64 bytes from 10.0.2.2: seq=0 ttl=64 time=0.642 ms
64 bytes from 10.0.2.2: seq=1 ttl=64 time=1.684 ms

--- alpine1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.642/1.163/1.684 ms
```

```
/ # exit
```

Arrêtez maintenant le conteneur **alpine2** sur **Worker1** :

```
root@worker1:~# docker container stop alpine2
alpine2
```

Saisissez la commande **docker network ls** :

```
root@worker1:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
3bb80f391804        bridge              bridge              local
ee22b3e623ca        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
r8htcvc8oxmz        ingress             overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que le réseau **test-net** a été supprimé.

Supprimez le conteneur **alpine2**:

```
root@worker1:~# docker container rm alpine2
alpine2
```

Arrêtez le conteneur **alpine1** et supprimez le réseau **test-net** sur **Manager** :

```
/ # exit
root@manager:~# docker container stop alpine1
alpine1
root@manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
```

```
a604e7db6f95        bridge              bridge              local
d4c9b0c9437a        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
jxu667wzmj2u        ingress             overlay             swarm
de563e30d473        none                null                local
518l09lcjhsp        test-net            overlay             swarm
root@manager:~# docker network rm test-net
test-net
```

2.5 - Création d'un Réseau overlay Personnalisé

Il est possible de créer un réseau overlay personnalisé. Dans ce cas là, il convient de supprimer le réseau ingress déjà existant :

```
root@manager:~# docker network rm ingress
WARNING! Before removing the routing-mesh network, make sure all the nodes in your swarm run the same docker
engine version. Otherwise, removal may not be effective and functionality of newly create ingress networks will
be impaired.
Are you sure you want to continue? [y/N] y
ingress
```

Créez ensuite votre réseau personnalisé :

```
root@manager:~# docker network create --driver overlay --ingress --subnet=10.11.0.0/16 --gateway=10.11.0.2 --opt
com.docker.network.driver.mtu=1200 my-ingress
44ozn3vtg23zkksrvloxuulcl
root@manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
24be8a0f0ef5        bridge              bridge              local
d4c9b0c9437a        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
44ozn3vtg23z        my-ingress          overlay             swarm
de563e30d473        none                null                local
```

Créez de nouveau le service **my-nginx** :

```
root@manager:~# docker service create --name my-nginx --publish target=80,published=80 --replicas=5 nginx
gp1iozmbi25dx3skn00m6suoz
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged

root@manager:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
gp1iozmbi25d        my-nginx            replicated          5/5                 nginx:latest        *:80->80/tcp

root@manager:~# docker service ps my-nginx
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
upmbwmtr76cm        my-nginx.1          nginx:latest        worker1.i2tch.loc   Running             Running about
a minute ago
qz6p1li7zmef        my-nginx.2          nginx:latest        worker2.i2tch.loc   Running             Running about
a minute ago
me50mkhd11yk        my-nginx.3          nginx:latest        manager.i2tch.loc   Running             Running about
a minute ago
sctjud70ihkl        my-nginx.4          nginx:latest        worker1.i2tch.loc   Running             Running about
a minute ago
kql9qx3phb73        my-nginx.5          nginx:latest        worker2.i2tch.loc   Running             Running about
a minute ago
```

Consultez les informations concernant le service **my-nginx** :

```
root@manager:~# docker service inspect my-nginx
[
```

```
    {
        "ID": "gp1iozmbi25dx3skn00m6suoz",
        "Version": {
            "Index": 230
        },
        "CreatedAt": "2019-10-28T14:49:33.6719228Z",
        "UpdatedAt": "2019-10-28T14:49:33.679624758Z",
        "Spec": {
            "Name": "my-nginx",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image":
"nginx:latest@sha256:922c815aa4df050d4df476e92daed4231f466acc8ee90e0e774951b0fd7195a4",
                    "Init": false,
                    "StopGracePeriod": 10000000000,
                    "DNSConfig": {},
                    "Isolation": "default"
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                    "Condition": "any",
                    "Delay": 5000000000,
                    "MaxAttempts": 0
                },
                "Placement": {
                    "Platforms": [
                        {
                            "Architecture": "amd64",
                            "OS": "linux"
                        },
```

```
                    {
                        "OS": "linux"
                    },
                    {
                        "Architecture": "arm64",
                        "OS": "linux"
                    },
                    {
                        "Architecture": "386",
                        "OS": "linux"
                    },
                    {
                        "Architecture": "ppc64le",
                        "OS": "linux"
                    },
                    {
                        "Architecture": "s390x",
                        "OS": "linux"
                    }
                ]
            },
            "ForceUpdate": 0,
            "Runtime": "container"
        },
        "Mode": {
            "Replicated": {
                "Replicas": 5
            }
        },
        "UpdateConfig": {
            "Parallelism": 1,
            "FailureAction": "pause",
            "Monitor": 5000000000,
            "MaxFailureRatio": 0,
```

```
                "Order": "stop-first"
            },
            "RollbackConfig": {
                "Parallelism": 1,
                "FailureAction": "pause",
                "Monitor": 5000000000,
                "MaxFailureRatio": 0,
                "Order": "stop-first"
            },
            "EndpointSpec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishedPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            }
        },
        "Endpoint": {
            "Spec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishedPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            },
            "Ports": [
```

```
                {
                    "Protocol": "tcp",
                    "TargetPort": 80,
                    "PublishedPort": 80,
                    "PublishMode": "ingress"
                }
            ],
            "VirtualIPs": [
                {
                    "NetworkID": "44ozn3vtg23zkksrvloxuulcl",
                    "Addr": "10.11.0.1/16"
                }
            ]
        }
    }
]
```

Vérifiez maintenant que les conteneurs se trouvent dans le réseau **my-ingress** :

```
root@manager:~# docker inspect my-ingress
[
    {
        "Name": "my-ingress",
        "Id": "l11ucu5ufjfwwz6e0umtygdqy",
        "Created": "2020-03-10T11:02:38.278429829+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.11.0.0/16",
```

```
                "Gateway": "10.11.0.2"
            }
        ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": true,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "6f0168ff5153b899af31098740de34997b12417ef7c0f3824938edf79b2bca7f": {
            "Name": "my-nginx.3.me50mkhd11ykwz7aj07znloh1",
            "EndpointID": "41531d43496f4723cb62cad1d57c5a088faebe79c430d04a1765022e31d8ae17",
            "MacAddress": "02:42:0a:0b:00:05",
            "IPv4Address": "10.11.0.5/16",
            "IPv6Address": ""
        },
        "my-ingress-sbox": {
            "Name": "my-ingress-endpoint",
            "EndpointID": "0205796eeb005ef77b3ea382fd1e72c312a58fd717b5a79ca6cacc7e090068e6",
            "MacAddress": "02:42:0a:0b:00:0a",
            "IPv4Address": "10.11.0.10/16",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.driver.mtu": "1200",
        "com.docker.network.driver.overlay.vxlanid_list": "4100"
    },
    "Labels": {},
    "Peers": [
        {
```

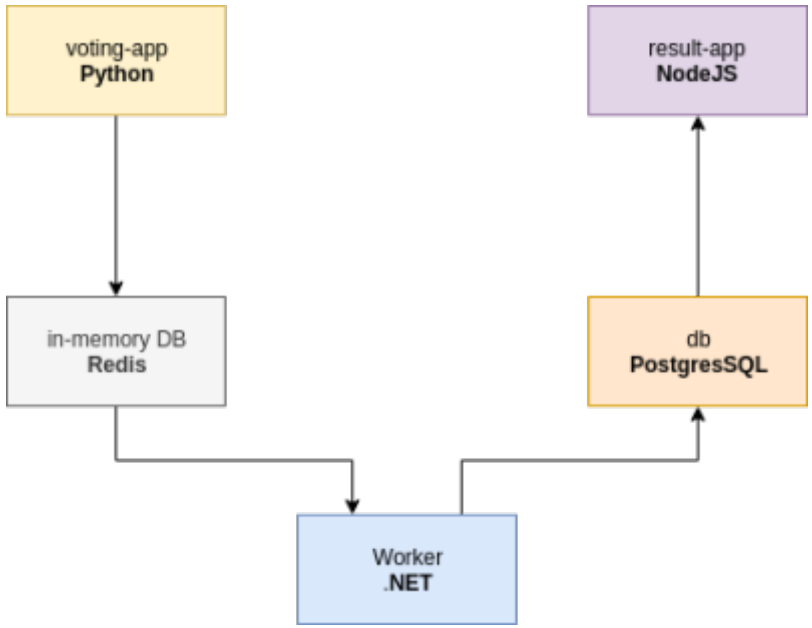
```
                "Name": "9a00e8bc72fe",
                "IP": "10.0.2.62"
            },
            {
                "Name": "3ea669d48ca2",
                "IP": "10.0.2.64"
            },
            {
                "Name": "f30e39df1704",
                "IP": "10.0.2.63"
            }
        ]
    }
]
```

Supprimez maintenant le service **my-nginx** :

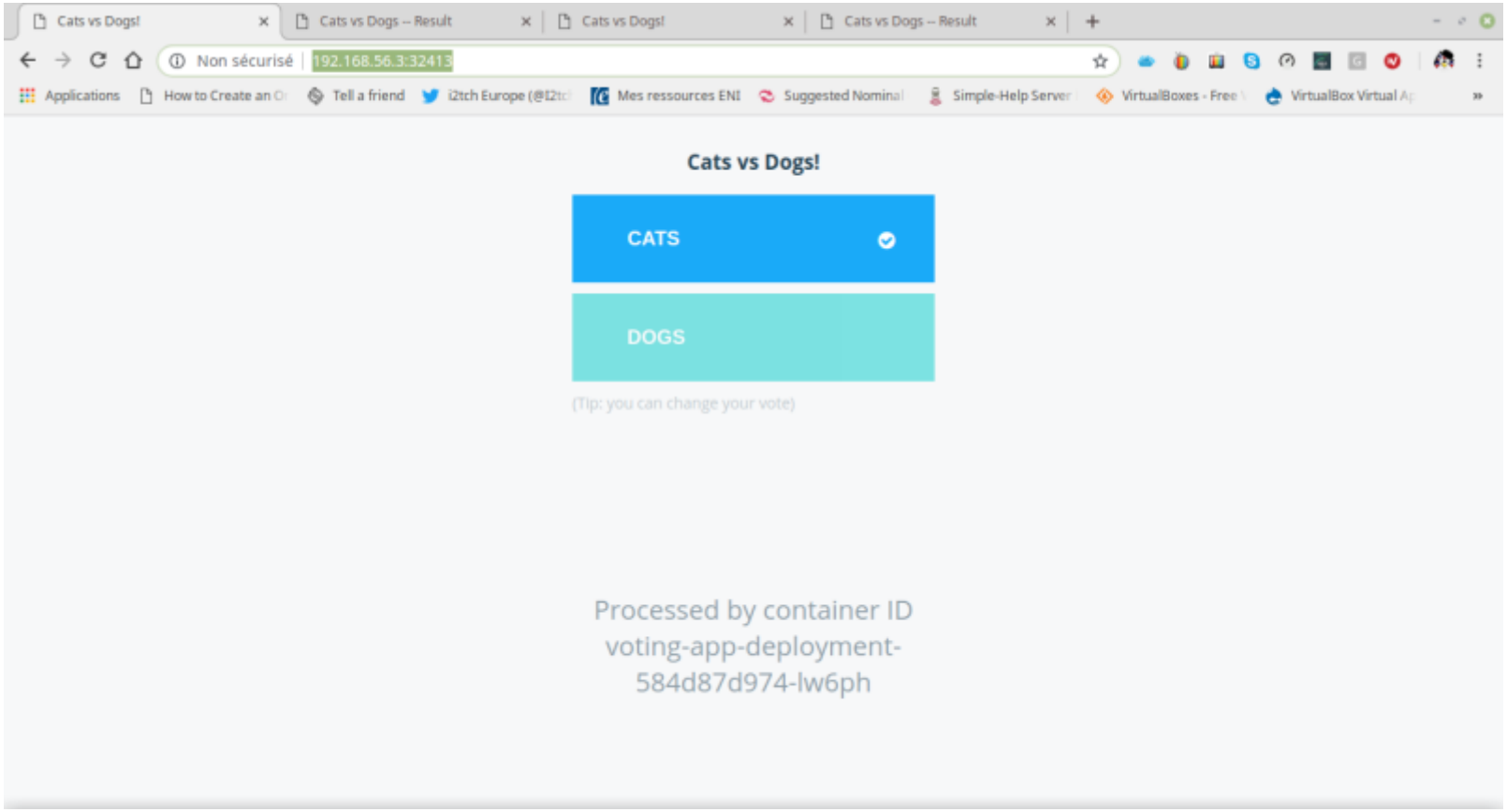
```
root@manager:~# docker service rm my-nginx
my-nginx
```

LAB #3 - Gestion d'une Architecture de Microservices

Vous allez mettre en place une application simple, appelé **demo-voting-app** et développé par Docker, sous forme de microservices :

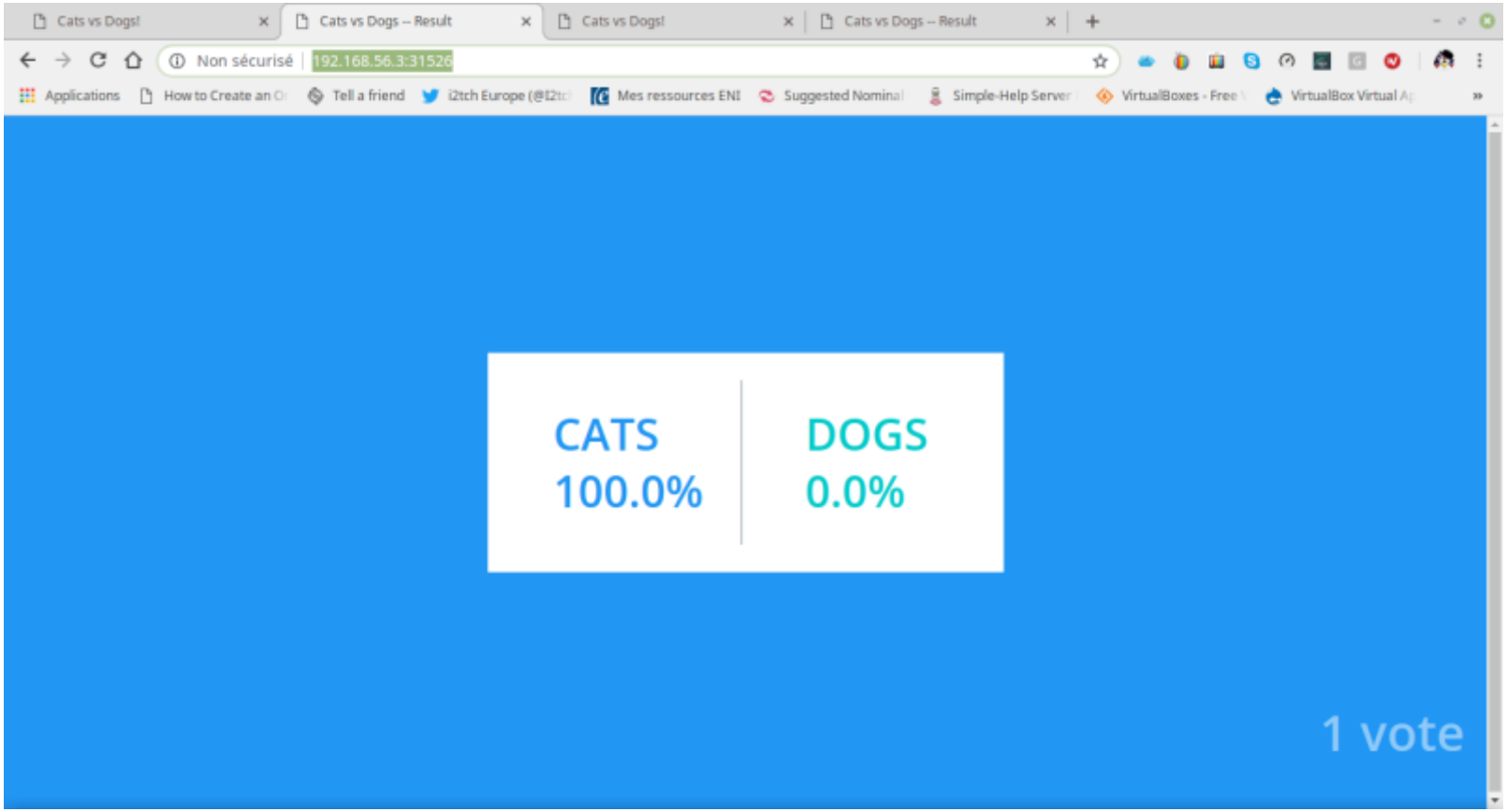


Dans cette application le conteneur **voting-app** permet de voter pour des **chats** ou des **chiens**. Cette application tourne sous Python et fournit une interface HTML :



Lors de la vote, le résultat de celle-ci est stocké dans **Redis** dans une base de données en mémoire. Le résultat est ensuite passé au conteneur **Worker** qui tourne sous .NET et qui met à jour la base de données persistante dans le conteneur **db** qui tourne sous PostgreSQL.

L'application **result-app** qui tourne sous NodeJS lit ensuite la table dans la base de données PostgreSQL et affiche le résultat sous forme HTML :



3.1 - Mise en Place avec Docker Swarm avec des réseaux Overlay

Cette application peut être mise en place sous docker swarm avec avec la commande **docker stack**. Un **stack** est un groupe de services.
Premièrement, vérifiez l'état du Swarm :

```
root@manager:~# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
ENGINE VERSION
b85hxlixbr1mh1txd1hrfe4us *   manager.i2tch.loc   Ready               Active              Leader
19.03.4
```

```
4sui75vvdhmet4qvt0zbvzlzl     worker1.i2tch.loc   Ready               Active
19.03.4
lbjtg5o9kw3x6xg7frm07jfuw     worker2.i2tch.loc   Ready               Active
19.03.4
```

Téléchargez maintenant le fichier **docker-stack.yml** :

```
root@manager:~# curl -O https://raw.githubusercontent.com/docker/example-voting-app/master/docker-stack.yml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1707  100  1707    0     0   2030      0 --:--:-- --:--:-- --:--:--  2029
```

Consultez le fichier téléchargé :

```
root@manager:~# cat docker-stack.yml
version: "3"
services:

  redis:
    image: redis:alpine
    networks:
      - frontend
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
```

```
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
```

```
    restart_policy:
      condition: on-failure

worker:
  image: dockersamples/examplevotingapp_worker
  networks:
    - frontend
    - backend
  depends_on:
    - db
    - redis
  deploy:
    mode: replicated
    replicas: 1
    labels: [APP=VOTING]
    restart_policy:
      condition: on-failure
      delay: 10s
      max_attempts: 3
      window: 120s
    placement:
      constraints: [node.role == manager]

visualizer:
  image: dockersamples/visualizer:stable
  ports:
    - "8080:8080"
  stop_grace_period: 1m30s
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
  deploy:
    placement:
      constraints: [node.role == manager]
```

```
networks:
  frontend:
  backend:

volumes:
  db-data:
```

Dans ce fichier on peut constater 6 services, **redis**, **db**, **vote**, **result**, **worker** et **visualizer**. Les 5 premiers services forment ensemble l'application tandis que le service **visualizer** nous permettra de voir comment l'application a été mise en place.

Dans un premier temps, regardez la clef **deploy** du service **worker** :

```
...
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]
...
```

La clef **deploy** permet de spécifier des options lors du déploiement du service :

- **mode** - Il existe deux types de services. **Replicated** où on spécifie le nombre d'instances que Docker doit mettre en place sur les hôtes **disponibles** en fonction de la valeur de **replicas** et **Global** qui implique que Docker démarrera une instance du service sur chaque hôte chaque fois qu'un hôte devient disponible.

- **replicas** - spécifie le nombre de replicas
- **restart_policy** spécifie ce qui se passe en cas d'arrêt du service. Dans le cas ci-dessus, docker va essayer de re-démarrer le service **3** fois (**max_attempts**) à des intervalles de **10** secondes (**delay**) en attendant chaque fois **120** secondes (**window**) pour constater si le service s'est effectivement re-démarré,
- **placement** - spécifie où le service doit être démarré.

Déployez maintenant le stack :

```
root@manager:~# docker stack deploy -c docker-stack.yml app
Creating network app_backend
Creating network app_default
Creating network app_frontend
Creating service app_worker
Creating service app_visualizer
Creating service app_redis
Creating service app_db
Creating service app_vote
Creating service app_result
```

Important - Notez que chaque réseau et chaque service a comme préfixe le nom de l'application **app**.

Consultez maintenant l'état du stack :

```
root@manager:~# docker stack ls
NAME                SERVICES            ORCHESTRATOR
app                 6                   Swarm
```

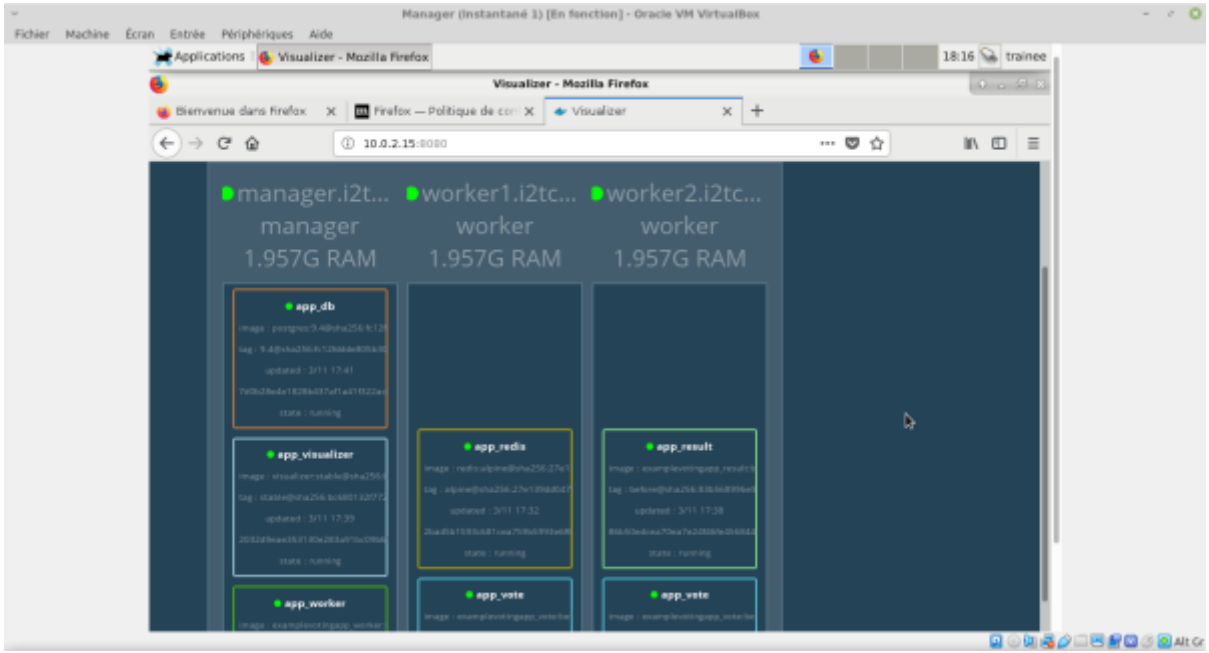
Consultez ensuite l'état des services :

```
root@manager:~# docker service ls
```

```
ID                  NAME                MODE                REPLICAS            IMAGE
PORTS
d0i4ac4fshw0        app_db              replicated          1/1                 postgres:9.4
funp5kboyip1        app_redis           replicated          1/1                 redis:alpine
dpdkc49oj671        app_result          replicated          1/1
dockersamples/examplevotingapp_result:before   *:5001->80/tcp
vrkahv38v5mn        app_visualizer      replicated          1/1                 dockersamples/visualizer:stable
*:8080->8080/tcp
t4u16cpdrx21        app_vote            replicated          2/2
dockersamples/examplevotingapp_vote:before     *:5000->80/tcp
so40eljbcviy        app_worker          replicated          1/1
dockersamples/examplevotingapp_worker:latest
```

Important : Notez que la configuration du service **visualizer** a exposé le port **8080**. De cette façon, ce service est disponible sur le port 8080 de chaque nœud dans la swarm.

Retournez à la fenêtre d'Apache Guacamole dans le navigateur de **votre** ordinateur. Cliquez sur la connexion **TraineeXX_VNC**. Lancez ensuite un navigateur Internet dans la machine virtuelle **debian9**. Naviguez à l'URL http://10.0.2.62:8080 et consultez le service **visualizer** :



Comme vous pouvez constater, conformément au fichier **docker-stack.yml**, les trois conteneurs **db**, **worker** et **visualizer** ont été démarrés sur le nœud manager.

Retournez à votre connexion SSH et consultez l'état des réseaux dans les trois nœuds :

```
root@manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
sw489bb290zb        app_backend         overlay             swarm
smuxoglyudpo        app_default         overlay             swarm
lfizui95od90        app_frontend        overlay             swarm
24be8a0f0ef5        bridge              bridge              local
d4c9b0c9437a        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
x7l4mk4ldb75        my-ingress          overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que les trois réseaux créés sont de type **overlay**.

```
root@worker1:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
qhysvpoolsw0        app_frontend        overlay             swarm
f9a69d02de3b        bridge              bridge              local
ee22b3e623ca        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
x7l4mk4ldb75        my-ingress          overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que seul le réseau **app_frontend** a été créé dans **worker1**.

```
root@worker2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
s4gbgi4isp1i        app_backend         overlay             swarm
qhysvpoolsw0        app_frontend        overlay             swarm
0e6c118bf3fd        bridge              bridge              local
0ce1d8369c29        docker_gwbridge     bridge              local
f3cb3bc3c581        host                host                local
x7l4mk4ldb75        my-ingress          overlay             swarm
de563e30d473        none                null                local
```

Important : Notez que les deux réseaux **app_frontend** et **app_backend** ont été créés dans **worker2**.

Consultez les informations concernant le réseau **app_backend** :

```
root@manager:~# docker inspect app_backend
[
    {
        "Name": "app_backend",
        "Id": "s4gbgi4isp1i5wjpgnf4uci2a",
        "Created": "2019-11-03T17:30:56.822222239+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.3.0/24",
                    "Gateway": "10.0.3.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "7d0b28e4e1828b437af1a41f322acb5cf19afc25c42303986dd2c7b4d5aea568": {
                "Name": "app_db.1.s6g6w47k532rvaeoyske8as9i",
                "EndpointID": "c26795c837f6dc736a3f9be34525ae505e9db6381a2144bb62087b3ee6c7ff25",
                "MacAddress": "02:42:0a:00:03:03",
                "IPv4Address": "10.0.3.3/24",
```

```
                "IPv6Address": ""
            },
            "ef7227281d297b001bb0f60ac81a0c9926e8fb663a7f43eb201cced632dc5564": {
                "Name": "app_worker.1.38kniuqoe1vfyonwdcytlhpqo",
                "EndpointID": "990065eec5062ff159e82bc1e4666fd098d5597439221995af4f01040ab24599",
                "MacAddress": "02:42:0a:00:03:09",
                "IPv4Address": "10.0.3.9/24",
                "IPv6Address": ""
            },
            "lb-app_backend": {
                "Name": "app_backend-endpoint",
                "EndpointID": "913845cbab9a6c3011eaaa87fcc66f10268b5e11554be9f1a20b1078f7b9b8a4",
                "MacAddress": "02:42:0a:00:03:04",
                "IPv4Address": "10.0.3.4/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4101"
        },
        "Labels": {
            "com.docker.stack.namespace": "app"
        },
        "Peers": [
            {
                "Name": "377986fb7d5a",
                "IP": "10.0.2.62"
            },
            {
                "Name": "5cc4b863da9f",
                "IP": "10.0.2.64"
            }
        ]
    }
```

```
]
```

Important : Notez que le réseau est **10.0.3.0/24** et la passerelle **10.0.3.1**.

Consultez les informations concernant le réseau **app_frontend** :

```
root@manager:~# docker inspect app_frontend
[
    {
        "Name": "app_frontend",
        "Id": "qhysvpoolsw0318gsubbvd3rx",
        "Created": "2019-11-03T17:31:27.354293132+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.2.0/24",
                    "Gateway": "10.0.2.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
```

```
        "ConfigOnly": false,
        "Containers": {
            "ef7227281d297b001bb0f60ac81a0c9926e8fb663a7f43eb201cced632dc5564": {
                "Name": "app_worker.1.38kniuqoe1vfyonwdcytlhpqo",
                "EndpointID": "3fad9773920412464b6aeee59f8d9ffc5aac2e937b88dc384268591cf7d21fb9",
                "MacAddress": "02:42:0a:00:02:0a",
                "IPv4Address": "10.0.2.10/24",
                "IPv6Address": ""
            },
            "lb-app_frontend": {
                "Name": "app_frontend-endpoint",
                "EndpointID": "343887373c1f92ac08b271ee52dd160089eeed7cad13b7924e438919254b6442",
                "MacAddress": "02:42:0a:00:02:0b",
                "IPv4Address": "10.0.2.11/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4100"
        },
        "Labels": {
            "com.docker.stack.namespace": "app"
        },
        "Peers": [
            {
                "Name": "0e21ba1bbfab",
                "IP": "10.0.2.63"
            },
            {
                "Name": "5cc4b863da9f",
                "IP": "10.0.2.64"
            },
            {
                "Name": "377986fb7d5a",
```

```
                "IP": "10.0.2.62"
            }
        ]
    }
]
```

Important : Notez que le réseau est **10.0.2.0/24** et la passerelle **10.0.2.1**.

Consultez les informations concernant le réseau **app_default** :

```
root@manager:~# docker inspect app_default
[
    {
        "Name": "app_default",
        "Id": "z62t49w18wl2mrboa92tunrhq",
        "Created": "2019-10-28T17:22:44.724040846+01:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.1.0/24",
                    "Gateway": "10.0.1.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
```

```
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "2032d9eae353130e283a91bc09b65b4a84b7e8f5602a466f4ea1bd9c64e964dc": {
                "Name": "app_visualizer.1.nbf78cn5g37dmu0fwrxt7kbrg",
                "EndpointID": "d5274ff057c9d9af0288efb7f9bfed3a5ca1b3e656e265ad343f52c0b1c161f5",
                "MacAddress": "02:42:0a:00:01:03",
                "IPv4Address": "10.0.1.3/24",
                "IPv6Address": ""
            },
            "lb-app_default": {
                "Name": "app_default-endpoint",
                "EndpointID": "6afb8909d94528633e4150054311f645790280b1ab1c686c43e865ba97ec3df9",
                "MacAddress": "02:42:0a:00:01:04",
                "IPv4Address": "10.0.1.4/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4099"
        },
        "Labels": {
            "com.docker.stack.namespace": "app"
        },
        "Peers": [
            {
                "Name": "377986fb7d5a",
                "IP": "10.0.2.62"
            }
        ]
    }
```

```
]
```

Important : Notez que le réseau est **10.0.1.0/24** et la passerelle **10.0.1.1**.

Schématiquement, la mise en place de l'application dans le Swarm est ainsi :

Dernièrement, supprimez le stack :

```
root@manager:~# docker stack ls
NAME                SERVICES            ORCHESTRATOR
app                 6                   Swarm
root@manager:~# docker stack rm app
Removing service app_db
Removing service app_redis
Removing service app_result
Removing service app_visualizer
Removing service app_vote
Removing service app_worker
Removing network app_frontend
Removing network app_backend
Removing network app_default
root@manager:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
d02c6115724c        alpine              "/bin/sh"                6 days ago          Exited (0) 6 days ago
alpine1
```