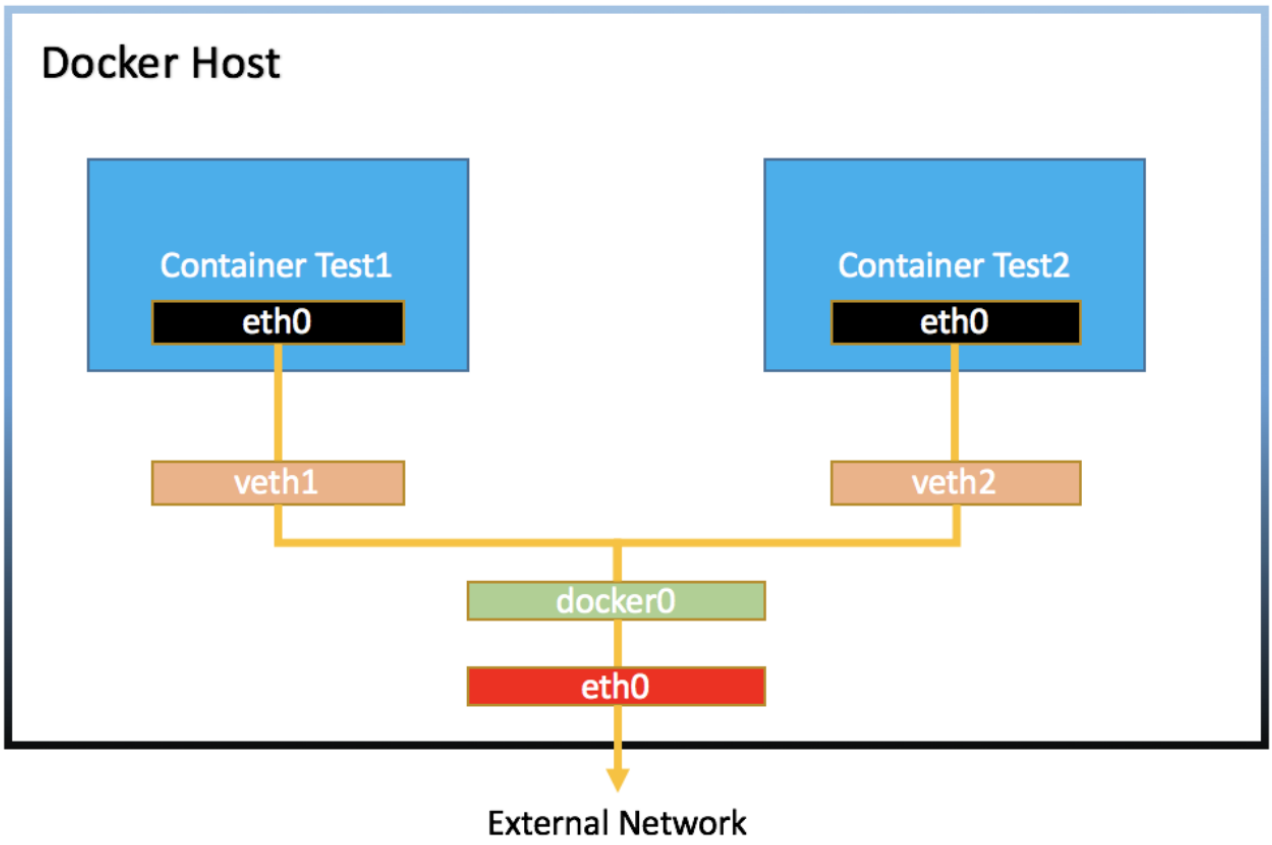
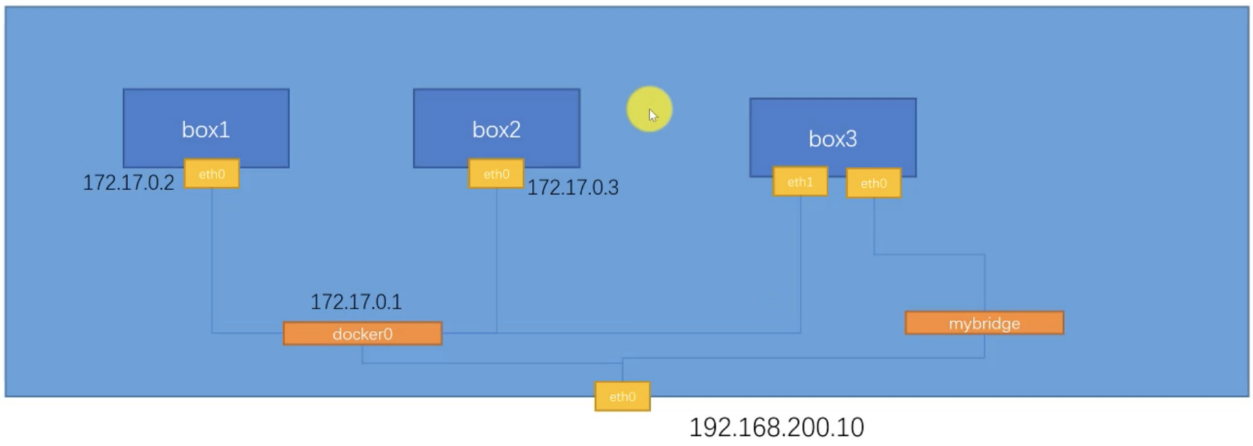
Docker Network





建立兩容器

```
$ docker container run -d --rm --name box1 busybox /bin/sh -c "while true; do sleep 3600; done"
$ docker container run -d --rm --name box2 busybox /bin/sh -c "while true; do sleep 3600; done"
$ docker container ls
CONTAINER ID   IMAGE     COMMAND               CREATED         STATUS         PORTS     NAMES
4f3303c84e53   busybox   "/bin/sh -c 'while t…"   49 minutes ago   Up 49 minutes            box2
03494b034694   busybox   "/bin/sh -c 'while t…"   49 minutes ago   Up 49 minutes            box1
```

容器間的通信->使用bridge

```
$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
```

```
1847e179a316   bridge    bridge    local
a647a4ad0b4f   host      host      local
fbd81b56c009   none      null      local
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "1847e179a316ee5219c951c2c21cf2c787d431d1ffb3ef621b8f0d1edd197b24",
        "Created": "2021-07-01T15:28:09.265408946Z",
        "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": {
            "03494b034694982fa085cc4052b6c7b8b9c046f9d5f85f30e3a9e716fad20741": {
                "Name": "box1",
                "EndpointID": "072160448becebb7c9c333dce9bbdf7601a92b1d3e7a5820b8b35976cf4fd6ff",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "4f3303c84e5391ea37db664fd08683b01decdadae636aaa1bfd7bb9669cbd8de": {
                "Name": "box2",
                "EndpointID": "4cf0f635d4273066acd3075ec775e6fa405034f94b88c1bcacdaae847612f2c5",
                "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": {}
    }
]
```

```
"IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
    {
        "Subnet": "172.17.0.0/16",
       "Gateway": "172.17.0.1"
    }
    ]
},

"com.docker.network.bridge.name": "docker0",
```

```
# brctl 顯示bridge tool
$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242759468cf       no              veth8c9bb82
                                                        vethd8f9afb
```

容器對外通訊->iptables nat

```
$ ip route
default via 10.0.2.2 dev eth0 proto dhcp metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
192.168.200.0/24 dev eth1 proto kernel scope link src 192.168.200.10 metric 101
```

```
$ sudo iptables --list -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere             anywhere             ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere             !loopback/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16       anywhere

Chain DOCKER (2 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere
```

```
$ docker network create -d bridge --gateway 172.200.0.1 --subnet 172.200.0.0/16 demo
e353488ae0b3ed73c8ae47410d3f03b626fd5b43
```

docker network create --help

docker network create -d {driver} --gateway {gateway} --subnet {subnet} {name}
docker network create -d bridge --gateway 127.200.0.1 -subnet 172.200.0.0/16 demo

```
[tomcat@docker-qc html-doc]$ docker network create -d bridge mybridge
65c5ba6cbea2fa677e8dd0623a510f3461e24ccac196b7374fc5e8d03803ec9e

[tomcat@docker-qc html-doc]$  docker network ls
NETWORK ID          NAME              DRIVER            SCOPE
780162e5fe57        bridge            bridge            local
3c5b59c8a708        host              host              local
65c5ba6cbea2        mybridge          bridge            local

[tomcat@docker-qc html-doc]$ docker network inspect mybridge
[
   {
      "Name": "mybridge",
      "Id": "65c5ba6cbea2fa677e8dd0623a510f3461e24ccac196b7374fc5e8d03803ec9e",
      "Created": "2022-01-18T18:44:39.95740673+08:00",
      "Scope": "local",
      "Driver": "bridge",
      "EnableIPv6": false,
      "IPAM": {
         "Driver": "default",
         "Options": {},
         "Config": [
            {
               "Subnet": "172.18.0.0/16",
```

```
            "Gateway": "172.18.0.1"
          }
        ]
      },
      "Internal": false,
      "Attachable": false,
      "Ingress": false,
      "ConfigFrom": {
         "Network": ""
      },
      "ConfigOnly": false,
      "Containers": {},
      "Options": {},
      "Labels": {}
   }
]
```

```
# 建立連接mybridge的container (--network 不設定會用預設的)
$ docker container run -d --rm --name box3 --network mybridge busybox /bin/sh -c "while true; do sleep 3600; done"
# 既有container 連接mybridge
$ docker network connect mybridge box1
# container 中斷 mybridge
$ docker network disconnect mybridge box1
```

Network type

- bridge
- host
- none

bridge: 新增一網段，橋接本地網路
host: 直接使用本地網路(無法看到port -> 等於直接開在本機port 80，好處不用nat，壞處只能開一個)

```
#docker dontainer run -d --name {container name} --network {network type} {image}
docker dontainer run -d --name web --network host nginx
```

```
[vagrant@docker-host1 ~]$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
237047b2ec33   nginx     "/docker-entrypoint.…"   51 seconds ago   Up 51 seconds             web4
08ad8793636d   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80/tcp    web3
5c59480a8ad4   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80/tcp    web2
0095cec5cff8   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80/tcp    web1
```

只能開一個，多開會報錯

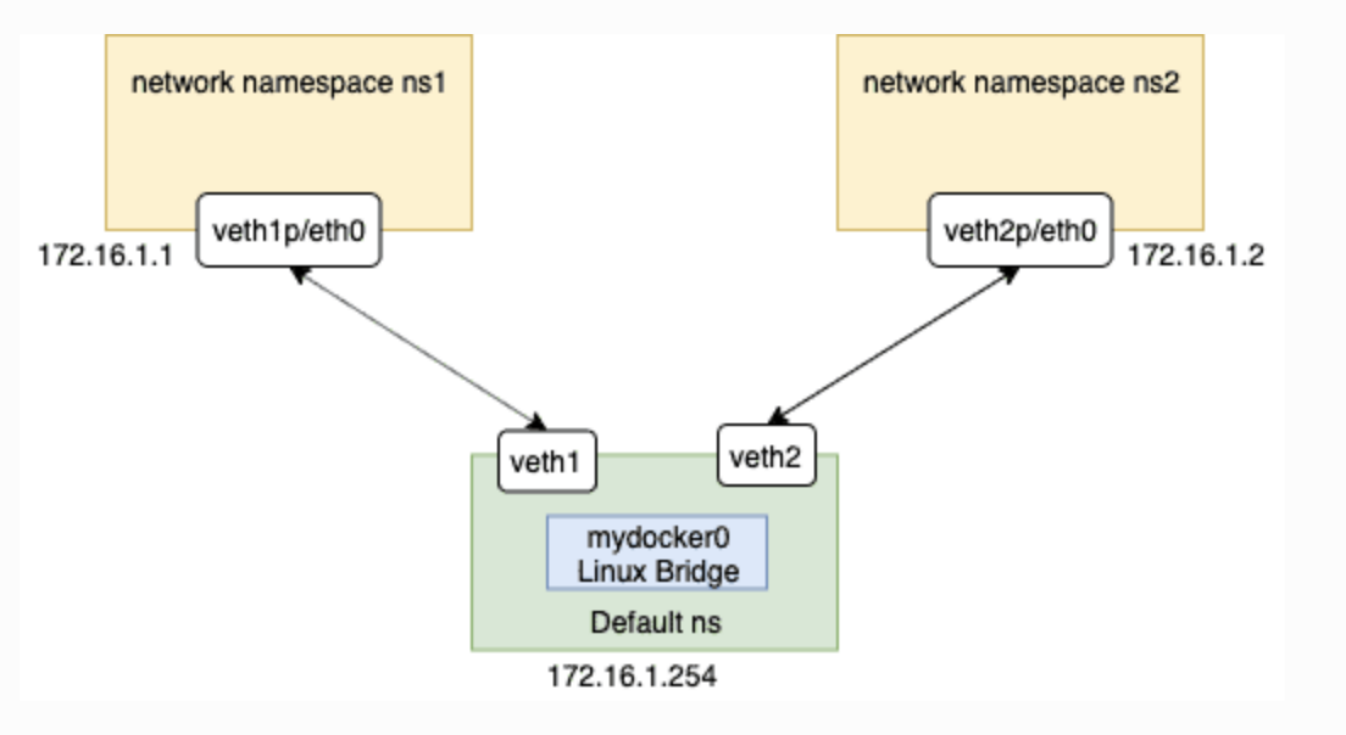
```
2021/07/05 21:41:24 [emerg] 1#1: bind() to [::]:80 failed (98: Address already in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
```

none: 只有 lookback 127.0.0.1 沒有其他網路

網路命名空間

- bridge tool : brctl 安裝

```
#Ubuntu
apt-get install bridge-utils
#Centos
sudo yum install bridge-utils
```



創建bridge

```
[vagrant@docker-host1 ~]$ sudo brctl addbr mydocker0
[vagrant@docker-host1 ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
mydocker0               8000.000000000000       no
[vagrant@docker-host1 ~]$
```

add-ns-to-br.sh

```
#!/bin/bash

bridge=$1
namespace=$2
addr=$3

vethA=veth-$namespace
vethB=eth00

sudo ip netns add $namespace
sudo ip link add $vethA type veth peer name $vethB

sudo ip link set $vethB netns $namespace
sudo ip netns exec $namespace ip addr add $addr dev $vethB
sudo ip netns exec $namespace ip link set $vethB up

sudo ip link set $vethA up

sudo brctl addif $bridge $vethA
```

```
#run shell
[vagrant@docker-host1 ~]$ sh add-ns-to-br.sh mydocker0 ns1 172.16.1.1/16
[vagrant@docker-host1 ~]$ sh add-ns-to-br.sh mydocker0 ns2 172.16.1.2/16

#up bridge
[vagrant@docker-host1 ~]$ sudo ip link set dev mydocker0 up
```

驗證

```
[vagrant@docker-host1 ~]$ sudo ip netns exec ns1 bash
[root@docker-host1 vagrant]# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
```

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: eth00@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether f2:59:19:34:73:70 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.1.1/16 scope global eth00
    valid_lft forever preferred_lft forever
    inet6 fe80::f059:19ff:fe34:7370/64 scope link
    valid_lft forever preferred_lft forever
[root@docker-host1 vagrant]# ping 172.16.1.2
PING 172.16.1.2 (172.16.1.2) 56(84) bytes of data.
64 bytes from 172.16.1.2: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 172.16.1.2: icmp_seq=2 ttl=64 time=0.080 ms
^C
--- 172.16.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.029/0.054/0.080/0.026 ms
[root@docker-host1 vagrant]#
```

```
[root@docker-host1 vagrant]# sudo ip netns list
ns2
ns1
[root@docker-host1 vagrant]# sudo ip netns exec ns1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: eth00@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether f2:59:19:34:73:70 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.1.1/16 scope global eth00
       valid_lft forever preferred_lft forever
    inet6 fe80::f059:19ff:fe34:7370/64 scope link
       valid_lft forever preferred_lft forever
[root@docker-host1 vagrant]# sudo ip netns exec ns2 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
7: eth00@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether fe:92:bf:24:0c:2e brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.1.2/16 scope global eth00
       valid_lft forever preferred_lft forever
    inet6 fe80::fc92:bfff:fe24:c2e/64 scope link
       valid_lft forever preferred_lft forever
[root@docker-host1 vagrant]# sudo ip netns exec ns2 ip a
```

```
[vagrant@docker-host1 ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
mydocker0               8000.8e5108541d61       no              veth-ns1
                                                                veth-ns2
```

使用iptables 實現nat
NAT with Linux and iptables - Tutorial (Introduction) (karlrupp.net)

使用自定義網路練習

使用自定義網路，預設自帶dns，容器間可相互使用name ping 通

準備python 檔案 **app.py**

```
from flask import Flask
from redis import Redis
import os
import socket

app = Flask(__name__)
redis = Redis(host=os.environ.get('REDIS_HOST', '127.0.0.1'), port=6379)


@app.route('/')
def hello():
    redis.incr('hits')
    return f"Hello Container World! I have been seen {redis.get('hits').decode('utf-8')} times and my hostname is
{socket.gethostname()}.\n"
```

準備一個 Dockerfile

```
FROM python:3.9.5-slim

RUN pip install flask redis && \
    groupadd -r flask && useradd -r -g flask flask && \
    mkdir /src && \
    chown -R flask:flask /src

USER flask

COPY app.py /src/app.py

WORKDIR /src

ENV FLASK_APP=app.py REDIS_HOST=redis

EXPOSE 5000

CMD ["flask", "run", "-h", "0.0.0.0"]
```

build image 並下載一個redis image

```
$ docker image pull redis
$ docker image build -t flask-demo .
$ docker image ls
REPOSITORY   TAG         IMAGE ID      CREATED            SIZE
flask-demo   latest      4778411a24c5   About a minute ago   126MB
python       3.9.5-slim   c71955050276   8 days ago         115MB
redis        latest      08502081bff6   2 weeks ago        105MB
```

建立一個bridge

```
$ docker network create -d bridge demo-network
8005f4348c44ffe3cdcbbda165beea2b0cb520179d3745b24e8f9e05a3e6456d
$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
2a464c0b8ec7   bridge        bridge    local
8005f4348c44   demo-network   bridge    local
80b63f711a37   host          host      local
fae746a75be1   none          null      local
```

建立redis container 並連至 demo-network

```
$ docker container run -d --name redis-server --network demo-network redis
002800c265020310231d689e6fd35bc084a0fa015e8b0a3174aa2c5e29824c0e
$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
002800c26502   redis     "docker-entrypoint.s..."   4 seconds ago   Up 3 seconds   6379/tcp   redis-server
```

建立 flask container

```
$ docker container run -d --network demo-network --name flask-demo --env REDIS_HOST=redis-server -p 5000:5000 flask-demo
```

打開瀏覽器 http://127.0.0.1:5000

可以看到以下內容，每次刷新次次就會+1
Hello Container World! I have been seen36 times and my hostname is 925ecb8d111a.

整理後腳本

```
# prepare image
docker image pull redis
docker image build -t flask-demo .

# create network
docker network create -d bridge demo-network
```

```
# create container
docker container run -d --name redis-server --network demo-network redis
docker container run -d --network demo-network --name flask-demo --env REDIS_HOST=redis-server -p 5000:5000 flask-demo
```

修訂版本 #24
★由 treeman 建立於 19 ﷺ 2022 01:02:48
由 treeman 更新於 5 ﷺ 2023 10:14:58