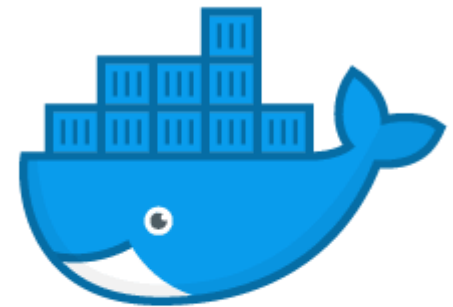


Docker

Riccardo Cattaneo



docker

Networking

Il modello di gestione del Networking di docker è denominato «Container Network Model» (CNM). La configurazione di base del networking in docker prevede 3 tipologie di networking :

BRIDGE (è quella di default e quella di nostro interesse. Connessione del container con l'host : sono 2 reti)

HOST (viene «scavalcato» il networking di Docker. Connessione diretta con l'host : la rete è 1)

NULL (nessuna rete quindi nessun ip assegnato, in questo caso non è permessa la comunicazione tramite la rete)

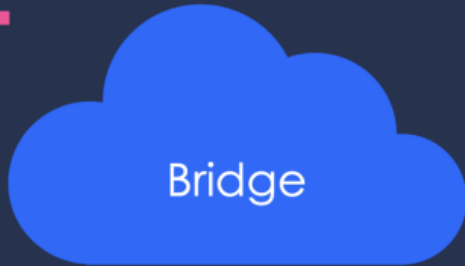
E' possibile verificarlo tramite il comando

docker network ls

Modalità «bridge»

La modalità «bridge» ci permette di interagire con l'host (ovvero il kernel Linux) tramite l'interfaccia Linux "docker0".

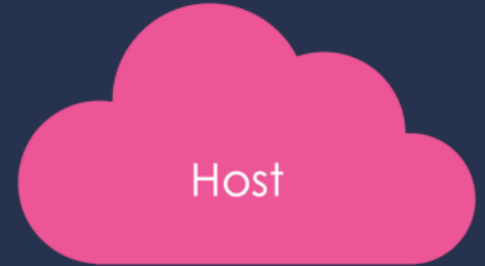
Per cui è presente una dipendenza tra il networking fornito da Docker (bridge) e il networking del sistema operativo.



`docker run ubuntu`

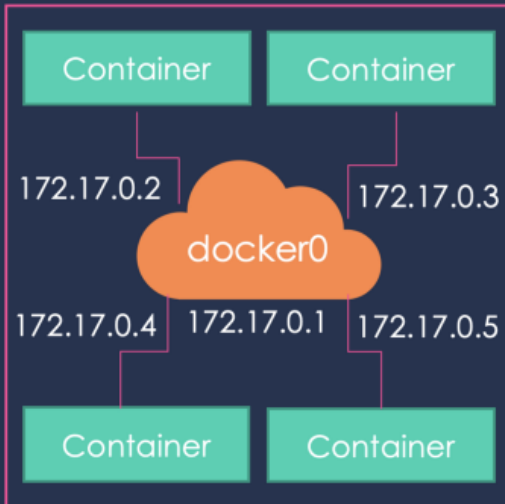


`docker run \`
`--network=none`
`ubuntu`

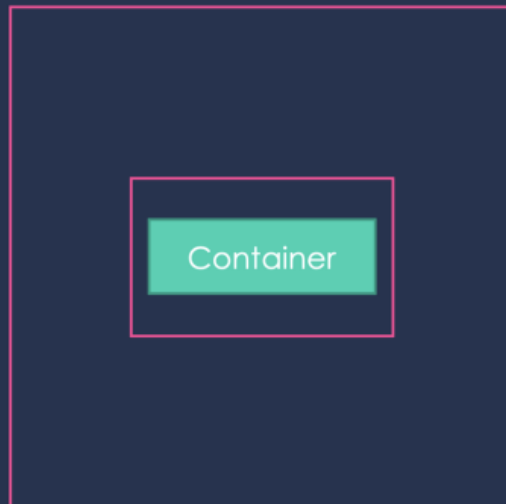


`docker run \`
`--network=host`
`ubuntu`

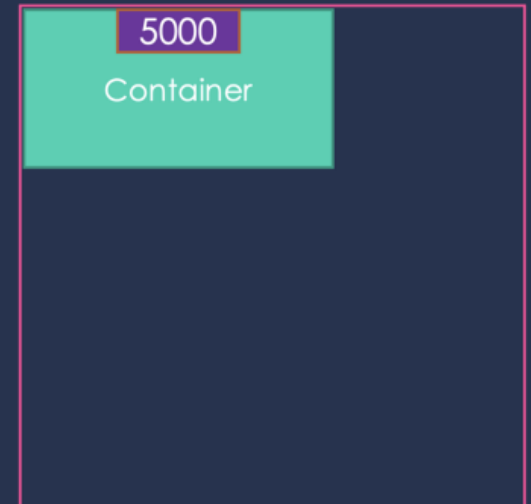
Docker Host



Docker Host



Docker Host



Tipologie di reti in docker

Con il comando `docker network ls` possiamo visualizzare le reti di default presenti nel docker.

```
riccardo@riccardo-prof01: ~  
riccardo@riccardo-prof01:~$ docker network ls
```

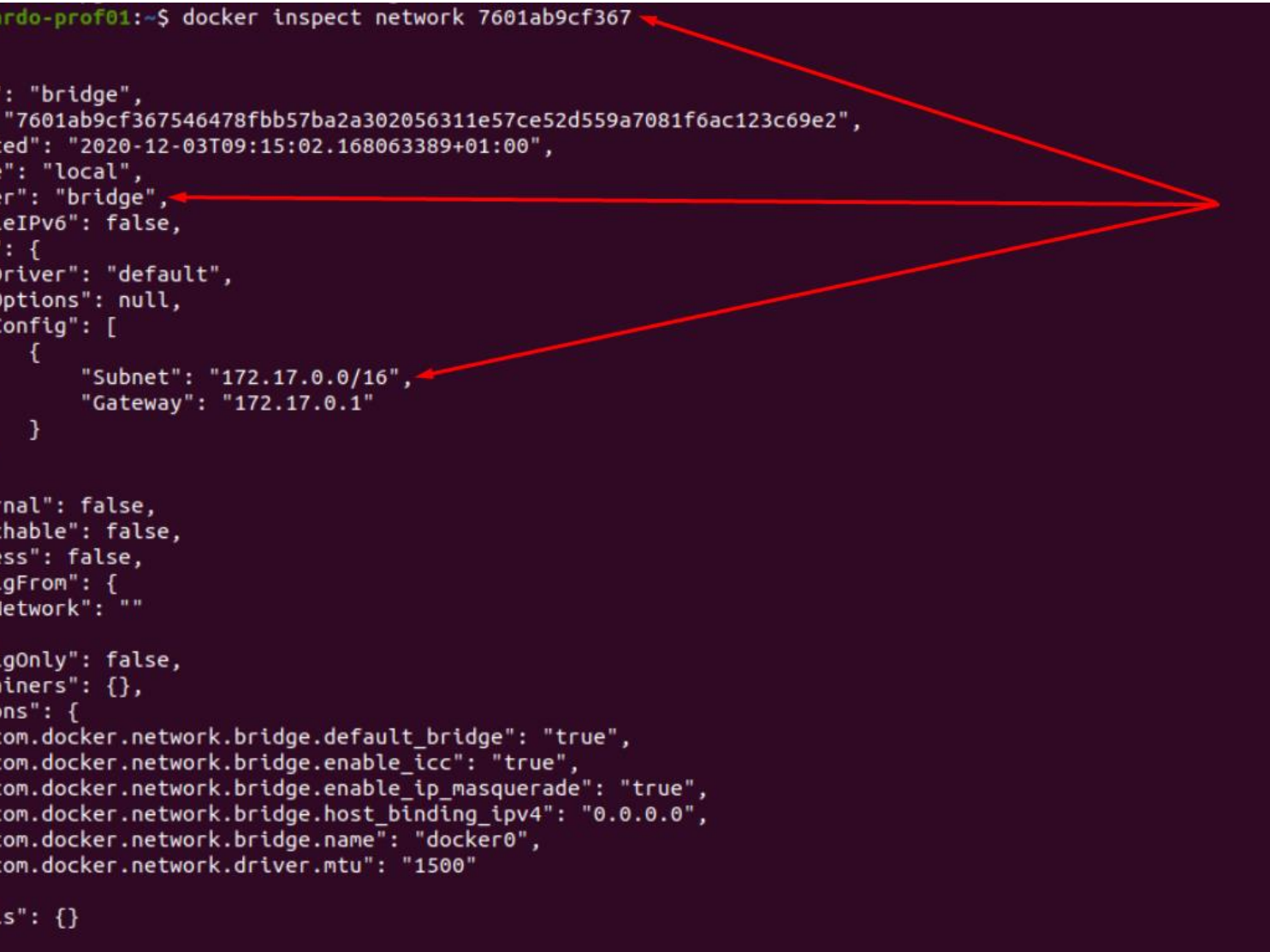
NETWORK ID	NAME	DRIVER	SCOPE
7601ab9cf367	bridge	bridge	local
a32da9a9e666	host	host	local
c890f119ec11	none	null	local

Proviamo ora ad analizzare la rete bridge di default di docker con il seguente comando :

docker inspect network ID_NETWORK

Come possiamo vedere abbiamo delle informazioni dettagliate sulla rete, tra cui la modalità dei driver utilizzata, poi abbiamo una sottorete (Subnet) che servirà poi per assegnare un indirizzo ip ad ogni container.

```
riccardo@riccardo-prof01:~$ docker inspect network 7601ab9cf367
[
  {
    "Name": "bridge",
    "Id": "7601ab9cf367546478fbb57ba2a302056311e57ce52d559a7081f6ac123c69e2",
    "Created": "2020-12-03T09:15:02.168063389+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": {}
  }
]
```



Andiamo ora a creare un container ubuntu e verifichiamo l'assegnamento dell'indirizzo ip :

```
docker run -it --name cnetwork1 ubuntu bash
```

```
apt-get update
```

```
apt-get install net-tools
```

```
ifconfig
```

Normalmente viene assegnato un nome alla nostra scheda di rete (di solito è **eth0**) ed il nostro indirizzo ip viene identificato con la sigla **inet** :


```
root@9c44cec962e6:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.17.0.2 netmask 255.255.0.0  broadcast 172.17.255.255
    ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
    RX packets 273  bytes 542697 (542.6 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 247  bytes 14484 (14.4 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    loop  txqueuelen 1000  (Local Loopback)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ctrl p + ctrl q

docker network ls

docker inspect network ID_NETWORK

Possiamo quindi verificare che l'indirizzo ip del container rientra nella stessa rete di default di docker.

Comunicazione tra 2 container

```
docker run -it --name cnetworkA ubuntu bash  
apt-get update  
apt-get install net-tools  
ifconfig
```

CTRL + P e CTRL + Q

```
docker run -it --name cnetworkB ubuntu bash  
apt-get update  
apt-get install net-tools  
ifconfig
```

```
docker inspect network ID_NETWORK
```

Dopo aver creato i due container connettiamoci al primo con il seguente comando :

```
docker exec -it cnetworkA bash
```

Proviamo ora a fare un ping verso l'indirizzo ip del secondo container (prima installarlo con `apt-get install iputils-ping`)

```
ping INDIRIZZO_IP
```

Confermiamo quindi che i due container si «vedono» e possono comunicare tra di loro.

Comunicazione verso l'esterno NAT

```
docker run -it --name cnetworkC ubuntu bash  
apt-get update  
apt-get install iputils-ping
```

Una volta dentro il container vogliamo effettuare un ping verso 8.8.8.8 che è il DNS di Google come esempio per testare la connettività verso l'esterno.

```
ping 8.8.8.8
```

Possiamo notare che il server ci risponde regolarmente dimostrando che comunica tranquillamente con server pubblici esterni. Come è possibile ?

Questo avviene perché l'host ci sta offrendo un meccanismo di NAT (Network Address Translation) ovvero sta convertendo il nostro indirizzo ip privato verso il mondo esterno in un indirizzo ip pubblico...

Vediamo ora come può essere effettuato il **Port Mapping delle porte**. Ovvero poter connettere una porta del container con una porta dell'host così da poter uscire dal container e comunicare con l'esterno.

Andremo adesso ad eseguire un container e mappare la porta del container con una porta dell'host e poi provare a connetterci a quella porta specificata sull'host e verificare che il traffico sia poi redirezionato verso il container (con l'opzione **-p** che sta per **publish**).

```
docker run -d - -name webserver1 -p 8082:80 nginx
```

PORTA HOST : PORTA CONTAINER

Per verificarlo tramite l'host proviamo ad eseguire il seguente comando : `wget 127.0.0.1:8082` oppure tramite il browser digitando sempre l'indirizzo `127.0.0.1:8082`

Creazione di una Rete

docker network create reteTest

Eseguendo questo comando viene creata una nuova rete di tipo «bridge» e viene chiamata reteTest. A questo punto verifichiamo se è stata creata la rete :

docker network ls

docker inspect network reteTest

Creare ora un container e proviamo ad assegnarlo alla nostra rete appena creata (ricordiamo che è possibile digitare il comando `docker network --help` per vedere le varie opzioni):

```
docker run -it --name ubuntu1 --network  
reteTest ubuntu bash
```

```
docker inspect ubuntu1
```

Verificare che siamo all'interno della nostra reteTest. Ogni rete è isolata, se creo un container nella reteTest, lo stesso container non è visibile all'interno della rete generica bridge.

Attenzione

Ricordiamo che tutti i container che apparterranno alla reteTest potranno comunicare **solo** internamente tra di loro. Per farli comunicare dovremmo aggiungere un container che fa da «routing» dei pacchetti, ma questa è una situazione particolare che non rientra nelle casistiche più comuni.

DNS

La Rete di default bridge non supporta la risoluzione dei DNS. Le reti custom di tipo bridge invece supportano la risoluzione dei nomi tramite un DNS interno a docker.

Per verificarlo eseguiamo due container sulla rete bridge di default e due container sulla reteTest e provare a fare il ping tra i due container della rete generica e il ping tra i due container della reteTest

```
docker run -it --name ubuntu1reteTest --network reteTest  
ubuntu bash
```

```
docker run -it --name ubuntu2reteTest --network reteTest  
ubuntu bash
```

```
docker run -it --name ubuntu1default ubuntu bash
```

```
docker run -it --name ubuntu2default ubuntu bash
```

```
docker ps
```

```
docker exec -it ubuntu1default bash
```

```
ping ubuntu2default (NON FUNZIONA)
```

```
docker exec -it ubuntu1reteTest bash
```

```
ping ubuntu2reteTest (FUNZIONA)
```