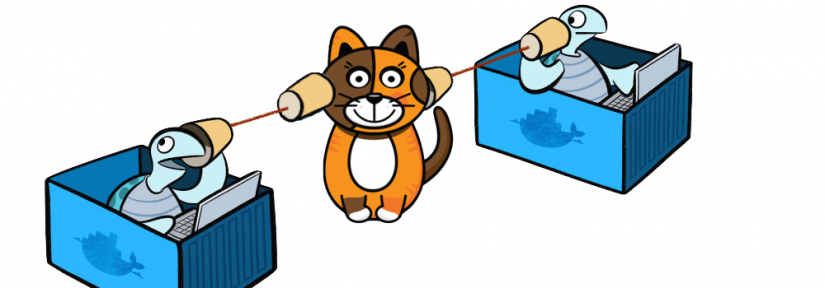
**Tìm hiểu và làm việc với docker container networks (P1)**

[Docker](https://viblo.asia/tags/docker)[DevOps](https://viblo.asia/tags/devops)

 Aug 29th, 2016 06:42:17

 276 3 0



**Bài viết này xin đưa ra cái nhìn tổng quan về default networking bên trong Docker native. Mô tả các loại network trong docker, làm thế nào để tạo ra chúng, các tài nguyên cần thiết để tạo network trên single host hoặc cluster hosts.**

**1. Default Networks**

Khi bạn cài đặt xong Docker, sẽ có 3 loại networks được tự động tạo ra. Chúng ta có thể liệt kê tất cả bằng câu lệnh docker network ls:

$ docker network ls

NETWORK ID NAME DRIVER

7fca4eb8c647 bridge bridge

9f904ee27bf5 none null

cf03ee007fb4 host host

Ở đây chúng ta có 3 loại networks mặc định của docker, Khi bạn run một container, bạn có thể sử dụng flag: **--network** để chỉ định loại network mà bạn muốn container chạy trên đó khi **run**. Vì mặc định được tao ra bởi docker, nên bạn có thể sử dụng nó bất cứ khi nào cần.

***bridge*** network được thể hiện bởi *docker0* network. Trù khi bạn chỉ định một option network khác bằng lệnh docker run --network=<NETWORK>, nếu ko Docker daemon sẽ tự động connect các containers tới loại network này. Bạn có thể thấy ***bridge*** như một phần của host’s network stack bằng cách sử dụng câu lệnh ifconfig trên host:

$ ifconfig

docker0 Link encap:Ethernet HWaddr 02:42:47:bc:3a:eb

inet addr:172.17.0.1 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:47ff:febc:3aeb/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1

RX packets:17 errors:0 dropped:0 overruns:0 frame:0

TX packets:8 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:1100 (1.1 KB) TX bytes:648 (648.0 B)

***none*** network thêm một container vào container-specific network stack. Container này bị thiếu network interface. Thử attach một container và xem điều gì đã diễn ra :v

$ docker attach nonenetcontainer

root@0cb243cd1293:/*# 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

root@0cb243cd1293:/*# ifconfig*

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

root@0cb243cd1293:/*#*

***host*** network thêm một container lên hosts network stack. Bạn có thể tìm thấy network configuration bên trong container đồng nhất với host.

Ngoại trừ **bridge** network, bạn ko thực sự cần tương tác với default networks. Khi bạn có thể list hoặc inspect chúng, tuy nhiên bạn không thể xóa chúng. Ngoài default network, ta có thể add thêm user-defined networks, có thể xóa khi ko muốn dùng. Trước khi học cách để tạo ra networks, chúng ta sẽ tìm hiểu một chút về **default bridge network**.

**Bridge network**

**Default bridge network** được hiện diện trên tất cả các Docker hosts. Câu lệnh docker network inspect trả lại các thông tin về network:

$ docker network inspect bridge

[

{

"Name": "bridge",

"Id": "f7ab26d71dbd6f557852c7156ae0574bbf62c42f539b50c8ebde0f728a253b6f",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.17.0.1/16",

"Gateway": "172.17.0.1"

}

]

},

"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": "9001"

}

}

]

Docker Engine tự động tạo ra Subnet và Gateway tới network. Trong khi câu lệnh docker run tự động add container mới tạo vào chính network này:

$ docker run -itd --name=container1 busybox

3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c

$ docker run -itd --name=container2 busybox

94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c

Thử inspect **bridge network** lại, sau khi chạy 2 containers chúng ta sẽ thấy id của container hiển thị trên “Containers” section:

$ docker network inspect bridge

{[

{

"Name": "bridge",

"Id": "f7ab26d71dbd6f557852c7156ae0574bbf62c42f539b50c8ebde0f728a253b6f",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.17.0.1/16",

"Gateway": "172.17.0.1"

}

]

},

"Containers": {

"3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c": {

"EndpointID": "647c12443e91faf0fd508b6edfe59c30b642abb60dfab890b4bdccee38750bc1",

"MacAddress": "02:42:ac:11:00:02",

"IPv4Address": "172.17.0.2/16",

"IPv6Address": ""

},

"94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c": {

"EndpointID": "b047d090f446ac49747d3c37d63e4307be745876db7f0ceef7b311cbba615f48",

"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": "9001"

}

}

]

Câu lệnh docker network inspect hiển thị toàn bộ các container được kết nối và network resources của loại network được truyền vào. Containers trong default network có thể giao tiếp với nhau bằng IP addresses. Docker không hỗ trợ tự động quét service trên default bridge network. Nếu bạn muốn giao tiếp thông qua container names trong default bridge network, bạn phải kết nối các containers với nhau thông qua docker run --link option.

Bạn có thể attach & chạy một container, và xem thông tin về configuration:

$ docker attach container1

root@0cb243cd1293:/# ifconfig

ifconfig

eth0 Link encap:Ethernet HWaddr 02:42:AC:11:00:02

inet addr:172.17.0.2 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe11:2/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1

RX packets:16 errors:0 dropped:0 overruns:0 frame:0

TX packets:8 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:1296 (1.2 KiB) TX bytes:648 (648.0 B)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Hãy thử sử dụng ping để gửi 3 **ICMP** requests, và test kết nối của các containers trên bridge network.

root@0cb243cd1293:/*# ping -w3 172.17.0.3*

PING 172.17.0.3 (172.17.0.3): 56 data bytes

64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.096 ms

64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.17.0.3: seq=2 ttl=64 time=0.074 ms

*--- 172.17.0.3 ping statistics ---*

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 0.074/0.083/0.096 ms

Cuối cùng, hãy sử dụng câu lệnh cat để kiểm tra cấu hình network container1:

root@0cb243cd1293:/# cat /etc/hosts

172.17.0.2 3386a527aa08

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

Để detach ra khỏi container1 nhưng vẫn muốn chúng tiếp tục running, ta sử dụng CTRL-p CTRL-q. Sau đó attach container2 và lặp lại 3 câu lệnh: ifconfig, ping, cat.

$ docker attach container2

root@0cb243cd1293:/# ifconfig

eth0 Link encap:Ethernet HWaddr 02:42:AC:11:00:03

inet addr:172.17.0.3 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1

RX packets:15 errors:0 dropped:0 overruns:0 frame:0

TX packets:13 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:1166 (1.1 KiB) TX bytes:1026 (1.0 KiB)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

root@0cb243cd1293:/# ping -w3 172.17.0.2

PING 172.17.0.2 (172.17.0.2): 56 data bytes

64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.067 ms

64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.075 ms

64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.072 ms

--- 172.17.0.2 ping statistics ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 0.067/0.071/0.075 ms

/ # cat /etc/hosts

172.17.0.3 94447ca47985

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

Mặc định docker0 bridge network hỗ trợ sử dụng port mapping và docker run --link cho phép các containers giao tiếp với nhau qua docker0 network.

**2. User-defined networks**

Bạn có thể tạo ra user-defined networks, Docker cung cấp một vài default **network drivers** cho việc create networks. Bạn có thể tạo ra 1 **bridge network** mới hoàn toàn hoặc **overlay network** hoặc **MACVLAN network**. Bạn cũng có thể tạo ra **network plugin** hoặc **remote network**.

Docker cho phép ta tạo nhiều network, Một network có thể có nhiều containers, các containers chỉ có thể communicate với nhau bên trong network, một container được attached vào 2 network.

**A bridge network**

**user-defined network** dễ dàng tạo ra nhất là bridge network. Network này tương tự default docker0 network.

$ docker network create *--driver bridge isolated\_nw*

1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b

$ docker network inspect isolated\_nw

[

{

"Name": "isolated\_nw",

"Id": "1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.21.0.0/16",

"Gateway": "172.21.0.1/16"

}

]

},

"Containers": {},

"Options": {}

}

]

$ docker network ls

NETWORK ID NAME DRIVER

9f904ee27bf5 none null

cf03ee007fb4 host host

7fca4eb8c647 bridge bridge

c5ee82f76de3 isolated\_nw bridge

Sau khi tạo ra network, bạn có thể chạy containers trên đó bằng cách sử dụng docker run --network=<NETWORK>option.

$ docker run --network=isolated\_nw -itd --name=container3 busybox

8c1a0a5be480921d669a073393ade66a3fc49933f08bcc5515b37b8144f6d47c

$ docker network inspect isolated\_nw

[

{

"Name": "isolated\_nw",

"Id": "1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{}

]

},

"Containers": {

"8c1a0a5be480921d669a073393ade66a3fc49933f08bcc5515b37b8144f6d47c": {

"EndpointID": "93b2db4a9b9a997beb912d28bcfc117f7b0eb924ff91d48cfa251d473e6a9b08",

"MacAddress": "02:42:ac:15:00:02",

"IPv4Address": "172.21.0.2/16",

"IPv6Address": ""

}

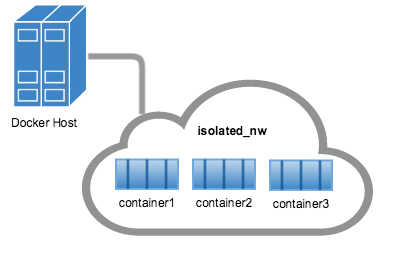
},

"Options": {}

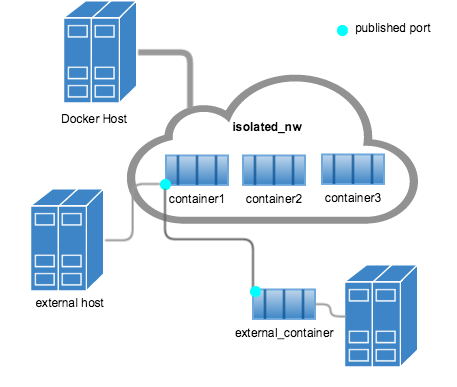
}

]

Container mà bạn chạy trên network này đều phải thuộc về cùng một Docker host. Mỗi container trong network có thể communicate với các containers khác trong cùng network.



linking không được hỗ trợ bên trong một **user-defined bridge network**. Bạn có thể expose và publish container ports trên các containers trong network này. Điều này rất hữu ích nếu bạn muốn tạo ra 1 phầm của bridge networkavailable với một outside network.



**bridge network** rất hữu ích trong các trường hợp mà ở đó ta muốn chạy một relatively small network trên single host. Tuy nhiên chúng ta có thể tạo ra các networks lớn hơn bằng cách tạo một **overlay network**.

**An overlay network with Docker Engine swarm mode**

Bạn có thể tạo ra một **overlay network** trên một manager node running trong swarm mode mà không cần tới một external key-value store. Swarm tạo ra **overlay network** chỉ available với các nodes bên trong swarm. Khi bạn tạo ra một service sử dụng overlay network, manager node sẽ tự động kế thừa overlay network tới các nodes chạy các service tasks.

Ví dụ sau sẽ hướng dẫn cách tạo ra một network và sử dụng nó cho một service từ một manager node bên trong swarm:

*# Create an overlay network `my-multi-host-network`.*

$ docker network create \

--driver overlay \

--subnet 10.0.9.0/24 \

my-multi-host-network

400g6bwzd68jizzdx5pgyoe95

*# Create an nginx service and extend the my-multi-host-network to nodes where*

*# the service's tasks run.*

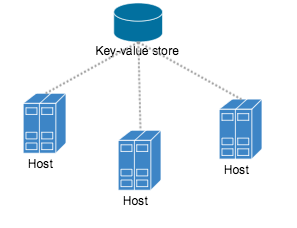
$ $ docker service create --replicas 2 --network my-multi-host-network --name my-web nginx

716thylsndqma81j6kkkb5aus

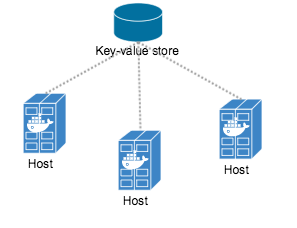
**An overlay network with an external key-value store**

Nếu bạn không sử dụng Docker Engine trong swarm mode, thì **overlay network** yêu cầu phải có một valid key-value store service. Hỗ trợ key-value stores gồm có Consul, Etcd, và ZooKeeper (Distributed store).

**Chú ý: Docker Engine chạy trong swarm mode không tương thích với networking sử dụng external key-value store.**



Mỗi host trong network phải run một Docker Engine instance. Cách dễ dàng nhất để tạo ra các host là sử dụng Docker Machine.



Bạn nên open các ports của mỗi hosts.

| **Protocol** | **Port** | **Description** |
| --- | --- | --- |
| udp | 4789 | Data plane (VXLAN) |
| tcp/udp | 7946 | Control plane |

key-value store service có thể yêu cầu thêm một vài port khác. Check vendor’s documentation và open bất kỳ required port nào cần thêm.

**3. Tài liệu tham khảo**

* <https://docs.docker.com/engine/userguide/networking/>
* <https://docs.docker.com/engine/userguide/networking/work-with-networks/>

**Tìm hiểu và làm việc với docker container networks (P2)**

[Docker](https://viblo.asia/tags/docker)[DevOps](https://viblo.asia/tags/devops)

 Aug 30th, 2016 04:51:44

 97 2 0



**Trong bài viết trước chúng ta đã tìm hiểu về**[**các loại networking bên trong Docker native**](https://viblo.asia/euclid/posts/XqakEmmbkWK)**. Ở bài viết này chúng ta tiếp tục đi sâu vào các command của Docker Engine CLI giúp tương tác với Docker networks và containers bên trong chúng. Các command đó là:**

* docker network create
* docker network connect
* docker network ls
* docker network rm
* docker network disconnect
* docker network inspect

**1. Create networks**

Docker Engine tự động tạo ra bridge network sau khi nó được cài đặt. Network này tương ứng với docker0 bridge. Hơn nữa, với kiểu network này bạn có thể tự tạo ra **bridge network** của riêng mình hoặc **overlay network**.

Một bridge network chứa một single host chạy trên một instance của Docker Engine. Nếu bạn thực thi câu lệnh : docker network create và truyền vào một network name, Nó sẽ tạo ra một bridge network cho bạn.

$ docker network create simple-network

69568e6336d8c96bbf57869030919f7c69524f71183b44d80948bd3927c87f6a

$ docker network inspect simple-network

[

{

"Name": "simple-network",

"Id": "69568e6336d8c96bbf57869030919f7c69524f71183b44d80948bd3927c87f6a",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.22.0.0/16",

"Gateway": "172.22.0.1/16"

}

]

},

"Containers": {},

"Options": {}

}

]

**2. Connect containers**

Chúng ta có thể kết nối động các container tới một hoặc nhiều networks. Các networks có thể cùng giống hoặc khác network drivers. Với mỗi một kết nối, containers có thể giao tiếp với nhau thông qua container’s IP address hoặc container's name.

Với **overlay networks** hoặc **custom plugins** hỗ trợ multi-host connectivity, các containers kết nối tới cùng một multi-host network nhưng chạy trên các hosts khác nhau vẫn có thể giao tiếp với nhau qua cách này.

Hãy thử create 2 containers làm ví dụ:

$ docker run -itd --name=container1 busybox

18c062ef45ac0c026ee48a83afa39d25635ee5f02b58de4abc8f467bcaa28731

$ docker run -itd --name=container2 busybox

498eaaaf328e1018042c04b2de04036fc04719a6e39a097a4f4866043a2c2152

Sau đó create một isolated, bridge network để kiểm tra:

$ docker network create -d bridge --subnet 172.25.0.0/16 isolated\_nw

06a62f1c73c4e3107c0f555b7a5f163309827bfbbf999840166065a8f35455a8

Connect container2 vào network vừa được tạo và inspect network để xác nhận connection:

$ docker network connect isolated\_nw container2

$ docker network inspect isolated\_nw

[

{

"Name": "isolated\_nw",

"Id": "06a62f1c73c4e3107c0f555b7a5f163309827bfbbf999840166065a8f35455a8",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.25.0.0/16",

"Gateway": "172.25.0.1/16"

}

]

},

"Containers": {

"90e1f3ec71caf82ae776a827e0712a68a110a3f175954e5bd4222fd142ac9428": {

"Name": "container2",

"EndpointID": "11cedac1810e864d6b1589d92da12af66203879ab89f4ccd8c8fdaa9b1c48b1d",

"MacAddress": "02:42:ac:19:00:02",

"IPv4Address": "172.25.0.2/16",

"IPv6Address": ""

}

},

"Options": {}

}

]

Bạn có thể thấy rằng Engine tự động assigns một IP address vào container2. Cho chúng ta 1 --subnet khi network được tạo, Engine sẽ sinh một address từ subnet này. Ngay bây giờ hãy thử start container3 và connect nó tới network bằng command docker run --network option:

$ docker run --network=isolated\_nw --ip=172.25.3.3 -itd --name=container3 busybox

467a7863c3f0277ef8e661b38427737f28099b61fa55622d6c30fb288d88c551

Như bạn đã thấy, chúng ta có thể chỉ định ip address cho container. Có thể lựa chọn giữa IPv4 hoặc IPv6 address(es) cho container khi chạy câu lệnh docker run và docker network connect bằng việc truyền vào --ip và --ip6 flags cho IPv4 và IPv6. Selected IP address trở thành một phần của container networking configuration và sẽ được giữ nguyên ngay cả khi reload container. Tính năng này chỉ có đối với user defined networks, vì chúng đảm bảo subnets configuration không đổi đối với daemon reload.

Nào, hãy cùng inspect network resources được sử dụng bởi container3:

$ docker inspect --format='' container3

{"isolated\_nw":{"IPAMConfig":{"IPv4Address":"172.25.3.3"},"NetworkID":"1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",

"EndpointID":"dffc7ec2915af58cc827d995e6ebdc897342be0420123277103c40ae35579103","Gateway":"172.25.0.1","IPAddress":"172.25.3.3","IPPrefixLen":16,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:19:03:03"}}

Lặp lại câu lệnh này cho container2. Nếu máy bạn có cài python, có thể hiển thị đẹp hơn như sau:

$ docker inspect --format='' container2 | python -m json.tool

{

"bridge": {

"NetworkID":"7ea29fc1412292a2d7bba362f9253545fecdfa8ce9a6e37dd10ba8bee7129812",

"EndpointID": "0099f9efb5a3727f6a554f176b1e96fca34cae773da68b3b6a26d046c12cb365",

"Gateway": "172.17.0.1",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"IPAMConfig": null,

"IPAddress": "172.17.0.3",

"IPPrefixLen": 16,

"IPv6Gateway": "",

"MacAddress": "02:42:ac:11:00:03"

},

"isolated\_nw": {

"NetworkID":"1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",

"EndpointID": "11cedac1810e864d6b1589d92da12af66203879ab89f4ccd8c8fdaa9b1c48b1d",

"Gateway": "172.25.0.1",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"IPAMConfig": null,

"IPAddress": "172.25.0.2",

"IPPrefixLen": 16,

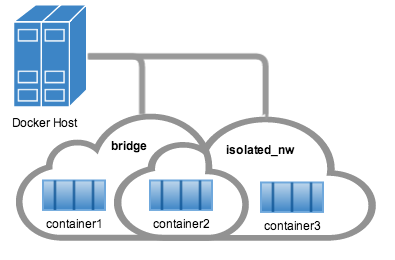
"IPv6Gateway": "",

"MacAddress": "02:42:ac:19:00:02"

}

}

Bạn sẽ thấy rằng container2 thuộc về 2 networks. **Bridge network** mặc định và **isolated\_nw**.



Với trường hợp của container3, Bạn đã kết nối thông qua docker run tới **isolated\_nw** nên nó ko kết nối tới bridge network mặc định.

Sử dụng câu lệnh docker attach để kết nối tới container2 đang khởi chạy và khám phá networking stack của nó, ta được:

$ docker attach container2

Phân tích network stack của container2, chúng ta sẽ thấy 2 Ethernet interfaces, 1 cho **default bridge** network và 1 cho **isolated\_nw** network.

/ # ifconfig

eth0 Link encap:Ethernet HWaddr 02:42:AC:11:00:03

inet addr:172.17.0.3 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1

RX packets:8 errors:0 dropped:0 overruns:0 frame:0

TX packets:8 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:648 (648.0 B) TX bytes:648 (648.0 B)

eth1 Link encap:Ethernet HWaddr 02:42:AC:15:00:02

inet addr:172.25.0.2 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe19:2/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:8 errors:0 dropped:0 overruns:0 frame:0

TX packets:8 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:648 (648.0 B) TX bytes:648 (648.0 B)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Trên **isolated\_nw**, Docker nhúng DNS server để resolve name cho các containers khác cùng mạng. Bên trong container2, ta có thể ping container3 bằng container's name.

/ *# ping -w 4 container3*

PING container3 (172.25.3.3): 56 data bytes

64 bytes from 172.25.3.3: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.3.3: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.3.3: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.3.3: seq=3 ttl=64 time=0.097 ms

*--- container3 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

Nếu bạn muốn, bạn có thể kết nối container1 với container2 bằng câu lệnh docker run --link, khi đó 2 container này có thể giao tiếp với nhau qua tên hoặc địa chỉ IP.

Detach container2 và vẫn để nó chạy bằng cách sử dụng tổ hợp phím CTRL-p CTRL-q.

Trong ví dụ này, container2 được kết nối tới cả 2 networks, do đó nó có thể tương tác với cả container1 và container3. Tuy nhiên, container3 và container1 thì ko cùng nằm trong 1 network và ko thể giao tiếp với nhau được. Để kiểm chức việc này, ta dùng lệnh ping tới contianer1 bên trong container 3:

$ docker attach container3

/ # ping 172.17.0.2

PING 172.17.0.2 (172.17.0.2): 56 data bytes

^C

--- 172.17.0.2 ping statistics ---

10 packets transmitted, 0 packets received, 100% packet loss

**Linking containers in user-defined networks**

Trong ví dụ trên, container2 có thể tự động resolve container3’s name trong **isolated\_nw** network, nhưng name resolution lại ko được thực hiện một cách tự động trong **default bridge** network. Sở dĩ có điều này là để duy trì khả năng tương thích ngược với các **legacy link** (tồn tại trước đó).

**legacy link** đưa ra 4 chức năng chính cho **default bridge** network:

* name resolution
* name alias cho các container đã được liên kết bằng cách sử dụng --link=CONTAINER-NAME:ALIAS
* secured container connectivity (in isolation via --icc=false)
* environment variable injection

So sánh 4 chức năng kể trên với **isolated\_nw** (non-default user-defined networks) trong ví dụ này, docker network cung cấp:

* name resolution tự động bằng DNS
* Tự động cô lập environment của các containers trong một network
* Khả năng attach và detach multiple networks
* Hỗ trợ --link option để cung cấp name alias cho các container đã liên kết

Tiếp tục ví dụ trên, chúng ta sẽ tạo ra container4 trong isolated\_nw với option --link tới name resolution sử dụng alias cho các containers khác cùng chung một network.

$ docker run --network=isolated\_nw -itd --name=container4 --link container5:c5 busybox

01b5df970834b77a9eadbaff39051f237957bd35c4c56f11193e0594cfd5117c

Nhờ có option --link mà từ giờ container4 có thể kết nối tới container5 với aliased name c5.

Bạn cần lưu ý một điều là khi tạo ra container4, chúng ta đã link tới container5 (thực tế còn chưa được tạo ra). Đây có thể coi là một sự khác biệt với **legacy link** trong **default bridge** network.

Bây giờ hãy thử chạy container5 link tới container4 với alias là c4.

$ docker run --network=isolated\_nw -itd --name=container5 --link container4:c4 busybox

72eccf2208336f31e9e33ba327734125af00d1e1d2657878e2ee8154fbb23c7a

Đúng như mong đợi, container4 sẽ kết nối được với container5 bằng cả container name và alias c5, ngược lại container5 sẽ kết nối được với container4 bằng cả container name và alias c4:

$ docker attach container4

/ *# ping -w 4 c5*

PING c5 (172.25.0.5): 56 data bytes

64 bytes from 172.25.0.5: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.5: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=3 ttl=64 time=0.097 ms

*--- c5 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

/ *# ping -w 4 container5*

PING container5 (172.25.0.5): 56 data bytes

64 bytes from 172.25.0.5: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.5: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=3 ttl=64 time=0.097 ms

*--- container5 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

$ docker attach container5

/ *# ping -w 4 c4*

PING c4 (172.25.0.4): 56 data bytes

64 bytes from 172.25.0.4: seq=0 ttl=64 time=0.065 ms

64 bytes from 172.25.0.4: seq=1 ttl=64 time=0.070 ms

64 bytes from 172.25.0.4: seq=2 ttl=64 time=0.067 ms

64 bytes from 172.25.0.4: seq=3 ttl=64 time=0.082 ms

*--- c4 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.065/0.070/0.082 ms

/ *# ping -w 4 container4*

PING container4 (172.25.0.4): 56 data bytes

64 bytes from 172.25.0.4: seq=0 ttl=64 time=0.065 ms

64 bytes from 172.25.0.4: seq=1 ttl=64 time=0.070 ms

64 bytes from 172.25.0.4: seq=2 ttl=64 time=0.067 ms

64 bytes from 172.25.0.4: seq=3 ttl=64 time=0.082 ms

*--- container4 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.065/0.070/0.082 ms

Cũng cần lưu ý một điều rằng nếu một container thuộc về nhiều networks, linked alias sẽ có phạm vi trong mạng đã cho. Do đó, các container có thể liên kết tới các alias khác nhau trên các mạng khác nhau.

Mở rộng ví dụ, chúng ta sẽ tạo 1 network khác có tên là **local\_alias**

$ docker network create -d bridge --subnet 172.26.0.0/24 local\_alias

76b7dc932e037589e6553f59f76008e5b76fa069638cd39776b890607f567aaa

Connect container4 và container5 tới mạng network local\_alias

$ docker network connect --link container5:foo local\_alias container4

$ docker network connect --link container4:bar local\_alias container5

$ docker attach container4

/ *# ping -w 4 foo*

PING foo (172.26.0.3): 56 data bytes

64 bytes from 172.26.0.3: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.26.0.3: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.26.0.3: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.26.0.3: seq=3 ttl=64 time=0.097 ms

*--- foo ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

/ *# ping -w 4 c5*

PING c5 (172.25.0.5): 56 data bytes

64 bytes from 172.25.0.5: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.5: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.5: seq=3 ttl=64 time=0.097 ms

*--- c5 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

**3. Tài liệu tham khảo**

* <https://docs.docker.com/engine/userguide/networking/>
* <https://docs.docker.com/engine/userguide/networking/work-with-networks/>

**Tìm hiểu và làm việc với docker container networks (P3)**

[Docker](https://viblo.asia/tags/docker)[DevOps](https://viblo.asia/tags/devops)

 Sep 5th, 2016 00:27:53

 108 2 0



**Trong bài viết trước chúng ta đã tìm hiểu về**[**cách kết nối các container trong docker network thông qua command**](https://viblo.asia/euclid/posts/XqakEmmbkWK)**. Ở bài viết này chúng ta tiếp tục tìm hiểu về cách ngắt kết nối container ra khỏi docker network, hoặc remove một network:**

**1. Network-scoped alias**

Trong khi \*link\*s cung cấp một private name giới hạn bên trong mỗi container thì **network-scoped alias** lại cung cấp một phương thức giúp các container được tìm ra bởi các tên thay thế bằng bất kỳ container nào khác bên trong phạm vi của mạng đặc thù. Không giống như **link alias**, được định nghĩa bởi các đối tượng sử dụng service, **network-scoped alias** được định nghĩa bởi chính container đưa ra service vào network.

Tiếp tục ví dụ ở phần 2, ta tạo một container khác bên trong isolated\_nw với một **network alias**.

$ docker run --network=isolated\_nw -itd --name=container6 --network-alias app busybox

8ebe6767c1e0361f27433090060b33200aac054a68476c3be87ef4005eb1df17

$ docker attach container4

/ *# ping -w 4 app*

PING app (172.25.0.6): 56 data bytes

64 bytes from 172.25.0.6: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.6: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=3 ttl=64 time=0.097 ms

*--- app ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

/ *# ping -w 4 container6*

PING container5 (172.25.0.6): 56 data bytes

64 bytes from 172.25.0.6: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.6: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=3 ttl=64 time=0.097 ms

*--- container6 ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

Bây giờ hãy thử connect container6 vào local\_alias network với một **network-scoped alias** khác.

$ docker network connect --alias scoped-app local\_alias container6

container6 trong ví dụ này có:

* alias là: app trong network isolated\_nw
* alias là: scoped-app trong network local\_alias.

Thử kết nối tới các aliases này từ container4 (Container được connect tới cả 2 networks) và container5 (Chỉ được connect tới isolated\_nw network).

$ docker attach container4

/ *# ping -w 4 scoped-app*

PING foo (172.26.0.5): 56 data bytes

64 bytes from 172.26.0.5: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.26.0.5: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.26.0.5: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.26.0.5: seq=3 ttl=64 time=0.097 ms

*--- foo ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

$ docker attach container5

/ *# ping -w 4 scoped-app*

ping: bad address 'scoped-app'

Như bạn có thể thấy, alias được giới hạn bên trong network mà nó định nghĩa và chỉ có các containers kết nối tới network có thể truy cập alias.

In addition to the above features, multiple containers can share the same network-scoped alias within the same network. For example, let’s launch container7 in isolated\_nw with the same alias as container6

Ngoài các tính năng trên, nhiều container có thể chia sẻ một **network-scoped alias** bên trong một network. Ví dụ, chúng ta hãy khởi động container7 trong isolated\_nw với alias như container6

$ docker run --network=isolated\_nw -itd --name=container7 --network-alias app busybox

3138c678c123b8799f4c7cc6a0cecc595acbdfa8bf81f621834103cd4f504554

Khi nhiều container cùng chia sẻ một alias, name resolution tới các alias sẽ xảy ra với một trong các container (thông thường sẽ là container đầu tiên được aliased). Khi một container liên kết tới alias bị tắt bỏ hoặc ngắt kết nối khỏi network, thì container tiếp đó liên kết tới alias sẽ được resolved.

Thử ping tới **alias app** từ container4 và tắt bỏ container6 để xác nhận container7 resolving **app alias**.

$ docker attach container4

/ *# ping -w 4 app*

PING app (172.25.0.6): 56 data bytes

64 bytes from 172.25.0.6: seq=0 ttl=64 time=0.070 ms

64 bytes from 172.25.0.6: seq=1 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=2 ttl=64 time=0.080 ms

64 bytes from 172.25.0.6: seq=3 ttl=64 time=0.097 ms

*--- app ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.070/0.081/0.097 ms

$ docker stop container6

$ docker attach container4

/ *# ping -w 4 app*

PING app (172.25.0.7): 56 data bytes

64 bytes from 172.25.0.7: seq=0 ttl=64 time=0.095 ms

64 bytes from 172.25.0.7: seq=1 ttl=64 time=0.075 ms

64 bytes from 172.25.0.7: seq=2 ttl=64 time=0.072 ms

64 bytes from 172.25.0.7: seq=3 ttl=64 time=0.101 ms

*--- app ping statistics ---*

4 packets transmitted, 4 packets received, 0% packet loss

round-trip min/avg/max = 0.072/0.085/0.101 ms

**2. Disconnecting containers**

Bạn có thể disconnect một container với network bằng cách sử dụng docker network disconnect command.

$ docker network disconnect isolated\_nw container2

$ docker inspect --format='' container2 | python -m json.tool

{

"bridge": {

"NetworkID":"7ea29fc1412292a2d7bba362f9253545fecdfa8ce9a6e37dd10ba8bee7129812",

"EndpointID": "9e4575f7f61c0f9d69317b7a4b92eefc133347836dd83ef65deffa16b9985dc0",

"Gateway": "172.17.0.1",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"IPAddress": "172.17.0.3",

"IPPrefixLen": 16,

"IPv6Gateway": "",

"MacAddress": "02:42:ac:11:00:03"

}

}

$ docker network inspect isolated\_nw

[

{

"Name": "isolated\_nw",

"Id": "06a62f1c73c4e3107c0f555b7a5f163309827bfbbf999840166065a8f35455a8",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.21.0.0/16",

"Gateway": "172.21.0.1/16"

}

]

},

"Containers": {

"467a7863c3f0277ef8e661b38427737f28099b61fa55622d6c30fb288d88c551": {

"Name": "container3",

"EndpointID": "dffc7ec2915af58cc827d995e6ebdc897342be0420123277103c40ae35579103",

"MacAddress": "02:42:ac:19:03:03",

"IPv4Address": "172.25.3.3/16",

"IPv6Address": ""

}

},

"Options": {}

}

]

Một container bị disconnected khỏi một network sẽ không thể nói chuyện với các containers khác đang connected tới network đó. Ví dụ, container2 sẽ ko thể nói chuyện với container3 trên isolated\_nw network.

$ docker attach container2

/ # ifconfig

eth0 Link encap:Ethernet HWaddr 02:42:AC:11:00:03

inet addr:172.17.0.3 Bcast:0.0.0.0 Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1

RX packets:8 errors:0 dropped:0 overruns:0 frame:0

TX packets:8 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:648 (648.0 B) TX bytes:648 (648.0 B)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

/ # ping container3

PING container3 (172.25.3.3): 56 data bytes

^C

--- container3 ping statistics ---

2 packets transmitted, 0 packets received, 100% packet loss

Trong khi container2 vẫn kết nối tới bridge network

/ *# ping container1*

PING container1 (172.17.0.2): 56 data bytes

64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.119 ms

64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.174 ms

^C

*--- container1 ping statistics ---*

2 packets transmitted, 2 packets received, 0% packet loss

round-trip min/avg/max = 0.119/0.146/0.174 ms

/ *#*

Ở một số trường hợp như khởi động lại Docker daemon trong mạng **multi-host network**(đa máy chủ), nơi mà các daemon là không thể cleanup kết nối cũ. Enpoint đó có thể gây ra lỗi "container already connected to network"khi có một container mới được kết nối với mạng có cùng tên với endpoint cũ. Để cleanup các điểm kết cuối cũ, ta cần force disconnect (docker network disconnect -f). Một khi các thiết bị đầu cuối được cleanup, các container có thể kết nối vào mạng.

$ docker run -d --name redis\_db --network multihost redis

ERROR: Cannot start container bc0b19c089978f7845633027aa3435624ca3d12dd4f4f764b61eac4c0610f32e: container already connected to network multihost

$ docker rm -f redis\_db

$ docker network disconnect -f multihost redis\_db

$ docker run -d --name redis\_db --network multihost redis

7d986da974aeea5e9f7aca7e510bdb216d58682faa83a9040c2f2adc0544795a

**3. Remove a network**

Khi tất cả các containers bên trong một network được stopped hoặc disconnected, bạn có thể gỡ bỏ network đó.

$ docker network disconnect isolated\_nw container3

docker network inspect isolated\_nw

[

{

"Name": "isolated\_nw",

"Id": "06a62f1c73c4e3107c0f555b7a5f163309827bfbbf999840166065a8f35455a8",

"Scope": "local",

"Driver": "bridge",

"IPAM": {

"Driver": "default",

"Config": [

{

"Subnet": "172.21.0.0/16",

"Gateway": "172.21.0.1/16"

}

]

},

"Containers": {},

"Options": {}

}

]

$ docker network rm isolated\_nw

Liệt kê lại tất cả các networks để xác nhận rằng isolated\_nw đã bị gỡ bỏ:

$ docker network ls

NETWORK ID NAME DRIVER

72314fa53006 host host

f7ab26d71dbd bridge bridge

0f32e83e61ac none null

**4. Tài liệu tham khảo**

* <https://docs.docker.com/engine/userguide/networking/>
* <https://docs.docker.com/engine/userguide/networking/work-with-networks/>