Networking in Docker

Default networks

When you install Docker, it creates three networks automatically. You can list these networks using the docker network ls command:

$ docker network ls

NETWORK ID NAME DRIVER

7fca4eb8c647 bridge bridge

9f904ee27bf5 none null

cf03ee007fb4 host host

These three networks are built into Docker. When you run a container, you can use the --network flag to specify which networks your container should connect to.

The bridge network represents the docker0 network present in all Docker installations. Unless you specify otherwise with the docker run --network=<NETWORK> option, the Docker daemon connects containers to this network by default. You can see this bridge as part of a host’s network stack by using the ip addr show command (or short form, ip a) on the host. (The ifconfig command is deprecated. It may also work or give you a command not founderror, depending on your system.)

$ ip addr show

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)

### **The default bridge network**

The default bridge network is present on all Docker hosts. If you do not specify a different network, new containers are automatically connected to the default bridge network.

The docker network inspect command returns information about a 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"

},

"Labels": {}

}

]

Run the following two commands to start two busybox containers, which are each connected to the default bridge network.

$ docker run -itd --name=container1 busybox

3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c

$ docker run -itd --name=container2 busybox

94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c

Inspect the bridge network again after starting two containers. Both of the busyboxcontainers are connected to the network. Make note of their IP addresses, which will be different on your host machine than in the example below.

$ 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"

},

"Labels": {}

}

]

* Containers connected to the default bridge network can communicate with each other by IP address.
* **Docker does not support automatic service discovery on the default bridge network. If you want containers to be able to resolve IP addresses by container name, you should use user-defined networks instead**.
* You can link two containers together using the legacy docker run --link option, but this is not recommended in most cases and this is depreciated.
* You can attach to a running container to see how the network looks from inside the container. You are connected as root, so your command prompt is a # character.

$ docker attach container1

root@3386a527aa08:/# ip -4 addr

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue qlen 1

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

633: eth0@if634: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

inet 172.17.0.2/16 scope global eth0

valid\_lft forever preferred\_lft forever

From inside the container, use the ping command to test the network connection to the IP address of the other container.

root@3386a527aa08:/# 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

Use the cat command to view the /etc/hosts file on the container. This shows the hostnames and IP addresses the container recognizes.

root@3386a527aa08:/# 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

To detach from the container1 container and leave it running, use the keyboard sequence **CTRL-p CTRL-q**. If you wish, attach to container2 and repeat the commands above.

The default docker0 bridge network supports the use of port mapping and docker run --linkto allow communications among containers in the docker0 network. This approach is not recommended. Where possible, you should use [user-defined bridge networks](https://docs.docker.com/engine/userguide/networking/#user-defined-networks) instead.

## User-defined networks

* It is recommended to use user-defined bridge networks to control which containers can communicate with each other, and also to enable automatic DNS resolution of container names to IP addresses.
* Docker provides default **network drivers** for creating these networks. You can create a new **bridge network**, **overlay network** or **MACVLAN network**.
* You can also create a **network plugin** or **remote network** for complete customization and control.
* You can create as many networks as you need, and you can connect a container to zero or more of these networks at any given time.
* In addition, you can connect and disconnect running containers from networks without restarting the container.
* When a container is connected to multiple networks, its external connectivity is provided via the first non-internal network, in lexical order.

Create a User defined bridge 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": {},

"Labels": {}

}

]

$ docker network ls

NETWORK ID NAME DRIVER

9f904ee27bf5 none null

cf03ee007fb4 host host

7fca4eb8c647 bridge bridge

c5ee82f76de3 isolated\_nw bridge

After you create the network, you can launch containers on it using thedocker 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": ""

}

},

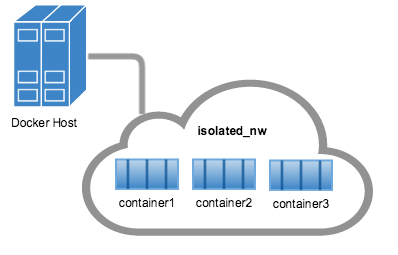
"Options": {},

"Labels": {}

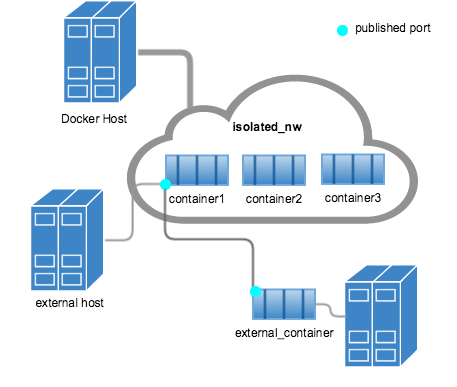
}

]

The containers you launch into this network must reside on the same Docker host. Each container in the network can immediately communicate with other containers in the network. Though, the network itself isolates the containers from external networks.



Within a user-defined bridge network, linking is not supported. You can [expose and publish container ports](https://docs.docker.com/engine/userguide/networking/#exposing-and-publishing-ports) on containers in this network. This is useful if you want to make a portion of thebridge network available to an outside network.



the docker daemon implements an embedded DNS server which provides built-in service discovery for any container created with a valid name or net-alias or aliased by link. The exact details of how Docker manages the DNS configurations inside the container can change from one Docker version to the next. So you should not assume the way the files such as /etc/hosts, /etc/resolv.conf are managed inside the containers and leave the files alone and use the following Docker options instead.

Various container options that affect container domain name services.

|  |  |
| --- | --- |
| --name=CONTAINER-NAME | Container name configured using --name is used to discover a container within an user-defined docker network. The embedded DNS server maintains the mapping between the container name and its IP address (on the network the container is connected to). |
| --network-alias=ALIAS | In addition to --name as described above, a container is discovered by one or more of its configured --network-alias (or --alias in docker network connect command) within the user-defined network. The embedded DNS server maintains the mapping between all of the container aliases and its IP address on a specific user-defined network. A container can have different aliases in different networks by using the --alias option in docker network connectcommand. |
| --link=CONTAINER\_NAME:ALIAS | Using this option as you run a container gives the embedded DNS an extra entry named ALIAS that points to the IP address of the container identified by CONTAINER\_NAME. When using --link the embedded DNS will guarantee that localized lookup result only on that container where the --link is used. This lets processes inside the new container connect to container without having to know its name or IP. |
| --dns=[IP\_ADDRESS...] | The IP addresses passed via the --dns option is used by the embedded DNS server to forward the DNS query if embedded DNS server is unable to resolve a name resolution request from the containers. These --dns IP addresses are managed by the embedded DNS server and will not be updated in the container's /etc/resolv.conffile. |
| --dns-search=DOMAIN... | Sets the domain names that are searched when a bare unqualified hostname is used inside of the container. These --dns-search options are managed by the embedded DNS server and will not be updated in the container's /etc/resolv.conf file. When a container process attempts to access host and the search domain example.com is set, for instance, the DNS logic will not only look up host but also host.example.com. |
| --dns-opt=OPTION... | Sets the options used by DNS resolvers. These options are managed by the embedded DNS server and will not be updated in the container's /etc/resolv.conf file.  See documentation for resolv.conf for a list of valid options. |

In the absence of the --dns=IP\_ADDRESS..., --dns-search=DOMAIN..., or --dns-opt=OPTION...options, Docker uses the /etc/resolv.conf of the host machine (where the docker daemon runs). While doing so the daemon filters out all localhost IP address nameserver entries from the host’s original file.

Filtering is necessary because all localhost addresses on the host are unreachable from the container’s network. After this filtering, if there are no more nameserver entries left in the container’s /etc/resolv.conf file, the daemon adds public Google DNS nameservers (8.8.8.8 and 8.8.4.4) to the container’s DNS configuration.