### What is Docker Networking ?

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an ifconfig on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host.

* $ ifconfig
* you will see docker0 Interface.

For Docker containers to communicate with each other and the outside world via the host machine, there has to be a layer of networking involved. Docker supports different types of networks, each fit for certain use cases.

For example, building an application which runs on a single Docker container will have a different network setup as compared to a web application with a cluster with database, application and load balancers which span multiple containers that need to communicate with each other. Additionally, clients from the outside world will need to access the web application container

### Ping containers in default network

Create two containers, by default it will get assigned bridge network. IP address are assigned in the range of gateway [docker0 interface]

$ docker exec container1 bash -c “ping container2ipadd”

$ docker exec container2 bash -c “ping container1ipadd”

**Assign A static IP to a container**

You can only assign IP addresses to the containers using networks that you create [user defined networks]

$ docker network create -d bridge --subnet 172.40.0.0/16--gateway 172.40.0.1 <networkName>

$ docker runt -dti --network networkname --name <contianerName> centos

The container will get random Ip address

For static IP

$ docker runt -dti --network networkname –ip=172.40.0.100 --name <contianerName> centos

**HostNetwork**

It’s a default network that you cannot delete. It uses driver bridge. Basically when you create a container using the host network, it will be exactly the same thing in network in terms as your docker hosts.

$ docker run --rm -ti –network host centos bash

root#

You will see that your container inherit pretty much everything from your docker host. Type **hostname** then you will have exactly same hostname as docker host. And if you take a look at your DNS, add

**None Network**

It’s a container without network

### Docker Network Types

Docker Networking subsystem is pluggable using drivers.

* Bridge
* Host
* Overlay
* Macvlan
* None

### What is Default Network in Docker [docker0]

When Docker is installed, a default bridge network named docker0 is created. Each new Docker container is automatically attached to this network, unless a custom network is specified.

Besides docker0 , two other networks get created automatically by Docker: host (no isolation between host and containers on this network, to the outside world they are on the same network) and none (attached containers run on container-specific network stack).

### Docker Network ls

List Networks

API 1.21+ The client and daemon API must both be at least 1.21 to use this command. Use the docker version command on the client to check your client and daemon API versions.

***Syntax****$ docker network ls # List all networks*

Options

* --filter , -f Provide filter values (e.g. 'driver=bridge')
* --format Pretty-print networks using a Go template
* --no-trunc Do not truncate the output
* --quiet , -q Only display network IDs

**Examples**

**List all networks**

$ sudo docker network ls

NETWORK ID NAME DRIVER SCOPE

7fca4eb8c647 bridge bridge local

9f904ee27bf5 none null local

cf03ee007fb4 host host local

78b03ee04fc4 multi-host overlay swarm

**Filters Example**

The currently supported filters are:

* driver
* id (network’s id)
* label (label=<key> or label=<key>=<value>)
* name (network’s name)
* scope (swarm|global|local)
* type (custom|builtin)

The driver filter matches networks based on their driver.

* $ docker network ls --filter driver=bridge

The id filter matches on all or part of a network’s ID

* $ docker network ls --filter id=63d1ff1f77b07ca51070a8c227e962238358bd310bde1529cf62e6c307ade161

The label filter matches networks based on the presence of a label alone or a label and a value.

* $ docker network ls -f "label=usage"

The following filter matches networks with the usage label with the prod value.

* $ docker network ls -f "label=usage=prod"

The name filter matches on all or part of a network’s name.

* $ docker network ls --filter name=foobar

The scope filter matches networks based on their scope.

* $ docker network ls --filter scope=swarm
* $ docker network ls --filter scope=local

The type filter supports two values; builtin displays predefined networks (bridge, none, host), whereas custom displays user defined networks.

* $ docker network ls --filter type=custom

By having this flag it allows for batch cleanup. For example, use this filter to delete all user defined networks:

* $ docker network rm `docker network ls --filter type=custom -q`

Formating

$ docker network ls --format "{{.ID}}: {{.Driver}}"

### Docker Network Inspect

* Display detailed information on one or more networks
* API 1.21+ The client and daemon API must both be at least 1.21 to use this command. Use the docker version command on the client to check your client and daemon API versions.
* Returns information about one or more networks. By default, this command renders all results in a JSON object.

***Syntax****$ docker network inspect [OPTIONS] NETWORK [NETWORK...]*

*--format , -f Format the output using the given Go template*

*--verbose , -v Verbose output for diagnostics*

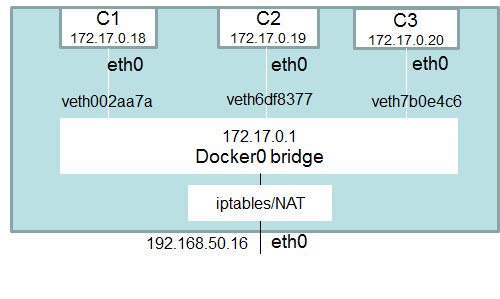
**Example**

Inspecting Bridge Network. It shows all the info reated to bridge. Display all contianers mapped to bridge and subnet , IP etc.

$ sudo docker network inspect bridge

### Bridge Network Drive

* bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. **Bridge networks are usually used when your applications run in standalone containers that need to communicate.**
* In terms of networking, a bridge network is a Link Layer device which forwards traffic between network segments. A bridge can be a hardware device or a software device running within a host machine’s kernel.
* In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network. The Docker bridge driver automatically installs rules in the host machine so that containers on different bridge networks cannot communicate directly with each other.
* Bridge networks apply to containers running on the **same** Docker daemon host. For communication among containers running on different Docker daemon hosts, you can either manage routing at the OS level, or you can use an [overlay network](https://docs.docker.com/network/overlay/).
* When you start Docker, a [default bridge network](https://docs.docker.com/network/bridge/#use-the-default-bridge-network) (also called bridge) is created automatically, and newly-started containers connect to it unless otherwise specified. You can also create user-defined custom bridge networks. **User-defined bridge networks are superior to the default bridge network.**



* Three container c1, c2, c3 will have the IP address depends upon the subnet associated with bridge network docker0 bridge. Docker0 bridge act as a gateway. The container want to connect with internet typically needs to go with Docker0 bridge and iptables and etho interface. C1 c2 c3 can also able to communicate each other automcatically under bridge network.
* Contianer within each other or container to host the traffic shloud go through docker0 bridge interface. Verification. Login to the container.
* $ route -n
* Destinatoin Gateway Genmask Flags Metric Ref Use Inter

### Create User Defined Bridge [docker network create]

$ docker network ls

$ docker network create –driver bridge mybridge

This will create the mybridge user defined network. Also It creates the interfaces.

$ ifconfig

Docker0 : this interface for dfault bridge

Br-fcb…: This is the interface ceated and associated with mybridge network

Asscoiate the contianet to the mybridge nwtork

$ docker container run -dt –name contianername1 –network mybridge ubutu

$ docker container run -dt –name contianername2 –network mybridge ubutu

This will create two new interfaces under mybridge network.

$ brctl show

It displays bridge name along with the associated interfaces.

BridgeName BridgeID Interfaces

Br-fcb…: veth…. & veth…

Docker0

Create a network

API 1.21+ The client and daemon API must both be at least 1.21 to use this command. Use the docker version command on the client to check your client and daemon API versions.

***Syntax****$ docker network create [OPTIONS] NETWORK*

**Extended description**

* Creates a new network. The DRIVER accepts bridge or overlay which are the built-in network drivers. If you have installed a third party or your own custom network driver you can specify that DRIVER here also.
* When you launch a new container with docker run it automatically connects to this bridge network. You cannot remove this default bridge network, but you can create new ones using the network create command.

$ docker network create -d bridge my-bridge-network

* Bridge networks are isolated networks on a single Engine installation. If you want to create a network that spans multiple Docker hosts each running an Engine, you must create an overlay network

**Options**

* --attachable Enable manual container attachment
* --aux-address Auxiliary IPv4 or IPv6 addresses used by Network driver
* --config-from The network from which to copy the configuration
* --config-only Create a configuration only network
* --driver , -d bridge Driver to manage the Network
* --gateway IPv4 or IPv6 Gateway for the master subnet
* --ingress Create swarm routing-mesh network
* --internal Restrict external access to the network
* --ip-range Allocate container ip from a sub-range
* --ipam-driver IP Address Management Driver
* --ipam-opt Set IPAM driver specific options
* --ipv6 Enable IPv6 networking
* --label Set metadata on a network
* --opt , -o Set driver specific options
* --scope Control the network's scope
* --subnet Subnet in CIDR format that represents a network segment

**Example**

Create bridge network

* sudo docker network create --driver bridge alpine-net
* View the network list
* sudo docker network ls
* View the alpine-net network details
* sudo docker network inspect alpine-net

Create 3 container with:

* alpine1 container connect to default bridge network
* alpine2 container connect to alpine-net network
* alpine3 container connect to network default bridge and alpine-net
* sudo docker run -dit --name alpine1 alpine ash
* sudo docker run -dit --name alpine2 --network alpine-net alpine ash
* sudo docker run -dit --name alpine3 alpine ash
* sudo docker network connect alpine-net alpine3

View network bridge details

* sudo docker network inspect bridge
* view network alpine-net details
* sudo docker network inspect apline-net

Enter the alpine3 container and ping alpine1 ip, alpine1 and alpine2 name

* sudo docker attach alpine3
* ping IP alpine1
* ping -c 3 172.17.0.2
* ping name alpine1
* ping -c 3 alpine1
* ping alpine2 name
* ping -c 3 alpine2

Enter the alpine2 and ping to alpine1 IP and ping to the internet

* # ping -c 3 172.17.0.2
* failed, because different bridge network and subnet
* ping internet will be succeed.
* # ping -c 3 8.8.8.8

**$ docker network create \**

**--driver=bridge \**

**--subnet=172.28.0.0/16 \**

**--ip-range=172.28.5.0/24 \**

**--gateway=172.28.5.254 \**

**br0**

### Diff B/W User def Bridge and Def Bridge

**User-defined bridges provide better isolation and interoperability between containerized applications.**

Containers connected to the same user-defined bridge network automatically expose all ports to each other, and no ports to the outside world. This allows containerized applications to communicate with each other easily, without accidentally opening access to the outside world.

**User-defined bridges provide automatic DNS resolution between containers.**

Containers on the default bridge network can only access each other by IP addresses, unless you use the --link option, which is considered legacy. On a user-defined bridge network, containers can resolve each other by name or alias.

**Containers can be attached and detached from user-defined networks on the fly.**

During a container’s lifetime, you can connect or disconnect it from user-defined networks on the fly. To remove a container from the default bridge network, you need to stop the container and recreate it with different network options.

**Each user-defined network creates a configurable bridge.**

User-defined bridge networks are created and configured using docker network create. If different groups of applications have different network requirements, you can configure each user-defined bridge separately, as you create it.

**Linked containers on the default bridge network share environment variables.**

Containers connected to the same user-defined bridge network effectively expose all ports to each other. For a port to be accessible to containers or non-Docker hosts on different networks, that port must be published using the -p or --publish flag. source: <https://docs.docker.com/network/bridge/>

<https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/>

### Host Networks

* host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly. See use the host network.
* If you use the host network mode for a container, that container’s network stack is not isolated from the Docker host (the container shares the host’s networking namespace), and the container does not get its own IP-address allocated. For instance, if you run a container which binds to port 80 and you use host networking, the container’s application is available on port 80 on the host’s IP address.

Overview of bridge network

A bridge network uses a software bridge which allows containers conntected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge.

If you want the container in bridge network to monitor docker host eth0 interface, the it is not possible.Lets say you want to deploy a service called intrusion detection systems , you cannot install into a container associate with bridge network bcz it wil not monitor the etho . In this use case host network will be useful.

$ docker container run -dt –name myhost –network host ubuntu

$ docker container exec -it myhost bash

#netstat -ntlp

This will show all the ports of container as well as docker host post

* Host mode networking can be useful to optimize performance, and in situations where a container needs to handle a large range of ports, as it does not require network address translation (NAT), and no “userland-proxy” is created for each port.
* The host networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server.
* It is a public network
* It utilizes the host’s IP address and TCP port space to display the services running inside the container
* It effective

$ docker run --rm -d --network host --name my\_nginx nginx

**Note: Given that the container does not have its own IP-address when using host mode networking,**[**port-mapping**](https://docs.docker.com/network/overlay/#publish-ports)**does not take effect, and the -p, --publish, -P, and --publish-all option are ignored, producing a warning instead:**

### None Network

In this network driver, the Docker containers will neither have any access to external networks nor will it be able to communicate with other containers

This mode will not configure any IP for the container and doesn’t have any access to the external network as well as other container.

This option is used when a user wants to disable the networking access to a container

In simple terms, None is called a loopback interface, which means it has no external network interfaces

$ docker run --rm -d --network none --name my\_nginx nginx

### Docker Network Create

Little bit restricted in the default network because first you can’t ping containers by using the name & second you cannot control the ranges, subnets, the numbers, the IP addresses. You cannot control anything , you can just add containers to the existing network.

$ docker network create [OPTIONS] NETWORK

Options

* --attachable Enable manual container attachment
* --aux-address Auxiliary IPv4 or IPv6 addresses used by Network driver
* --config-from The network from which to copy the configuration
* --config-only Create a configuration only network
* --driver , -d bridge Driver to manage the Network
* --gateway IPv4 or IPv6 Gateway for the master subnet
* --ingress Create swarm routing-mesh network
* --internal Restrict external access to the network
* --ip-range Allocate container ip from a sub-range
* --ipam-driver IP Address Management Driver
* --ipam-opt Set IPAM driver specific options
* --ipv6 Enable IPv6 networking
* --label Set metadata on a network
* --opt , -o Set driver specific options
* --scope Control the network's scope
* --subnet Subnet in CIDR format that represents a network segment

Example

$ docker network create \

--driver=bridge \

--subnet=172.28.0.0/16 \

--ip-range=172.28.5.0/24 \

--gateway=172.28.5.254 \

br0

**Create your own network**

* First thing that you need to have in mind is that we need to provide a driver to create a network.
* Check the subnet of the default network bcz we don’t want to overlap, don’t want to use same subnet.
  + $ docker network inspect bridge

$ docker network create <NetworkName> -d bridge –-subnet 172.18.0.0/16 –-gateway 172.18.0.0

Gateway shld be inside the subnet

$ docker network ls

Embedded DNS! Ping container by name in the same network

$ docker network create <NetworkName> -d bridge –-subnet 172.18.0.0/16 –-gateway 172.18.0.0

Create a container in a new network

$ docker container run -dti –-network NetworkName –name container3 centos

In own network we can able to ping the container using its name.

$ docker exec container1 bash -c “ping container2”

### Docker Network Connect/Disconnect

Two container having different network will not get communicated by default. So you can use Docker network connect command to connect the one network to another so you can able to ping each container then disconnect it.

$ docker network connect [OPTIONS] NETWORK CONTAINER

Options

* --alias Add network-scoped alias for the container
* --driver-opt driver options for the network
* --ip IPv4 address (e.g., 172.30.100.104)
* --ip6 IPv6 address (e.g., 2001:db8::33)
* --link Add link to another container
* --link-local-ip Add a link-local address for the container

**Example**

**Connect container in different networks**

$ docker network connect <network1> <network2ContianerName>

It successfully connect now you can able to ping the containers in network2.

Basiclly we connected or assigned the network2 contianer to network 1

$ docker exec net1\_container bash -c “ping net2\_cont”

$ docker network disconnect <network1> <network2ContianerName>

**Connect a running container to a network**

$ docker network connect multi-host-network container1

**Connect a container to a network when it starts**

You can also use the docker run --network=<network-name> option to start a container and immediately connect it to a network.

$ docker run -itd --network=multi-host-network busybox

**Specify the IP address a container will use on a given network**

You can specify the IP address you want to be assigned to the container’s interface.

$ docker network connect --ip 10.10.36.122 multi-host-network container2

**Use the legacy --link option**

You can use --link option to link another container with a preferred alias

$ docker network connect --link container1:c1 multi-host-network container2

**Docker network disconnect**

Disconnects a container from a network. The container must be running to disconnect it from the network.

$ docker network disconnect [OPTIONS] NETWORK CONTAINER

-f force

$ docker network disconnect multi-host-network container1

### Docker Network Prune

Remove all unused networks. Unused networks are those which are not referenced by any containers.

$ docker network prune [OPTIONS]

* --filter Provide filter values (e.g. 'until=<timestamp>')
* --force , -f Do not prompt for confirmation

**Example**

$ docker network prune

WARNING! This will remove all custom networks not used by at least one container.

$ docker network prune --force --filter until=5m

### Docker Network rm

Removes one or more networks by name or identifier. To remove a network, you must first disconnect any containers connected to it.

$ docker network rm NETWORK [NETWORK...]

**Example**

$ docker network rm my-network

$ docker network rm 3695c422697f my-network