Networking in Docker

```
Networking in Docker:
=====================
   Containers present unique challenges when it comes to networking.
Docker includes multiple built-in solutions to these networking
challenges.
   Docker implements container networking using a framework called the
Container Networking Model (CNM) and manages the networking for
containers.

The CNM utilizes the following concepts:

   Sandbox: An isolated unit containing all networking components
associated with a single container. Usually a Linux network namespace.

   Endpoint: Connects a sandbox to a network. Each sandbox/container
can have any number of endpoints, but has exactly one endpoint for each
network it is connected to.

   Network: A collection of endpoints connected to one another.

   Network Driver: Handles the actual implementation of the CNM
concepts.

   IPAM Driver: IPAM means IP Address management. Automatically
allocates subnets and IP Addresses for networks and endpoints.

Network Drivers:
================
   Docker includes several built-in network drivers, know as Native
Network Drivers.
   These network drivers implement the concepts described in the CNM.

   The Native Network Drivers are:
     1) host
     2) bridge
     3) overlay
     4) macvlan
     5) none

   with docker run we can use --net flag to attach network driver to
container(s).

   The Host Network Driver:
   ------------------------
      The Host Network Driver allows containers to use the host's
network stack direclty.
       1) Containers use the host's networking resources direclty
```

```
             2) No sandboxes, all containers on the host using the hsot
driver share the same network namespace
       3) no two containers can use the same port(s)
    UseCases: Simple and easy setup, one or only few containers on a
single host.

   The Bridge Network Driver:
   --------------------------
     The Bridge Network Driver uses Linux bridge networks to provice
connectivity between containers on the same host.
       1) This is the default driver for containers running on a single
host (i.e, not in a swarm)
       2) Creates a Linux Bridge for each Docker Network
       3) Creates a default Linux bridge network called docker0.
Containers automatically connect to this if no other network is
specified
     UseCases: isolated networking among containers ona single host.

   The Overlay Network Driver:
   ---------------------------
     The Overlay Network Driver provides connectivity between
containers across multiple Docker hosts, i.e. with Docker swarm.
       1) Uses a VXLAN data plane, which allows the underlying network
infrastructure (underlay) to route data between hosts in a way that is
transparent to the containers themselves.
       2) Automatically configures network interfaces, bridges, etx. on
each hosts as needed.
     UseCases: Networking between containers in a swarm

   The macvlan Network Driver:
   ---------------------------
     The macvlan Network Driver offers a more lightweight
implementation by connecting container interfaces directly to host
interfaces.
       1) Uses direct association with Linux interfaces instead of a
bridge interface.
       2) Harder to configure and greater dependency between macvlan
and the external network.
       3) More lightweight and less latency.
     UseCases: When there is a need for extremely low latency, or a
need for containers with IP addresses in teh external subnet.

   The None Network Driver:
   ------------------------
     The None Network Driver does not provide any networking
implementation.
       1) Container is completely isolated from other containers and
the host.
       2) if you want networking with the None driver, you must set
everything up manually.
```

3) None does create a separate networking namespace for each container, but no interfaces or endpoints.

UseCases: When there is no need for container networking or you want to set all of the networking up yourself.

Managing Networks:
==================

We can create and manager our own networks with the "docker network" commands. if we do not specify a network driver, bridge will be used by default.

```
docker newtork ls
docker network create NETWORK ( create a bridge network by default )
docker network create --driver bridge NETWORK
docker network create --driver overlay NETWORK
docker network inspect NETWORK
docker network rm NETWORK

docker network connect NETWORK CONTAINER
docker network disconnect NETWORK CONTAINER
```

Embedded DNS:
=============

Docker networks implements an embedded DNS server, allowing containers and services to locate and communicate with one another.

Containers can communicate with other containers and services using the serice or container name, or network alias.

```
docker run --network-alias ALIAS
docker network connect --alias ALIAS
```

Example:

Create a container with a network alias and communicate with it from another container using both the name and the alias.

```
docker network create my-net
docker run -d --name my-net-nginx --network my-net --network-alias my-nginx-alias nginx
docker exec my-net-busybox curl my-net-nginx2:80
docker exec my-net-busybox curl my-nginx-alias:80
```

Create a container and provide a network alias with the docker network connect command.

```
docker run -d --name my-net-nginx2 nginx
docker network connect --alias another-alias my-net my-net-nginx3
docker exec my-net-busybox curl another-alias:80
```

Publishing Ports for Services:
==============================

```
Host vs. Ingress
    Docker Swarm supports two modes for publishing ports for services.

  Ingress:
    1) The default, used if no mode is specified.
    2) Uses a routing mesh. The published port listens on every node in
the cluster, and trasparently directs incoming traffic to any task that
is part of the service, on any node.

    publish a service port host mode:
      docker service create -p 8081:80 --name nginx_ingress_pub nginx

  Host:
    1) Publishes the port directly on the host where a task is running.
    2) cannot have multiple replicas on the same node if you use a
static port.
    3) Traffic to the published port on the node goes directly to the
task running on that specific node.

    publish a service port host mode:
      docker service create -p mode=host,published=8082,target=80 --
name nginx_host_pub nginx
```