Networking for Docker containers

Stefano Nicotri (INFN Bari)
nicotri@infn.it

Outline

- Prelude: some basic facts about networking
- Networking in Docker containers
- Networking Drivers
  - bridge
  - overlay
  - ipvlan and macvlan
  - none
  - third-party
- Challenge

Prelude: some basic facts about networking

IP address

An Internet Protocol address (IP address) is a numerical label such as 192.0.2.1 that is connected to a computer network that uses the Internet Protocol for communication.

An IP address serves two main functions: network interface identification, and location addressing.

IP addresses are described as consisting of **two groups of bits** in the address: the most significant bits are the **network prefix**, which identifies a whole network or subnet, and the least significant set forms the **host identifier**, which specifies a particular interface of a host on that network.

IP address

In the classful network architecture of IPv4, the three most significant bits of the 32-bit IP address define the size of the network prefix, and determine the network class A, B, or C (here we ignore D and E classes, but they exist)

Class	Most-significant bits	Network prefix size (bits)	Host identifier size (bits)	Address range
A	0	8	24	0.0.0.0–127.255.255.255
B	10	16	16	128.0.0.0–191.255.255.255
C	110	24	8	192.0.0.0–223.255.255.255

IP address

In practice:

an IP address is a string in the form **a.b.c.d** where a,b,c,d are numbers between 0 and 255

example: **192.168.0.9**

there are ~ 4 billion IP addresses (IPv4)

Classless Inter-Domain Routing (CIDR) notation

CIDR notation is a compact representation of an IP address and its associated network mask, which specifies an IP address, a slash ('/') character, and a decimal number, which is the count of consecutive leading 1-bits (from left to right) in the network mask. Each 1-bit denotes a bit of the address range which must remain identical to the given IP address.

The address may denote a specific interface address (including a host identifier, such as 10.0.0.1/8), or it may be the beginning address of an entire network (using a host identifier of 0, as in 10.0.0.0/8 or its equivalent 10/8).

CIDR notation can even be used with no IP address at all, e.g. when referring to a /24 as a generic description of an IPv4 network that has a 24-bit prefix and 8-bit host numbers.

Classless Inter-Domain Routing (CIDR) notation

In practice:

in a a.b.c.d/x network you have 2^{32-x} addresses, starting from a.b.c.d

some examples:

1. 192.168.13.0**/24** means $2^{32-24} = 2^8$ = 256 IPs, from 192.168.13.0 to 192.168.13/255
2. 192.168.13.5**/32** means $2^{32-32} = 2^0$ = **one** IP
3. 10.0.0.0**/8** means $2^{32-8} = 2^{24}$ = 16,777,216 IPs, from 10.0.0.0 to 10.255.255.255
4. 172.16.0.0**/12** means $2^{32-12} = 2^{20}$ = 1,048,576 IPs, from 172.16.0.0 to 172.31.255.255
5. 192.168.0.0**/16** means $2^{32-16} = 2^{16}$ = 65,536 IPs, from 192.168.0.0 to 192.168.255.255

Private network

A private network is a computer network that uses a private address space of IP addresses. These addresses are commonly used for local area networks (LANs)

Private network addresses are not allocated to any specific organization

IP packets originating from or addressed to a private IP address cannot be routed through the public Internet

Private network: IP addresses

The following IPv4 address ranges are **reserved** for private networks:

1. **10.0.0.0/8** means $2^{32-8} = 2^{24}$ = 16,777,216 IPs, from 10.0.0.0 to 10.255.255.255
2. **172.16.0.0/12** means $2^{32-12} = 2^{20}$ = 1,048,576 IPs, from 172.16.0.0 to 172.31.255.255
3. **192.168.0.0/16** means $2^{32-16} = 2^{16}$ = 65,536 IPs, from 192.168.0.0 to 192.168.255.255

Networking in Docker containers

Some of the questions we want to answer here are:

how do I connect two containers running on my host?

how do I isolate my container?

how do I access a specific port?

how do I connect containers running on different hosts?

how do I connect an external (possibly non-containerized) service to my container?

Networking in Docker containers

One powerful feature of Docker is its large variety of networking possibilities for containers.

It is possible to connect containers together, or connect them to non-Docker workloads, and services do not need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not.

The Docker networking system makes use of **drivers**, which provide different functionalities

Docker driver summary

User-defined **bridge** networks are best when you need multiple containers to communicate on the same Docker host.

Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.

Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.

IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing

Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.

Third-party network plugins allow you to integrate Docker with specialized network stacks.

Bridge networking driver

The **`bridge`** network driver is the default networking driver (the one you end up using if you don't specify any driver), and apply to containers running on the **same** Docker daemon host

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**

Docker automagically creates rules (e.g. via iptables) in the host machine to achieve such result

Bridge networking driver

When Docker is started, a default bridge network is created, called **`bridge`**, and all containers connect to it unless otherwise specified. As a result, all such containers can communicate with each other, but are not accessible from the outside

Each container has its own IP address on the **`bridge`**

It is also possible to create *user-defined* custom bridge networks, with interesting features

User-defined bridge networks: automatic DNS resolution

Containers on the default bridge network can only access each other by IP addresses

On a user-defined bridge network, containers can resolve each other by name or alias.

For example, if you call your containers **web** and **db**, the web container can connect to the db container at **db**, without needing to specify (or even know) its IP address

User-defined bridge networks: better isolation

Having all our containers attached to the default **`bridge`** network can be risky, as unrelated stacks/services/containers are then able to communicate, and a vulnerability in a service can affect unrelated services

Using a user-defined network provides a scoped network in which only containers attached to that network are able to communicate among each other

User-defined bridge networks: attachability

It is possible to connect or disconnect containers from user-defined networks on the fly (without stopping them), while this is not possible with the default **`bridge`**

To remove a container from the default bridge network, you need to stop the container and recreate it with different network options

Bridge network driver: usage

create the **my-net** bridge

```
$  docker network create my-net
```

create a container attached to the **my-net** bridge, **with a port exposed on the host**

```
$  docker create --name my-nginx --network my-net --publish 8080:80 nginx:latest
```

in this way, the port 8080 on the host will be mapped to the port 80 of the container (the syntax is **host_port:container_port**), and it will be possible to access the service exposed by the container directly from the host

Bridge network driver: usage

connect a container (here called **my-container**) to the **my-net** bridge:

```
$  docker network connect my-net my-container
```

disconnect it:

```
$  docker network disconnect my-net my-container
```

Host network driver

The **`host`** driver is mainly used for standalone containers

It removes network isolation between the container and the Docker host, and uses the host's networking directly.

If you run a container using this driver it **does not get its own IP-address** allocated

example: if you run a container which binds to port 80, the container's application is available on port 80 **on the host**'s IP address, without needing to publish any port

Host network driver: use cases

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 proxy is created for each port.

The host networking driver **only works on Linux**, and is not supported on Mac and/or Windows

Host network driver: usage

The host network driver is used passing the **`--network host`** option when creating a container

```
$  docker run --rm -d --network host --name my_nginx nginx
```

Overlay network driver

Overlay networks **connect multiple Docker daemons** together

The **`overlay`** network driver creates a distributed network among **multiple Docker daemon hosts**. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate

Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.

Overlay network driver: use cases

Overlay networks can be used to connect standalone containers living on different Docker hosts or connect a swarm service and a standalone container

Overlay networks support encryption (**not** on Windows) for secure communications among containers/swarms

Overlay network driver: usage

create the **my-overlay-net** overlay network

```
$ docker network create -d overlay my-overlay-net
```

use the **--attachable** flag to make it usable to standalone containers

```
$ docker network create --driver overlay --attachable my-overlay-net
```

Overlay network driver: encryption

It is possible to enforce data encryption adding the **`--opt encrypted`** flag when creating the overlay network. This enables IPSEC encryption at the level of the vxlan. This encryption imposes a non-negligible performance penalty, so you should test this option before using it in production.

```
$ docker network create --opt encrypted --driver overlay --attachable my-net
```

ipvlan and macvlan network driver

the **`ipvlan`** network driver give users total control over IPv4 and IPv6 addressing

The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing

Some applications (e.g. applications which monitor network traffic) expect to be directly connected to the physical network instead. In this type of situation, you can use the **`macvlan`** network driver to assign a MAC address to each container's virtual network interface, making it appear to be a physical network interface directly connected to the physical network

Used for legacy applications or if you need your containers to look like physical hosts on your network

“none” network driver

The last possibility is to disable networking features for the container altogether, using the **none** network driver

This is usually used in conjunction with a custom network driver

Third party network driver

As for storage drivers, multiple third-party plugins are available for networking as well, to integrate Docker with specialized network stacks

Thank you for your attention