Virtualizing a network with Docker

Why docker?

Docker containers and services are so powerful that you can connect them together, or connect them to non-Docker workloads. Docker containers and services do not even need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not. Whether your Docker hosts run Linux, Windows, or a mix of the two, you can use Docker to manage them in a platform-agnostic way.

Containers instead of virtualizing the underlying hardware, they virtualize the operating system so each container contains only the application and its libraries. This helps in development stages especially when we are creating a protocol.

Why VrNetLab?

Vrnetlab creates Docker images for each type of router that will run in the virtual network. It packages the router’s disk image together with KVM software, Python scripts, and any other resources required by the router into the Docker image.

### **Pros Of Container**

* Containers can be as small as 10MB and you can easily limit their memory and CPU usage. So, they are lightweight.
* Since they are small in size, they can boot up faster and can be quickly scaled too.
* Containers are exemplary when it comes to **Continuous Integration and Continuous Deployment** (CI/CD) implementation.

### **Cons Of Container**

* Since the containers run on host OS, it has a dependency on the host underlying host Operating System.
* Containers cannot all by themselves cannot provide security at a commendable level.
* When the container is deleted if the data inside the container is lost. You will have to add Data Volumes in order to store the data.

#### **Docker Network Types**

There are four different types of network configurations for containers. Listed from most secure/restrictive to most open/unrestrictive.

1. Closed
2. Bridged
3. Joined
4. Open

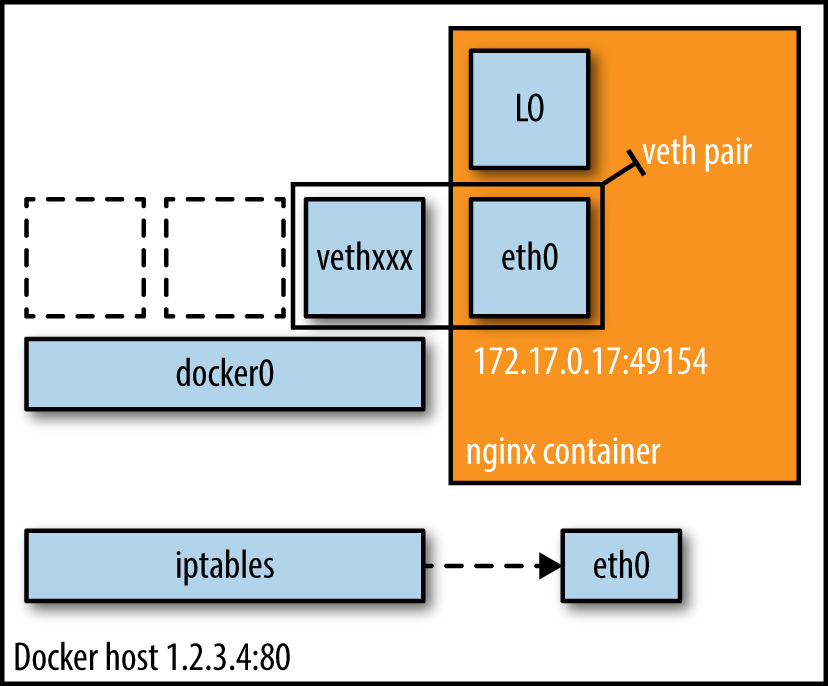
## **Bridge Mode Networking**

In this mode (see [Figure 2](https://www.oreilly.com/content/what-is-docker-networking/#fig-ch2-docker-bridge-mode)), the Docker daemon creates docker0, a virtual Ethernet bridge that automatically forwards packets between any other network interfaces that are attached to it. By default, the daemon then connects all containers on a host to this internal network through creating a pair of peer interfaces, assigning one of the peers to become the containers eth0 interface and other peer in the namespace of the host, as well as assigning an IP address/subnet from the [private IP range](https://tools.ietf.org/html/rfc1918) to the bridge ([Example 1](https://www.oreilly.com/content/what-is-docker-networking/#example-bridge-mode-networking)).

###### **Example 1. Docker bridge mode networking in action**

$ docker run -d -P --net=bridge nginx:1.9.1$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED  
STATUS PORTS NAMES  
17d447b7425d nginx:1.9.1 nginx -g 19 seconds ago  
Up 18 seconds 0.0.0.0:49153->443/tcp, 0.0.0.0:49154->80/tcp trusting\_feynman

###### **Note** Because bridge mode is the Docker default, you could have equally used docker run -d -P nginx:1.9.1 in [Example 1](https://www.oreilly.com/content/what-is-docker-networking/#example-bridge-mode-networking). If you do not use -P (which publishes all exposed ports of the container) or -p host\_port:container\_port (which publishes a specific port), the IP packets will not be routable to the container outside of the host.



Helpful Links:

<https://www.brianlinkletter.com/vrnetlab-emulate-networks-using-kvm-and-docker/>

<https://openwrt.org/docs/guide-user/virtualization/virtualbox-vm>

<https://openwrt.org/docs/guide-user/virtualization/docker_host>

<https://openwrt.org/docs/guide-user/network/openwrt_as_routerdevice>

<https://k21academy.com/docker-kubernetes/docker-vs-virtual-machine/>

<https://netbeez.net/blog/networking-with-docker/>

<https://netcraftsmen.com/docker-networking/>

<https://www.oreilly.com/content/what-is-docker-networking/>

<https://docs.docker.com/network/>