ISSN No. 0976-5697

Volume 8, No. 5, May – June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Basic Analysis of Docker Networking

Samyukta S Hegde
M.Tech, Department of Information Science
BMS College of Engineering
Bangalore, India

Dr. P Jayarekha
Associate Professor, Department of Information Science
BMS College of Engineering
Bangalore, India

Abstract: Containerization is virtualization at operating system level as opposed to full machine virtualization. Docker is a tool that is used to create, ship and run applications inside containers. A single host can have multiple containers running on it. The containers may have to communicate with each other and by using docker networking, this can be accomplished. In this paper we explore docker networking and analyze how it is based on the concept of Linux network namespaces. We will use Cisco Packet Tracer to implement a sample network. Then the same network will be implemented in Linux using ip tool. The same network will then be implemented using docker and the relationship between network namespaces and docker networking will be established.

Keywords: docker; Linux; network; namespaces; containers

I. Introduction

Containerization is a lightweight alternative to complete OS virtualization. It eliminates the need of hypervisor and virtual machines. It basically encapsulates a run time environment. The Linux features used are namespaces and control groups. To achieve containerization, docker is used.

This paper demonstrates the analogy between the real world networks and the docker networking. As docker networking is based on Linux network namespaces, the namespace concept and its implementation will also be touched upon. The following will be demonstrated:

- A simple network will be set up in Cisco packet tracer.
- The same network will be implemented in network namespace
- Docker will be used to represent the network scenario

II. Literature Survey

Docker is platform for building, shipping and running applications. It does not contain the whole operating system, but just the necessary binaries and libraries. It basically embodies a run time environment. It is run on user space on top of the OS. There is no concept of hypervisor. Namespaces facilitate the container to have its own network, IP etc. Control groups manage the usage of memory and CPU [4]. It is a platform for developing, shipping and running applications. It is a bit like the virtual machine. But the whole OS is not created. Docker uses the host machine's kernel. It is lightweight and reduces the size. The host machine is of no concern.

Images can be pulled from the repositories or can be created by users (e.g. by using Dockerfiles). Dockerfiles can be used to build images. Docker registries are utilized for the purpose of storing the images.

Docker containers get created when images are run. Data volumes can also be shared between containers. The containers/microservices (services that communicate with each over a network (service oriented architecture)) can be connected to each other using a network.

III. Project Setup

A. *Network Setup using Cisco Packet Tracer*

- First Cisco Packet Tracer is opened.
- Two PCs, 2 generic switches and one router are inserted.
- PC0 is connected to Switch0 and PC1 to Switch1 using copper Straight-through cable through Fast Ethernet ports.
- Switch0 and Switch1 are connected to Router0 using copper Straight-through cable through Fast Ethernet ports fa0/0 and fa1/0 respectively.
- The network setup looks as shown in Fig 1.
- Configuration of hosts is done as given in Table I

Table I. IP Address Configurations

Host.	*Interface Fa0/0*	*Interface Fa0/1*
PC0	172.19.1.2/24	
PC1	172.20.1.2/24	
Router0	172.19.1.1/24	172.20.1.1/24

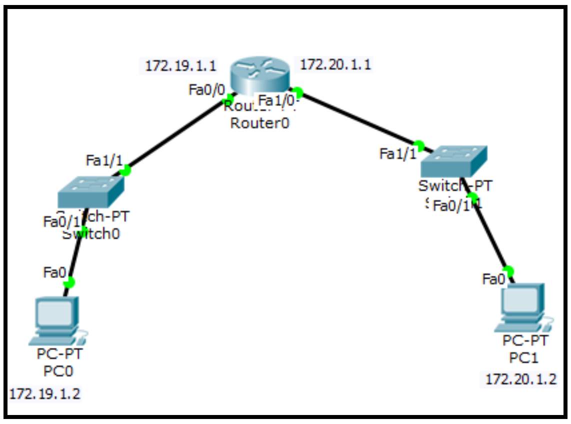


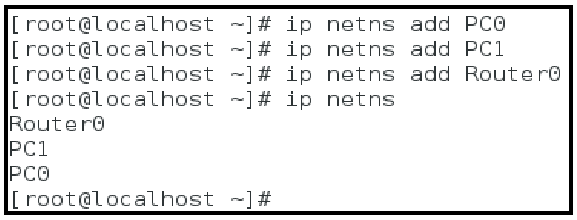
Fig. 1. Simple network designed in Cisco Packet Tracer

- Now, on pinging PC0 from PC1 and vice versa, the ping succeeds and the connectivity is correct.

B. *Network Setup in Linux Network Namespaces*

The OS used in this demonstration is Centos 7. After logging into a Linux box, two nodes are to be created using namespaces [1]. They are to be connected to a router which in turn is another namespace. The two nodes belong to separate networks. The aim is to enable connectivity between the two nodes (i.e. namespaces) [2].

- The commands used for the creation of the namespaces PC0, PC1 and Router0 and viewing them is shown in Fig 2.



```
[root@localhost ~]# ip netns add PC0
[root@localhost ~]# ip netns add PC1
[root@localhost ~]# ip netns add Router0
[root@localhost ~]# ip netns
Router0
PC1
PC0
[root@localhost ~]#
```

Fig. 2. Creation and Listing of network namespaces using ip tool

- The entries of the created namespaces are visible in the folder /var/run/netns/ (Fig 3).

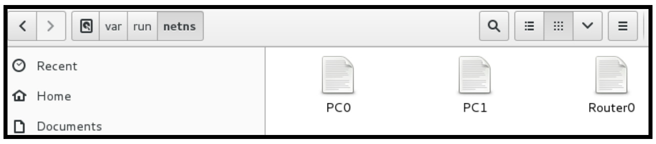
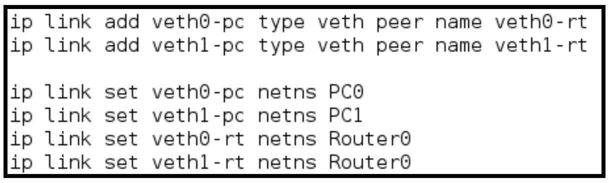


Fig. 3. /var/run/netns folder

- Virtual Ethernet (veth) pairs are created for connectivity [2]. There will be two veth pairs for both the PCs to connect to the Router. A veth pair is just like a pipe. The information sent from one end comes out from the other end [3]. Fig 4 displays the commands to be run to create veth pairs and bind them to respective namespaces.

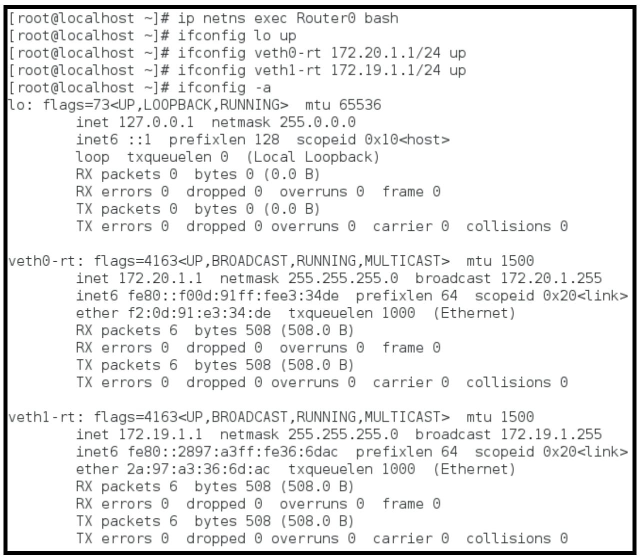


```
ip link add veth0-pc type veth peer name veth0-rt
ip link add veth1-pc type veth peer name veth1-rt

ip link set veth0-pc netns PC0
ip link set veth1-pc netns PC1
ip link set veth0-rt netns Router0
ip link set veth1-rt netns Router0
```

Fig. 4. Creation of virtual ethernet pairs and binding to namespaces

- The next step is to log into each network namespace, set the ip as in TABLE I and bring up the lo and veth interfaces. The commands for Router0 are shown in Fig 5. PC0 and PC1 also have to be configured in the same way.



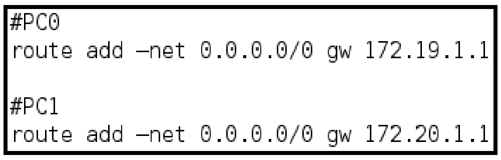
```
[root@localhost ~]# ip netns exec Router0 bash
[root@localhost ~]# ifconfig lo up
[root@localhost ~]# ifconfig veth0-rt 172.20.1.1/24 up
[root@localhost ~]# ifconfig veth1-rt 172.19.1.1/24 up
[root@localhost ~]# ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth0-rt: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.1.1  netmask 255.255.255.0  broadcast 172.20.1.255
        inet6 fe80::f00d:91ff:fee3:34de  prefixlen 64  scopeid 0x20<link>
        ether f2:0d:91:e3:34:de  txqueuelen 1000  (Ethernet)
        RX packets 6  bytes 508 (508.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 508 (508.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1-rt: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.19.1.1  netmask 255.255.255.0  broadcast 172.19.1.255
        inet6 fe80::2897:a3ff:fe36:6dac  prefixlen 64  scopeid 0x20<link>
        ether 2a:97:a3:36:6d:ac  txqueuelen 1000  (Ethernet)
        RX packets 6  bytes 508 (508.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 508 (508.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig. 5. Interface configuration of Router0 namespace

- The namespaces are not able to ping each other as there is no route added. On adding the ip route [2], by logging into each namespace, a static route is established (Fig 6).



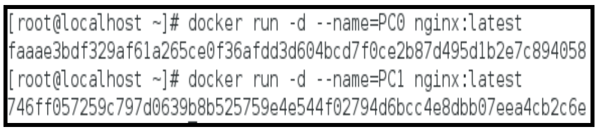
```
#PC0
route add -net 0.0.0.0/0 gw 172.19.1.1

#PC1
route add -net 0.0.0.0/0 gw 172.20.1.1
```

Fig. 6. Commands to add static routes

C. *Host Setup in Docker Containers*

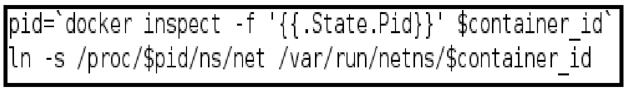
- The first task is to pull an image from the docker hub. The image used here is nginx:latest. Two containers are created by using the docker run command. They are named as PC0 and PC1 (Fig 7).



```
[root@localhost ~]# docker run -d --name=PC0 nginx:latest
faaae3bdf329af61a265ce0f36afdd3d604bcd7f0ce2b87d495d1b2e7c894058
[root@localhost ~]# docker run -d --name=PC1 nginx:latest
746ff057259c797d0639b8b525759e4e544f02794d6bcc4e8dbb07eea4cb2c6e
```

Fig. 7. Running nginx containers with host names PC0 and PC1

- The running of two images creates two network namespaces. But these, by default are not listed in the /var/run/netns folder. So the symlink has to be manually created to play around with networks [8]. First the pid of the docker containers is got and then a symlink is created as displayed in Fig 8.



```
pid=`docker inspect -f '{{.State.Pid}}' $container_id`
ln -s /proc/$pid/ns/net /var/run/netns/$container_id
```

Fig. 8 Creation of symlink

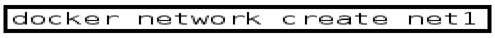
- It is done for both the containers and then when the command "ip netns" is run, both the namespaces are visible.
- The next task is to log into PC0 using the docker exec command. The ifconfig command is run in order to check the interfaces. The same thing is done with container PC1. As the network is not specified during

container creation, both PC0 and PC1 are connected to the same default bridge network. So if they ping each other, they can communicate with each other.

D. *Network Setup in Docker Containers*

1) Method 1:

Two networks are created by names net1 and net2. net1 is created as shown in Fig 9. net2 is also created in the same way [5].



```
docker network create net1
```

Fig. 9. Creation of a docker network

If networks are created by docker, it implies that, in reality, network bridges are created. These can be checked by running the command "brctl show."

Initially the two newly created bridges will have no interfaces listed. On association of both the containers with the networks created, the interfaces for the bridges will be listed [6].

The two containers belong to different networks and cannot ping each other. On running the "brctl show" command, the interfaces for the bridges are listed. It is because when a container is attached to a network, a veth pair is created. One end is inside the container and the other end is in the bridge.

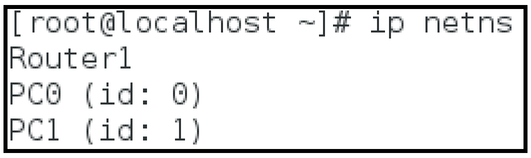
In order for both the containers to ping each other, another network is created i.e net3 [7]. Both the containers are connected to it using the docker connect command.

Now both the containers are connected to net3. This leads to the creation of two veth pairs. One end of both resides in each container and the other end in the net3 bridge. The ip and brctl commands can be run to check the veth pairs in the containers and the bridge respectively. Now both PC0 and PC1 each belong to two networks. As both are connected to net3, the containers can ping each other.

2) Method 2:

In order to make the containers in different networks to communicate with each other, the ip tool needs to be used to manipulate the interfaces.

The existing containers are removed. Two new containers are run with names PC0 and PC1. A new network namespace Router1 is created. There are three network namespaces now (The method to make the namespaces of containers visible is described in Section C of Project Setup). On doing so, Fig 10, lists the network namespaces using the "ip netns" command.



```
[root@localhost ~]# ip netns
Router1
PC0 (id: 0)
PC1 (id: 1)
```

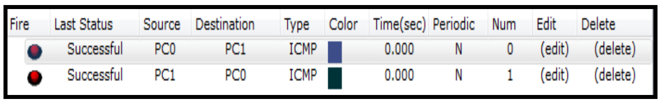
Fig. 10. List of network namespaces

A new network namespace Router1 is created. The containers PC0, PC1 and namespace Router1 are configured to have ip addresses 172.19.1.2/24, 172.20.1.2/24 and router interfaces addresses 172.19.1.1/24 and 172.20.1.1/24. The addresses are assigned to the namespaces by following the same steps as in Section C of Project Setup.

On doing this, it has been made possible to enable two containers PC0 and PC1 in different networks to communicate with each other using a third network namespace Router1 (which acts as a real world router).

IV. Results and Discussions

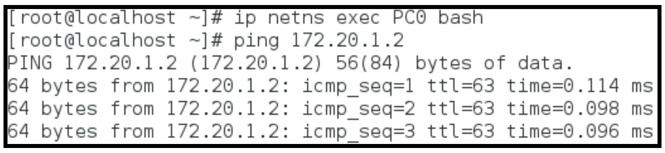
This section showcases the results of different network setups performed in the previous section. Fig 11. is a proof of successful connectivity between PC0 and PC1 in Cisco Packet Tracer.



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC1	ICMP	■	0.000	N	0	(edit)	(delete)
●	Successful	PC1	PC0	ICMP	■	0.000	N	1	(edit)	(delete)

Fig. 11 Successful ping in Cisco Packet Tracer

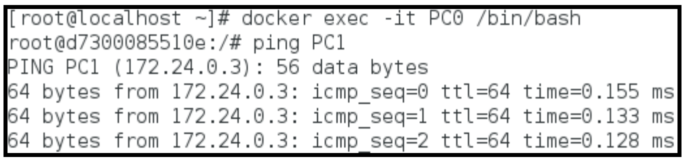
Fig 12 shows the connectivity between PC0 and PC1 for the method followed in Section B of Project Setup.



```
[root@localhost ~]# ip netns exec PC0 bash
[root@localhost ~]# ping 172.20.1.2
PING 172.20.1.2 (172.20.1.2) 56(84) bytes of data.
64 bytes from 172.20.1.2: icmp_seq=1 ttl=63 time=0.114 ms
64 bytes from 172.20.1.2: icmp_seq=2 ttl=63 time=0.098 ms
64 bytes from 172.20.1.2: icmp_seq=3 ttl=63 time=0.096 ms
```

Fig. 12 Successful ping in Cisco Packet Tracer

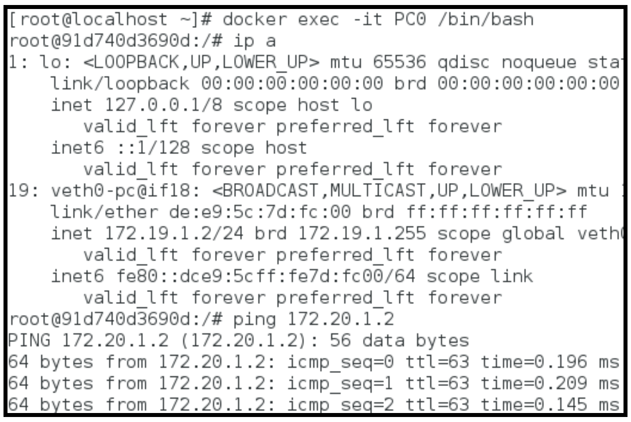
On association of containers PC0 and PC1 to network net3 as explained in Method 1 of Section D in Project Setup, they are able to ping each other, as shown in Fig 13.



```
[root@localhost ~]# docker exec -it PC0 /bin/bash
root@d7300085510e:/# ping PC1
PING PC1 (172.24.0.3): 56 data bytes
64 bytes from 172.24.0.3: icmp_seq=0 ttl=64 time=0.155 ms
64 bytes from 172.24.0.3: icmp_seq=1 ttl=64 time=0.133 ms
64 bytes from 172.24.0.3: icmp_seq=2 ttl=64 time=0.128 ms
```

Fig. 13 Successful ping from PC0 to PC1 on association with network net3

The containers belonging to different networks are able to ping each other because of their binding to Router1 network namespace (Method 2 of Section D in Project Setup). Fig 14 displays the connectivity. Similar results are observed when PC0 is pinged from PC1



```
[root@localhost ~]# docker exec -it PC0 /bin/bash
root@91d740d3690d:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue sta
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
19: veth0-pc@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu
    link/ether de:e9:5c:7d:fc:00 brd ff:ff:ff:ff:ff:ff
    inet 172.19.1.2/24 brd 172.19.1.255 scope global veth
       valid_lft forever preferred_lft forever
    inet6 fe80::dce9:5cff:fe7d:fc00/64 scope link
       valid_lft forever preferred_lft forever
root@91d740d3690d:/# ping 172.20.1.2
PING 172.20.1.2 (172.20.1.2): 56 data bytes
64 bytes from 172.20.1.2: icmp_seq=0 ttl=63 time=0.196 ms
64 bytes from 172.20.1.2: icmp_seq=1 ttl=63 time=0.209 ms
64 bytes from 172.20.1.2: icmp_seq=2 ttl=63 time=0.145 ms
```

Fig. 14 Successful ping from PC0 to PC1 on association with network namespace Router1

V. Conclusion

From the study of network namespaces, we conclude that the approach followed is helpful in having a basic understanding about the concept of networking in docker. The containers and their networking can further be explored in many different ways. Some of the areas of exploration are subnetting, NAT etc.

VI. ACKNOWLEDGMENT

I express my heartfelt gratitude to my guide Dr. P Jayarekha for her guidance and encouragement. During the entire duration of this project, she supported and encouraged me in every aspect. Her valuable suggestions have helped me in completing the project successfully.

VII. REFERENCES

[1] Simulate a LAN with Linux Network NameSpaces by Chandan available at https://www.youtube.com/watch?v=cWKQ7YtZUTk

[2] Simulate a Router with Linux Network NameSpaces by Chandan, available at https://www.youtube.com/watch?v=TlGdOx80Pqc

[3] Introducing Linux Network Namespaces, Scotts Weblog, available at http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/

[4] Cgroups, namespaces, and beyond: what are containers made from? with Jérôme Petazzoni, Tinkerer Extraordinaire, Docker, available at https://www.youtube.com/watch?v=sK5i-N34im8

[5] Docker container networking, docker docs, available at https://docs.docker.com/engine/userguide/networking/

[6] Customize the docker0 bridge, docker docs, available at https://docs.docker.com/engine/userguide/networking/default_network/custom-docker0/

[7] Stackoverflow Blog post on “Communicating between Docker containers in different networks on the same host” available at http://stackoverflow.com/questions/36035595/communicating-between-docker-containers-in-different-networks-on-the-same-host

[8] StackOverflow post available at http://stackoverflow.com/questions/31265993/docker-networking-namespace-not-visible-in-ip-netns