Report: Container Virtualization

Group 16: You Hu 12036234, Wenchen Lai 12482625

Docker is a software program that helps to manages applicants using containers. A docker container is a standalone process and provides all needed source code and dependencies for an applicant. Containers similar to virtual machines but more portable and lightweight. Docker builds an image from a Dockerfile and creates a new container over a specified image.

Test available bandwidth between docker containers using iperf

We create two VMs (VM1 and VM2) and install docker in both of them to deploy and run containers. At first, we are not at the right direction for this task. As Figure 1 shows, when a container communicate with another one at the same host, they will expose their port to the host and use host network to communicate. So Figure1(b) shows “172.17.0.1 port 50088” (host’s IP) “connected with 172.17.0.2 port 5001” instead of 172.17.0.4 which is the IP of client container. So does Figure 2. We can learn that containers are totally isolated. Then if we require them to communicate directly, we need to use zerotier p2p system. We install zerotier in each container and make them join the same network. Figure 3 shows the result, the bandwidth between docker containers is 4.78Mbits/sec, though the result is also affected by many factors such as zerotier limitation.

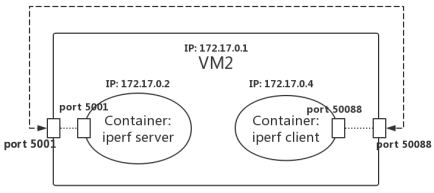

Figure1(a): two containers communication at same host

```
root@Ubuntu:/# iperf -c 172.17.0.2 -t 60 -l 8k -i 10
------------------------------------------------------------
Client connecting to 172.17.0.2, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 172.17.0.1 port 50088 connected with 172.17.0.2 port 5
001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  5.02 GBytes  4.31 Gbits/sec
[  3] 10.0-20.0 sec  5.12 GBytes  4.40 Gbits/sec
[  3] 20.0-30.0 sec  5.03 GBytes  4.32 Gbits/sec
[  3] 30.0-40.0 sec  5.25 GBytes  4.51 Gbits/sec
[  3] 40.0-50.0 sec  5.17 GBytes  4.44 Gbits/sec
[  3] 50.0-60.0 sec  18.0 GBytes  15.5 Gbits/sec
[  3]  0.0-60.0 sec  43.6 GBytes  6.25 Gbits/sec
```

Figure1(b): iperf result

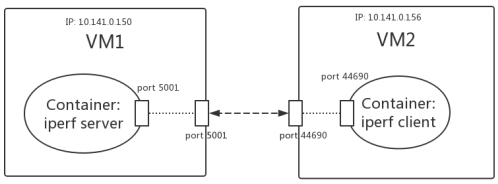

Figure2(a): two containers communication using host ip at different host

```
root@Ubuntu:/# iperf -c 10.141.0.150 -t 60 -l 8k -i 10
------------------------------------------------------------
Client connecting to 10.141.0.150, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.141.0.156 port 44690 connected with 10.141.0.15
0 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.00 GBytes   860 Mbits/sec
[  3] 10.0-20.0 sec  1.04 GBytes   897 Mbits/sec
[  3] 20.0-30.0 sec  1.08 GBytes   927 Mbits/sec
[  3] 30.0-40.0 sec  1.01 GBytes   871 Mbits/sec
[  3] 40.0-50.0 sec  1.05 GBytes   902 Mbits/sec
[  3] 50.0-60.0 sec  1.02 GBytes   877 Mbits/sec
[  3]  0.0-60.0 sec  6.21 GBytes   889 Mbits/sec
```

Figure2(b): iperf result

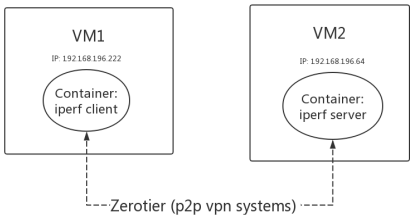

Figure3(a): two containers communicate directly using zerotier p2p system

```
root@Ubuntu:/# iperf -c 192.168.196.64 -t 60 -l 8k -i 10
------------------------------------------------------------
Client connecting to 192.168.196.64, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.196.222 port 37662 connected with 192.168.196.6
4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  5.23 MBytes  4.38 Mbits/sec
[  3] 10.0-20.0 sec  4.94 MBytes  4.15 Mbits/sec
[  3] 20.0-30.0 sec  7.42 MBytes  6.22 Mbits/sec
[  3] 30.0-40.0 sec  5.30 MBytes  4.45 Mbits/sec
[  3] 40.0-50.0 sec  9.95 MBytes  8.35 Mbits/sec
[  3] 50.0-60.0 sec  3.91 MBytes  3.28 Mbits/sec
[  3]  0.0-64.4 sec  36.7 MBytes  4.78 Mbits/sec
```

Figure3(b): iperf result

Test broadcasting messages between docker containers using socat

We deploy socat server at VM2 with command *socat UDP4-RECVFROM:6666, broadcast,fork EXEC:"echo received"* . Whatever broadcasting messages the socat server receive, it will return “received” message. We run socat client at VM1. Both client container and server container join the same network at zerotier system, then they can communicate with each other. And broadcast address is 192.168.196.255 port 6666. As Figure 4 shows, docker containers can send broadcasting messages to each other.

```
root@Ubuntu:/# socat STDIO UDP4-DATAGRAM:192.168.196.255:6666,broad
cast,range=192.168.196.255/24
hi
received
hellp
received
```

Figure4 broadcast testing

Deploy URL Shortener Web service

In our Dockerfile for url shortener web service, we use RUN instruction to pre-install necessary packages (such as python-pip, flask) when building an image. CMD instruction will run by default when a container is launched from an image. EXPOSE instruction is used for publishing ports. Sending files from host to images using ADD instruction. We need to notice that a Dockerfile only has one CMD instruction. So if we need to join our zerotier network and run our service, it will be more than one command and we can add a script file to do that. The result (Figure 5) shows we can access to our url shortener web service from containers on same host or on remote host.

```
root@23ac66d5406a:/# curl http://192.168.196.65:5000
['1']root@23ac66d5406a:/#
```

Figure5: remote host