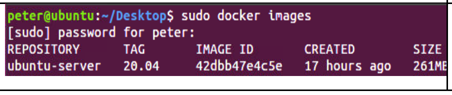
HW1: visualization

1. **Creation of three different disk images**

   The three disk images are as follows.

Disk Image	Result	Commands for Creation
QEMU-qcow2	ubuntu.img	**sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.6-live-server-amd64.iso -m 2046 -boot strict=on**
QEMU-raw	ubuntu_raw.raw	**sudo qemu-system-x86_64 -drive format=raw, file=ubuntu_raw.raw -boot d -cdrom ./ubuntu-20.04.6-live-server-amd64.iso -m 2046 -boot strict=on**
Container	 peter@ubuntu:~/Desktop$ sudo docker images [sudo] password for peter: REPOSITORY TAG IMAGE ID CREATED SIZE ubuntu-server 20.04 42dbb47e4c5e 17 hours ago 261MB	**sudo docker pull ubuntu:20.04**

2. **Present main steps to enable a QEMU VM. In addition, please present the detailed QEMU commands, and VM configurations: 10 points**

   - Steps to enable a QEMU VM

     Step1: install QEMU on a platform like Linux or window

     Step2: download 20.04 ubuntu server operating system.

     Step3: build a VM with the command mentioned below.

     Step4: start the VM with the command mentioned below.

   - Commands required

QEMU-qcow2	**1. sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.6-live-server-amd64.iso -m 2046** **2. sudo qemu-system-x86_64 -hda ubuntu.img**
QEMU-raw	**1. sudo qemu-system-x86_64 -drive format=raw, file=ubuntu_raw.raw -boot d -cdrom ./ubuntu-20.04.6-live-server-amd64.iso -m 2046 -boot strict=on** **2. sudo qemu-system-x86_64 -driv format=raw,file=ubuntu_raw.raw**

3. **Present main steps to create the Docker container. This must show your steps in creating your own image and your image history! Do not copy any classmate's image. In addition, please describe the operations you use to manage Docker containers (and some other operations which you think are also important): 10 points**

   - Steps to create a container

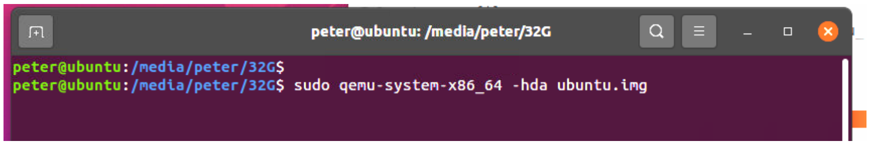
Step1: install docker on a platform like Linux or window
Step2: pull a ubuntu server image for docker hub.
Step3: run the image
Step4: install the tools such as sysbench, SSH, sshpass.
Step5: exit the image, and then save the container as a new image named ubnutu-server:20.04
Step6: start the new image.

- Commands required

Docker image	1. Pull an image from hub **sudo docker image pull ubuntu:20.04** 2. Run the image **sudo run -it ubuntu:20.04 /bin/bash** 3. Install the tools on the image **spt-get install sysbench** **spt-get install ssh** **spt-get install sshpass** 4. Exit the image 5. Show all active containers **sudo docker ps -a** 6. Create a new image with the active container created by the original image **sudo docker commit container_id new_name_image**

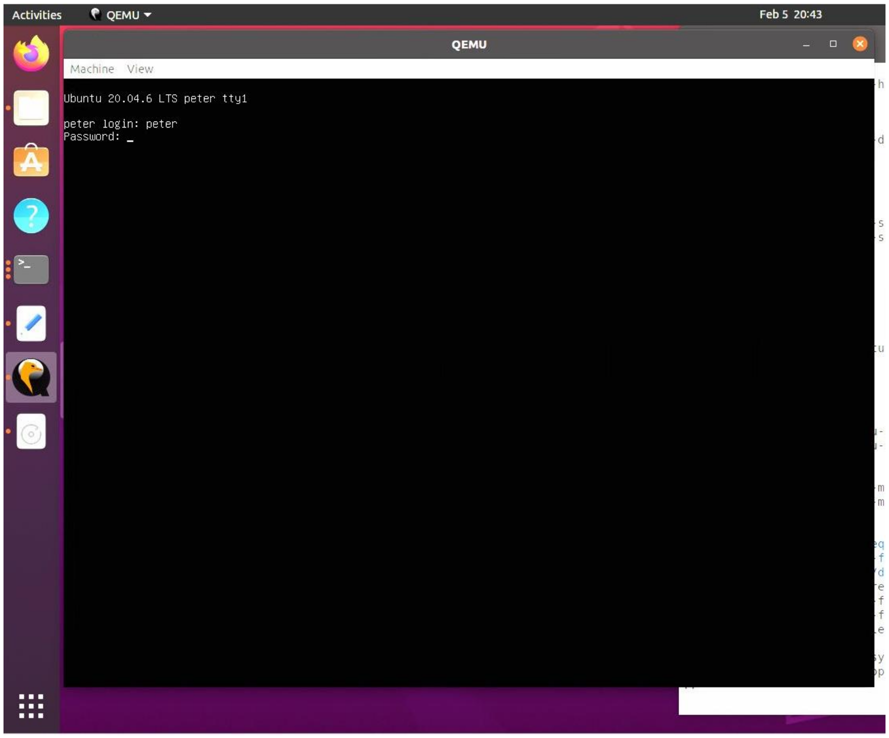
4. **Proof of experiment. Include screen snapshots of your Docker and QEMU running environments for each experiment: 10 points**
    - **QEMU**

    1. Start QEMU



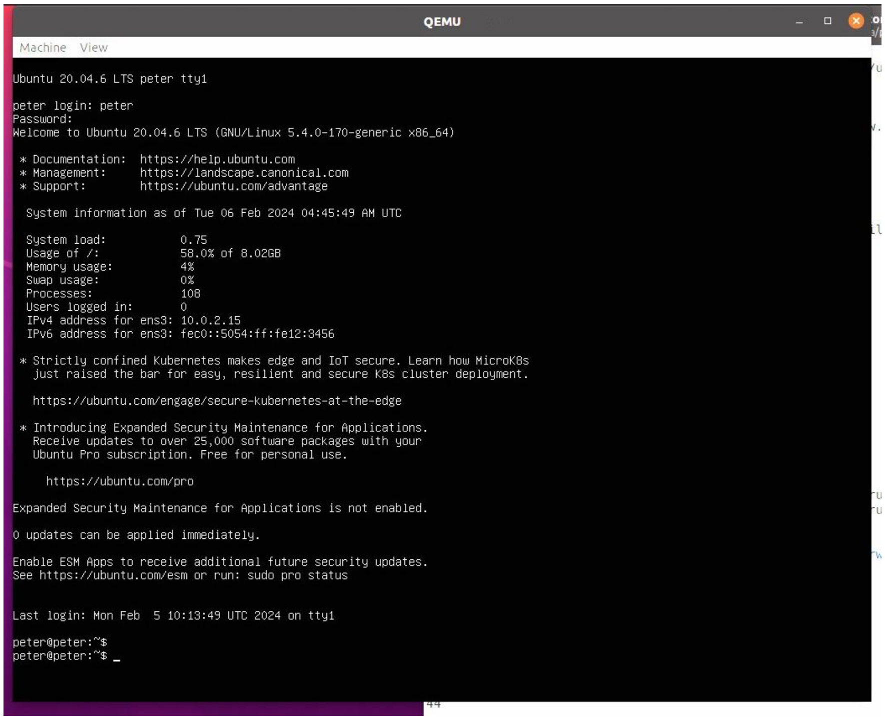
```
peter@ubuntu:/media/peter/32G$
peter@ubuntu:/media/peter/32G$ sudo qemu-system-x86_64 -hda ubuntu.img
```

    2. **Log in**



```
Ubuntu 20.04.6 LTS peter tty1

peter login: peter
Password: _
```

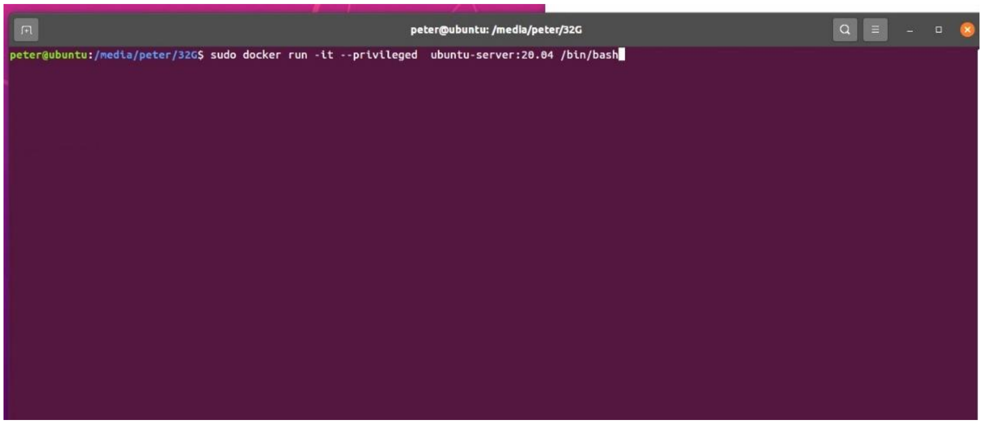
3. System info



```
Ubuntu 20.04.6 LTS peter tty1

peter login: peter
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-170-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue 06 Feb 2024 04:45:49 AM UTC

  System load:           0.75
  Usage of /:            58.0% of 8.02GB
  Memory usage:          4%
  Swap usage:            0%
  Processes:             108
  Users logged in:       0
  IPv4 address for ens3: 10.0.2.15
  IPv6 address for ens3: fec0::5054:ff:fe12:3456

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Mon Feb  5 10:13:49 UTC 2024 on tty1

peter@peter:~$
peter@peter:~$ _
```

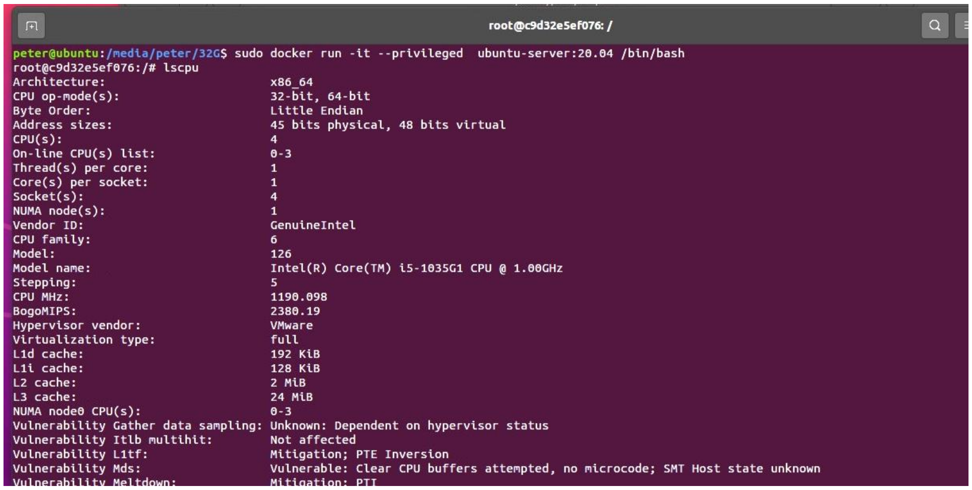
- **Container**

1. **Start container**



```
peter@ubuntu:/media/peter/32G$ sudo docker run -it --privileged  ubuntu-server:20.04 /bin/bash
```

2. **System info**



```
peter@ubuntu:/media/peter/32G$ sudo docker run -it --privileged  ubuntu-server:20.04 /bin/bash
root@c9d32e5ef076:/# lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   45 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       4
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           126
Model name:                      Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz
Stepping:                        5
CPU MHz:                         1190.098
BogoMIPS:                        2380.19
Hypervisor vendor:               VMware
Virtualization type:             full
L1d cache:                       192 KiB
L1i cache:                       128 KiB
L2 cache:                        2 MiB
L3 cache:                        24 MiB
NUMA node0 CPU(s):               0-3
Vulnerability Gather data sampling: Unknown: Dependent on hypervisor status
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
Vulnerability Meltdown:          Mitigation; PTI
```

5. **Present your measurements in given different scenarios for each virtualization technology: 20 points**
    - **For QEMU (qcow2)**

      There are 24 test cases, and perform each test case 5 times. The experiment results are as follows.

      8 test cases for CPU

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 2 -m 2G -hda ubuntu.img	2	2G	10000
	2	2G	20000

Command	Cpus	Memory	Prime

sudo qemu-system-x86_64 / -smp 2 -m 4G -hda ubuntu.img	2	4G	10000
	2	4G	20000

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 4 -m 2G -hda ubuntu.img	4	2G	10000
	4	2G	20000

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 4 -m 4G -hda ubuntu.img	4	4G	10000
	4	4G	20000

8 test cases for MEMORY

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 2 -m 2G -hda ubuntu.img	2	2G	1K
	2	2G	2K

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 2 -m 4G -hda ubuntu.img	2	4G	1K
	2	4G	2K

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 4 -m 2G -hda ubuntu.img	4	2G	1K
	4	2G	2K

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 4 -m 4G -hda ubuntu.img	4	4G	1K
	4	4G	2K

8 test cases for FILEIO

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 2 -m 2G -hda ubuntu.img	2	2G	seqrewr
	2	2G	rndwr

Command	Cpus	Memory	mode
	2	4G	seqrewr

sudo qemu-system-x86_64 / -smp 2 -m 4G -hda ubuntu.img	2	4G	rndwr

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 4 -m 2G -hda ubuntu.img	4	2G	seqrewr
	4	2G	rndwr

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 4 -m 4G -hda ubuntu.img	4	4G	seqrewr
	4	4G	rndwr

- **For QEMU (RAW)**

  There are 24 test cases, and perform each test case 5 times. The experiment results are as follows.

  8 test cases for CPU

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 2 -m 2G / -drive format=raw,file=ubuntu_raw.raw	2	2G	10000
	2	2G	20000

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 2 -m 4G / -drive format=raw,file=ubuntu_raw.raw	2	4G	10000
	2	4G	20000

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 4 -m 2G / -drive format=raw,file=ubuntu_raw.raw	4	2G	10000
	4	2G	20000

Command	Cpus	Memory	Prime
sudo qemu-system-x86_64 / -smp 4 -m 4G / -drive format=raw,file=ubuntu_raw.raw	4	4G	10000
	4	4G	20000

8 test cases for MEMORY

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 2 -m 2G /	2	2G	1K
	2	2G	2K

-drive format=raw,file=ubuntu_raw.raw			

•

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 2 -m 4G / -drive format=raw,file=ubuntu_raw.raw	2	4G	1K
	2	4G	2K

•

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 4 -m 2G / -drive format=raw,file=ubuntu_raw.raw	4	2G	1K
	4	2G	2K

•

Command	Cpus	Memory	Block Size
sudo qemu-system-x86_64 / -smp 4 -m 4G / -drive format=raw,file=ubuntu_raw.raw	4	4G	1K
	4	4G	2K

8 test cases for FILEIO

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 2 -m 2G / -drive format=raw,file=ubuntu_raw.raw	2	2G	seqrewr
	2	2G	rndwr

•

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 2 -m 4G / -drive format=raw,file=ubuntu_raw.raw	2	4G	seqrewr
	2	4G	rndwr

•

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 4 -m 2G / -drive format=raw,file=ubuntu_raw.raw	4	2G	seqrewr
	4	2G	rndwr

•

Command	Cpus	Memory	mode
sudo qemu-system-x86_64 / -smp 4 -m 4G / -drive format=raw,file=ubuntu_raw.raw	4	4G	seqrewr
	4	4G	rndwr

- **Container**

There are 24 test cases, and perform each test case 5 times. The experiment results are as follows.

8 test cases for CPU

Command	Cpus	Memory	Prime
sudo docker run -it --privileged / --cpus=2 --memory=2G / ubuntu-server:20.04 /bin/bash	2	2G	10000
	2	2G	20000

Command	Cpus	Memory	Prime
sudo docker run -it --privileged / --cpus=2 --memory=4G / ubuntu-server:20.04 /bin/bash	2	4G	10000
	2	4G	20000

Command	Cpus	Memory	Prime
sudo docker run -it --privileged / --cpus=4 --memory=2G / ubuntu-server:20.04 /bin/bash	4	2G	10000
	4	2G	20000

Command	Cpus	Memory	Prime
sudo docker run -it --privileged / --cpus=4 --memory=4G / ubuntu-server:20.04 /bin/bash	4	4G	10000
	4	4G	20000

8 test cases for MEMORY

Command	Cpus	Memory	Block Size
sudo docker run -it --privileged / --cpus=2 --memory=2G / ubuntu-server:20.04 /bin/bash	2	2G	1K
	2	2G	2K

Command	Cpus	Memory	Block Size
sudo docker run -it --privileged / --cpus=2 --memory=4G / ubuntu-server:20.04 /bin/bash	2	4G	1K
	2	4G	2K

Command	Cpus	Memory	Block Size
sudo docker run -it --privileged / --cpus=4 --memory=2G / ubuntu-server:20.04 /bin/bash	4	2G	1K
	4	2G	2K

Command	Cpus	Memory	Block Size
sudo docker run -it --privileged / --cpus=4 --memory=4G / ubuntu-server:20.04 /bin/bash	4	4G	1K
	4	4G	2K

8 test cases for FILEIO

Command	Cpus	Memory	mode
sudo docker run -it --privileged / --cpus=2 --memory=2G / ubuntu-server:20.04 /bin/bash	2	2G	`seqrewr`
	2	2G	`rndwr`

Command	Cpus	Memory	mode
sudo docker run -it --privileged / --cpus=2 --memory=4G / ubuntu-server:20.04 /bin/bash	2	4G	`seqrewr`
	2	4G	`rndwr`

Command	Cpus	Memory	mode
sudo docker run -it --privileged / --cpus=4 --memory=2G / ubuntu-server:20.04 /bin/bash	4	2G	`seqrewr`
	4	2G	`rndwr`

Command	Cpus	Memory	mode
sudo docker run -it --privileged / --cpus=4 --memory=4G / ubuntu-server:20.04 /bin/bash	4	4G	`seqrewr`
	4	4G	`rndwr`

6. **Shell scripts for running the experiment: 10 points**
   The shell script is used for performing the experiment automatically.

```
#!/bin/bash

#/** */
touch "results.txt"

#/** Init */
K=(1 2 3 4 5)
CPU=("10000" "20000")
MEMORY=("1k" "2k")
FILEIO=("seqrewr" "rndwr")
```

```
#/** Execute commands */
echo "################ CPU #################" | tee -a results.txt
for k in "${K[@]}"; do
    #/** compute five times */
    echo "******* $k ******" | tee -a results.txt
    #/** CPU */
    for cpu in "${CPU[@]}"; do
        echo "@@@ $cpu " | tee -a results.txt
        sysbench --test=cpu --cpu-max-prime=$cpu /
        --time=30 run | tee -a results.txt
        echo "" | tee -a results.txt
    done

done


echo "################ MEMO #################"| tee -a results.txt
for k in "${K[@]}"; do
    #/** MEMO */
    echo "********** $k *************" | tee -a results.txt
    for memo in "${MEMORY[@]}"; do
        echo "@@@ $memo " | tee -a results.txt
        sysbench --test=memory --memory-block-size=$memo /
        --time=30 run | tee -a results.txt
        echo "" | tee -a results.txt
    done

done


echo "################# FILEIO ##############" | tee -a results.txt
for k in "${K[@]}"; do
    #/** FILEIO */
    echo "*********** $k *************" | tee -a results.txt
    for fileio in "${FILEIO[@]}"; do
        echo "@@@ $fileio " | tee -a results.txt
        /** prepare data */
        sysbench --test=fileio prepare
        sysbench --test=fileio --file-test-mode=$fileio /
        --time=30 run | tee -a results.txt /
        sysbench --test=fileio cleanup
        echo "" | tee -a results.txt

        #/** clean cache */
        echo 3 > /proc/sys/vm/drop_caches | tee -a results.txt
    done
done

echo "################ UPLOAD ###############" | tee -a results.txt
```

```
sshpass -p 'ZTj2024!!$$' scp ./results.txt root@50.116.10.36:/home/
echo "success!" | tee -a results.txt

echo "################ DELETE ################"
rm results.txt
```

Table1 shell script

7. **Presentation and analysis of the performance data: 20 points**
    1. **Qemu(qcow2)**

Cpus = 2, Memory = 2G

CPUs	MEMORY	Prime	min	avg	max
2	2G	10000	2.272	3.71	91.78
2	2G	20000	6.36	9.23	94.9

CPUs	MEMORY	Block size	min	avg	max
2	2G	1k	0	0	23.248
2	2G	2k	0	0	25.504

CPUs	MEMORY	FiloIO	min	avg	max
2	2G	Seq	0.05	0.422	76.522
2	2G	Rnd	0.026	0.978	99.396

Cpus = 2, Memory = 4G

CPUs	MEMORY	Prime	min	avg	max
2	4G	10000	2.84	3.99	29.19
2	4G	20000	5.81	7.89	34.33

CPUs	MEMORY	Block size	min	avg	max
2	4G	1k	0	0	18.026
2	4G	2k	0	0	16.054

CPUs	MEMORY	FiloIO	min	avg	max
2	4G	seq	0.03	0.4	95.374
2	4G	rnd	0.024	0.928	89.854

Cpus =4, Memory = 2G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	2G	10000	2.17	2.816	39.946
4	2G	20000	5.766	7.136	26.282

CPUs	MEMORY	Block size	min	avg	max
4	2G	1k	0	0	14.93
4	2G	2k	0	0	12.69

CPUs	MEMORY	FiloIO	min	avg	max
4	2G	seq	0.042	0.442	56.828
4	2G	rnd	0.024	0.824	49.362

Cpus = 4, Memory = 4G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	4G	10000	2.142	2.86	18.466
4	4G	20000	5.676	6.936	23.794

CPUs	MEMORY	Block size	min	avg	max
4	4G	1k	0	0	15.214
4	4G	2k	0	0	17.706

CPUs	MEMORY	FiloIO	min	avg	max
4	4G	seq	0.02	0.36	45.284
4	4G	rnd	0.024	0.874	42.664

2 Qemu(RAW)

Cpus = 2, Memory = 2G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
2	2G	10000	2.158	3.284	48.716
2	2G	20000	5.754	7.626	46.552

CPUs	MEMORY	Block size	min	avg	max
2	2G	1k	0	0	5.73
2	2G	2k	0	0	7.964

CPUs	MEMORY	FiloIO	min	avg	max
2	2G	seq	0.042	0.312	19.566
2	2G	rnd	0.022	0.706	33.312

Cpus = 2, Memory = 4G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
2	4G	10000	2.154	3.362	31.05
2	4G	20000	5.888	8.69	44.33

CPUs	MEMORY	Block size	min	avg	max
2	4G	1k	0	0	17.762
2	4G	2k	0	0	18.008

CPUs	MEMORY	FiloIO	min	avg	max
2	4G	seq	0.024	0.394	75.674
2	4G	rnd	0.024	1.006	85.932

Cpus = 4, Memory = 2G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	2G	10000	2.288	3.79	53.662
4	2G	20000	5.982	8.732	56.116

CPUs	MEMORY	Block size	min	avg	max
4	2G	1k	0	0	57.716
4	2G	2k	0	0	59.154

CPUs	MEMORY	FiloIO	min	avg	max
4	2G	seq	0.062	1.082	124.124
4	2G	rnd	0.032	1.93	168.43

Cpus = 4, Memory = 4G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	4G	10000	2.168	3.238	55.16
4	4G	20000	5.932	7.932	43.622

CPUs	MEMORY	Block size	min	avg	max

4	4G	1k	0	0	27.928
4	4G	2k	0	0	28.022

CPUs	MEMORY	FiloIO	min	avg	max
4	4G	seq	0.028	0.45	106.908
4	4G	rnd	0.026	0.966	98.672

3 Docker

Cpus = 2, Memory = 2G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
2	2G	10000	0.32	0.452	34.586
2	2G	20000	0.84	1.118	27.442

CPUs	MEMORY	Block size	min	avg	max
2	2G	1k	0	0	16.112
2	2G	2k	0	0	22.74

CPUs	MEMORY	FiloIO	min	avg	max
2	2G	seq	0	0.076	11068.354
2	2G	rnd	0	0.124	4616.096

Cpus = 2, Memory = 4G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
2	4G	10000	0.406	0.49	34.546
2	4G	20000	1.078	1.25	36.058

CPUs	MEMORY	Block size	min	avg	max
2	4G	1k	0	0	27.21
2	4G	2k	0	0	10.142

CPUs	MEMORY	FiloIO	min	avg	max
2	4G	seq	0	0.072	5893.68
2	4G	rnd	0	0.13	4825.658

Cpus = 4, Memory = 2G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	2G	10000	0.446	0.61	36.644

4	2G	20000	1.156	1.546	41.526

CPUs	MEMORY	Block size	min	avg	max
4	2G	1k	0	0	17.632
4	2G	2k	0	0	10.078

CPUs	MEMORY	FiloIO	min	avg	max
2	2G	seq	0	0.052	4129
2	2G	rnd	0	0.13	857.616

Cpus = 4, Memory = 4G

CPUs	**MEMORY**	**Prime**	**min**	**avg**	**max**
4	4G	10000	0.32	0.424	2.786
4	4G	20000	0.848	1.126	179.568

CPUs	MEMORY	Block size	min	avg	max
4	4G	1k	0	0	1.992
4	4G	2k	0	0	1.48

CPUs	MEMORY	FiloIO	min	avg	max
4	4G	seq	0	0.022	138.08
4	4G	rnd	0	0.164	5088.78

As depicted above, the comparison among the three ways is as follow.

CPUs	**MEMORY**	**Performance (average latency)**
2	2G	Docker < Qemu(raw) < Qemu(qcow2)
2	4G	Docker < Qemu(raw) < Qemu(qcow2)
4	2G	Docker < Qemu(raw) <= Qemu(qcow2)
4	4G	Docker < Qemu(qcow2) <= Qemu(raw)

Table 2 comparison results

It's important to consider, Docker is much faster in terms of latency than both QEMU(qcow2) and QEMU(raw). For both QEMU(qcow2) and QEMU(raw), in the context of limited resources like CPU and memory, QEMU(raw) potentially outperform in terms of latency. However, if the resources become more powerful, QEMU(qcow2) will have lower latency than QEMU(raw).