Project Report
COEN 241 Cloud Computing HW1

Environment Setup:

Installing QEMU

1.

```
brew install qemu
```

```
ls /opt/homebrew/bin/qemu-*
/opt/homebrew/bin/qemu-edid              /opt/homebrew/bin/qemu-system-cris
/opt/homebrew/bin/qemu-system-mips64el   /opt/homebrew/bin/qemu-system-s390x
/opt/homebrew/bin/qemu-img               /opt/homebrew/bin/qemu-system-hppa
/opt/homebrew/bin/qemu-system-mipsel     /opt/homebrew/bin/qemu-system-sh4
/opt/homebrew/bin/qemu-io
```

```
qemu-system-aarch64 --version
QEMU emulator version 7.2.0
right (c) 2003-2022 Fabrice Bellard and the QEMU Project developers
```

2. – Download Ubuntu (or any Linux Distribution)

In order to be able to install an operating system running on a virtual machine in QEMU, you will need an ISO image containing the OS.

```
https://ubuntu.com/download/desktop
```

3. – Create empty image

Create a QEMU empty image where to install your Ubuntu OS later. To do so, run the following command:

```
qemu-img create -f raw  ~/qemu/ubuntu-latest.raw 40G
```

4. – Download pre-built EDK2 UEFI image for QEMU

```
mv ~/Downloads/QEMU_EFI-*.tar.gz ~/qemu
```

5. – Decompress the tar.gz file

```
tar xzvf QEMU_EFI-*.tar.gz
```

6. – Install Ubuntu Server

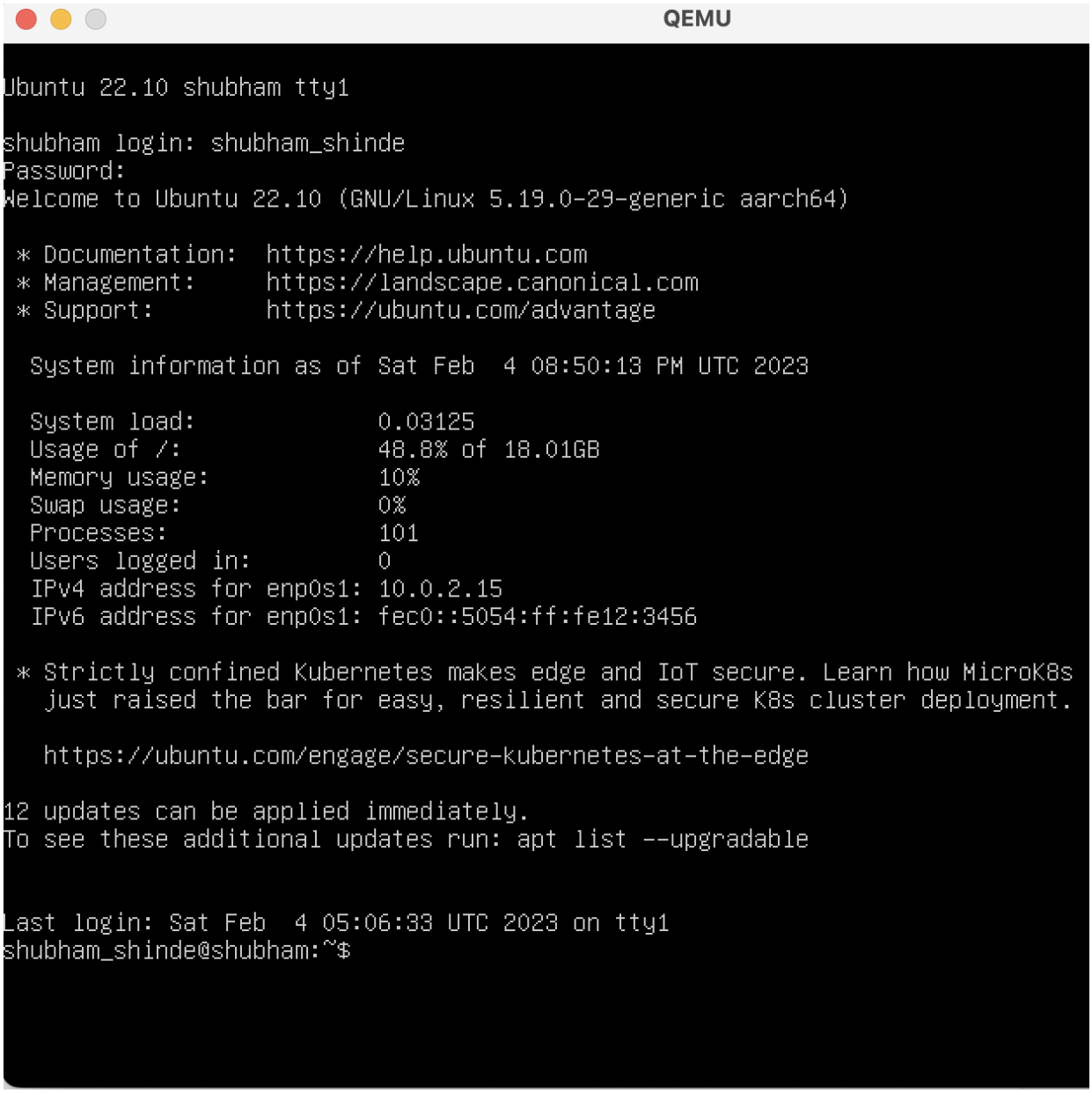
```
qemu-system-aarch64 \
  -monitor stdio \
  -M virt,highmem=off \
  -accel hvf \
  -cpu host \
  -smp 4 \
  -m 3000 \
  -bios QEMU_EFI.fd \
  -device virtio-gpu-pci \
  -display default,show-cursor=on \
  -device qemu-xhci \
  -device usb-kbd \
  -device usb-tablet \
  -device intel-hda \
  -device hda-duplex \
  -drive
file=ubuntu-latest.raw,format=raw,if=virtio,cache=writethrough \
```

7. – Start Ubuntu using QEMU

```
qemu-system-aarch64 \
  -monitor stdio \
  -M virt,highmem=off \
  -accel hvf \
  -cpu host \
  -smp 4 \
  -m 3000 \
  -bios QEMU_EFI.fd \
  -device virtio-gpu-pci \
  -display default,show-cursor=on \
  -device qemu-xhci \
  -device usb-kbd \
  -device usb-tablet \
  -device intel-hda \
  -device hda-duplex \
  -drive file=ubuntu-latest.raw,format=raw,if=virtio,cache=writethrough
\
```

Once you open a new terminal (or reload your context using "source ~/.zshrc"), you should be able to run the following:

```
$ ubuntu
QEMU 7.2.0 monitor - type 'help' for more information
```


QEMU
Ubuntu 22.10 shubham tty1
shubham login: shubham_shinde
Password:
Welcome to Ubuntu 22.10 (GNU/Linux 5.19.0-29-generic aarch64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
System information as of Sat Feb 4 08:50:13 PM UTC 2023
System load: 0.03125
Usage of /: 48.8% of 18.01GB
Memory usage: 10%
Swap usage: 0%
Processes: 101
Users logged in: 0
IPv4 address for enp0s1: 10.0.2.15
IPv6 address for enp0s1: 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
12 updates can be applied immediately.
To see these additional updates run: apt list --upgradable
Last login: Sat Feb 4 05:06:33 UTC 2023 on tty1
shubham_shinde@shubham:~$

```
root@9276df87b1a2:/# lscpu
Architecture:          aarch64
  CPU op-mode(s):      64-bit
  Byte Order:          Little Endian
CPU(s):                4
  On-line CPU(s) list: 0-3
Vendor ID:             0x00
  Model:               0
  Thread(s) per core:  1
  Core(s) per cluster: 4
  Socket(s):           -
  Cluster(s):          1
  Stepping:            0x0
  BogoMIPS:            48.00
  Flags:               fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid asimdrdm jscvt fcma lrc
                       pc dcpop sha3 asimddp sha512 asimdfhm dit uscat ilrcpc flagm ssbs sb paca pacg dcpodp flagm2
                       frint
Vulnerabilities:
  Itlb multihit:       Not affected
  L1tf:                Not affected
  Mds:                 Not affected
  Meltdown:            Not affected
  Mmio stale data:     Not affected
  Spec store bypass:   Vulnerable
  Spectre v1:          Mitigation; __user pointer sanitization
  Spectre v2:          Not affected
  Srbds:               Not affected
  Tsx async abort:     Not affected
root@9276df87b1a2:/#
```

```
root@9276df87b1a2:/# free -m
               total        used        free      shared  buff/cache   available
Mem:            7851         316        6723         322         811        7049
Swap:           1023           0        1023
root@9276df87b1a2:/#
```

Installing OS Virtualization (Docker) Setup

https://docs.docker.com/desktop/install/mac-install/

Setting up Docker

```
docker start ubuntu
```

To create a Docker container with specified CPU and memory, you can use the following command:

```
docker run --cpus 2 --memory 2g --name mycontainer -d ubuntu
```

This will create a Docker container named "mycontainer" with 2 CPUs and 2 GB memory, using the "ubuntu" image.

```
shubhamshinde@Shubhams-MacBook-Pro ~ % docker run hello world
Unable to find image 'hello:latest' locally
docker: Error response from daemon: pull access denied for hello, repository doe
s not exist or may require 'docker login': denied: requested access to the resou
rce is denied.
See 'docker run --help'.
shubhamshinde@Shubhams-MacBook-Pro ~ % docker run hello-world


Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

shubhamshinde@Shubhams-MacBook-Pro ~ %
```

```
shubham_shinde@shubham:~$ free -m
               total        used        free      shared  buff/cache   available
Mem:            2895         219        1606           3        1069        2588
Swap:           2894           0        2894
shubham_shinde@shubham:~$
```

Installing sysbench

```
$ sudo apt update

$ sudo apt install sysbench
```

```
root@9276df87b1a2:/# history
    1  ls
    2  apt install sysbench
    3  sysbench cpu --cpu-max-prime=2000  --num-threads={some_value} --time=20 run
    4  sysbench cpu --cpu-max-prime=2000  --num-threads=1 --time=20 run
    5  sysbench cpu --cpu-max-prime=2000  --num-threads=1 --time=40 run
    6  sysbench cpu --cpu-max-prime=15000  --num-threads=1 --time=35 run
    7  nano config1.sh
    8  ls
    9  chmod +x congif.sh
   10  chmod +x config1.sh
   11  sh config1.sh
   12  run config1.sh
   13  ./config1.sh
   14  cat config1.sh
   15  docker -v
   16  ls
   17  nano config1.sh
   18  chmod +x config1.sh
   19  ./config1.sh
   20  nano config2.sh
   21  ls
   22  cat config1.sh
   23  nano config2.sh
   24  chmod +x config2.sh
   25  ./config2.sh
   26  ls
   27  cat config2
   28  cat config2.sh
   29  nano config3.sh
   30  chmod +x config3.sh
   31  ./config3.sh
   32  ls
   33  cat config1.sh
   34  lscpu
   35  free -m
   36  sysbench --test=fileio --file-total-size=1G --test-mode=rndrw prepare
   37  sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw prepare
   38  sysbench --test=fileio --file-total-size=1G --time=30 --file-test-mode=rndrw run
   39  sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw cleanup
   40  history
root@9276df87b1a2:/#
```

Configurations:

1. 2 core with 2 gb
2. 4 core with 4 gb
3. 8 core with 3gb

Tests:

- ❖ **cpu and fileio test**

For each test case, “right” parameters are figured out.
1. The parameter for “–cpu-max-prime” to ensure the test case won’t end in a short time period (too small) or will never end (too large).
2. The –time parameter
3. The file size (e.g., –file-total-size) for Sysbench fileio test mode.
4. Test case lasting for at least 30 seconds

- ➢ The sysbench measurement are repeated at least 5 times for each test case
- ➢ Report of the average, min, max and std. values of my results
- ➢ The Sysbench reports some user-level performance data, e.g., total time).

Configurations Used :

A. Sysbench
   a. CPU = 5000, Thread = 1, Time = 35
   b. CPU = 10000, Thread = 1, Time = 30
   c. CPU = 150000, Thread = 1, Time = 50

B. Fileio
   a. Threads = 1, File size = 1gb, Time = 30
   b. Threads = 16, File size = 2gb
   c. Threads = 20, File size = 1gb

Bash scripts have been created to automate the experiment.

A. Sysbench

```
for i in {1..5}
do
        echo "Config 1, iteration $i"
        echo "CPU = 5000", Thread = 1, Time = 35
        sysbench cpu --cpu-max-prime=5000  --num-threads=1 --time=35 run
done

for j in {1..5}
do
    echo "Config 2, iteration $j"
    echo "CPU = 10000", Thread = 1, Time = 30
    sysbench cpu --cpu-max-prime=10000  --num-threads=1 --time=30 run
done

for k in {1..5}
do
    echo "Config 3, iteration $k"
    echo "CPU = 150000", Thread = 1, Time = 50
    sysbench cpu --cpu-max-prime=150000  --num-threads=1 --time=50 run

done
```

B. Fileio

```
for i in {1..5}
do
        sysbench --test=fileio --file-total-size=1G --test-mode=rndrw prepare
        sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw prepare
        echo "Config 1, iteration $i"
        echo "Threads = 1, size = 1gb, time = 30"
        sysbench --test=fileio --file-total-size=1G --time=30 --file-test-mode=rndrw run
        sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw cleanup
done

for j in {1..5}
do
    sysbench --test=fileio --file-total-size=2G --test-mode=rndrw prepare
    sysbench --test=fileio --file-total-size=2G --file-test-mode=rndrw prepare
    echo "Config 2, iteration $j"
    echo "Threads = 16, size = 2gb"
    sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=rndrw run
    sysbench --test=fileio --file-total-size=2G --file-test-mode=rndrw cleanup
done


for k in {1..5}
do
    sysbench --test=fileio --file-total-size=1G --test-mode=rndrw prepare
    sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw prepare
    echo "Config 3, iteration $k"
    echo "Threads = 20, size = 1gb"
    sysbench --num-threads=20 --test=fileio --file-total-size=1G --file-test-mode=rndrwrun
    sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw cleanup
done
```

Comparing Docker vs Qemu

Configuration 1 : 2 core with 2 gb

C. Sysbench

a. CPU = 5000, Thread = 1, Time = 35

Virtualization	Min	Max	Average
Docker	20992.12	21091	21050.842
Qemu	21095.16	21197.56	21153.964

b. CPU = 10000, Thread = 1, Time = 30

Virtualization	Min	Max	Average
Docker	8638.95	8662.83	8652.556
Qemu	8655.47	8704.97	8682.284

c. CPU = 150000, Thread = 1, Time = 50

Virtualization	Min	Max	Average
Docker	266.27	268.06	267.004
Qemu	449.67	456.73	454.958

Configuration 2 : 4 core with 4 gb

D. Sysbench

a. CPU = 5000, Thread = 1, Time = 35

Virtualization	Min	Max	Average
Docker	21043.59	21100.53	21082.66
Qemu	20945.97	21106.36	21055.45

b. CPU = 10000, Thread = 1, Time = 30

Virtualization	Min	Max	Average
Docker	8424.19	8679.15	8624.09
Qemu	8636.81	8666.33	8645.538

c. CPU = 150000, Thread = 1, Time = 50

Virtualization	Min	Max	Average
Docker	264.35	266.49	265.238
Qemu	266.27	268.06	267.378

Configuration 3 : 8 core with 3gb

E. Sysbench

a. CPU = 5000, Thread = 1, Time = 35

Virtualization	Min	Max	Average
Docker	20602.12	20940.84	20825.40
Qemu	21105.28	21156.61	21133.622

b. CPU = 10000, Thread = 1, Time = 30

Virtualization	Min	Max	Average
Docker	8424.4	8660.82	8553.534
Qemu	8676.14	8697.13	8691.76

c. CPU = 150000, Thread = 1, Time = 50

Virtualization	Min	Max	Average
Docker	264.4	268.82	265.988
Qemu	267.56	268.46	268.062

Git Repository Information

Git clone link : [Link to the repo]

Conclusion

The experiment showed that Qemu often runs quicker than Docker, but their performance is generally close with just a slight difference in events per second.

Screenshots Test Cases Run for the following configurations :

Docker :

Configuration 1 : 2 core with 2 gb

F. Sysbench
   a. CPU = 5000, Thread = 1, Time = 35

```
root@9276df87b1a2:/# nano config1.sh
root@9276df87b1a2:/# chmod +x config1.sh
root@9276df87b1a2:/# ./config1.sh
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 5000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 21051.09

General statistics:
    total time:                          35.0002s
    total number of events:              736816

Latency (ms):
         min:                                    0.05
         avg:                                    0.05
         max:                                    4.42
         95th percentile:                        0.05
         sum:                                34920.35

Threads fairness:
    events (avg/stddev):           736816.0000/0.00
    execution time (avg/stddev):   34.9204/0.00
```

```
root@9276df87b1a2:/# ./config1.sh
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 5000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 20992.12

General statistics:
    total time:                          35.0002s
    total number of events:              734751

Latency (ms):
         min:                                    0.05
         avg:                                    0.05
         max:                                    5.88
         95th percentile:                        0.05
         sum:                                34899.48

Threads fairness:
    events (avg/stddev):           734751.0000/0.00
    execution time (avg/stddev):   34.8995/0.00

root@9276df87b1a2:/#
```