System vs OS Virtualization

Git Repository

https://github.com/evanFengg/CloudComputing.git

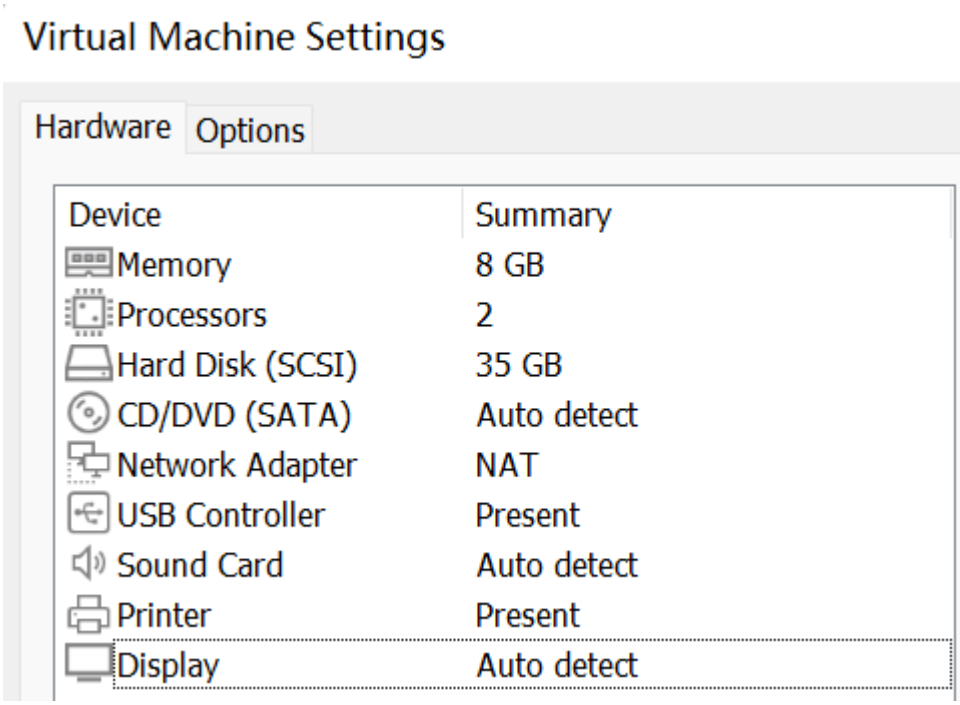
It is a public repository.

Configurations

I used VMware on my Windows 10 PC to create a Linux environment which is ubuntu 22.04

Linux Configurations(guest OS)

OS: Ubuntu 22.04 64-bit



QEMU VM Configurations

Guest OS: Ubuntu 20.04

Processor cores: 2

Memory: 512 MB, 1GB, 2GB

Storage: 12 GB

Docker Container Configurations

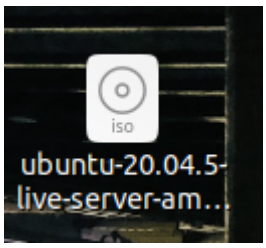
Image: My own image createc from Ubuntu:22.04(latest) + sysbench

Processor cores: 2

Memory: 512 MB, 1GB, 2GB

QEMU Setup

1. Download Ubuntu 20.04 server ISO image.



2. Open the terminal and install QEMU.

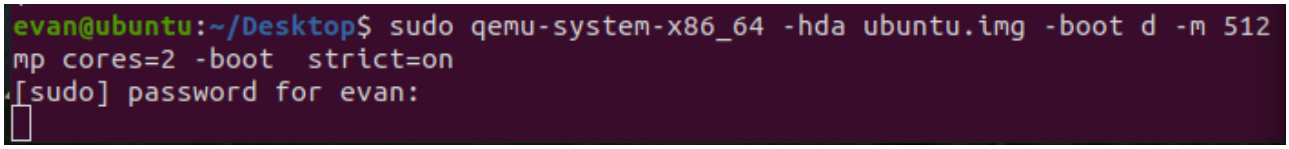
```
$ sudo apt-get install qemu

$ sudo apt-get install qemu-utils

$ sudo apt-get install qemu-system-x86
```

3. Create a QEMU image (12GB disk space, image format of QEMU copy-on-write v2).

```
$ sudo qemu-img create ubuntu.img 12G -f qcow2
```

4. Install QEMU VM.

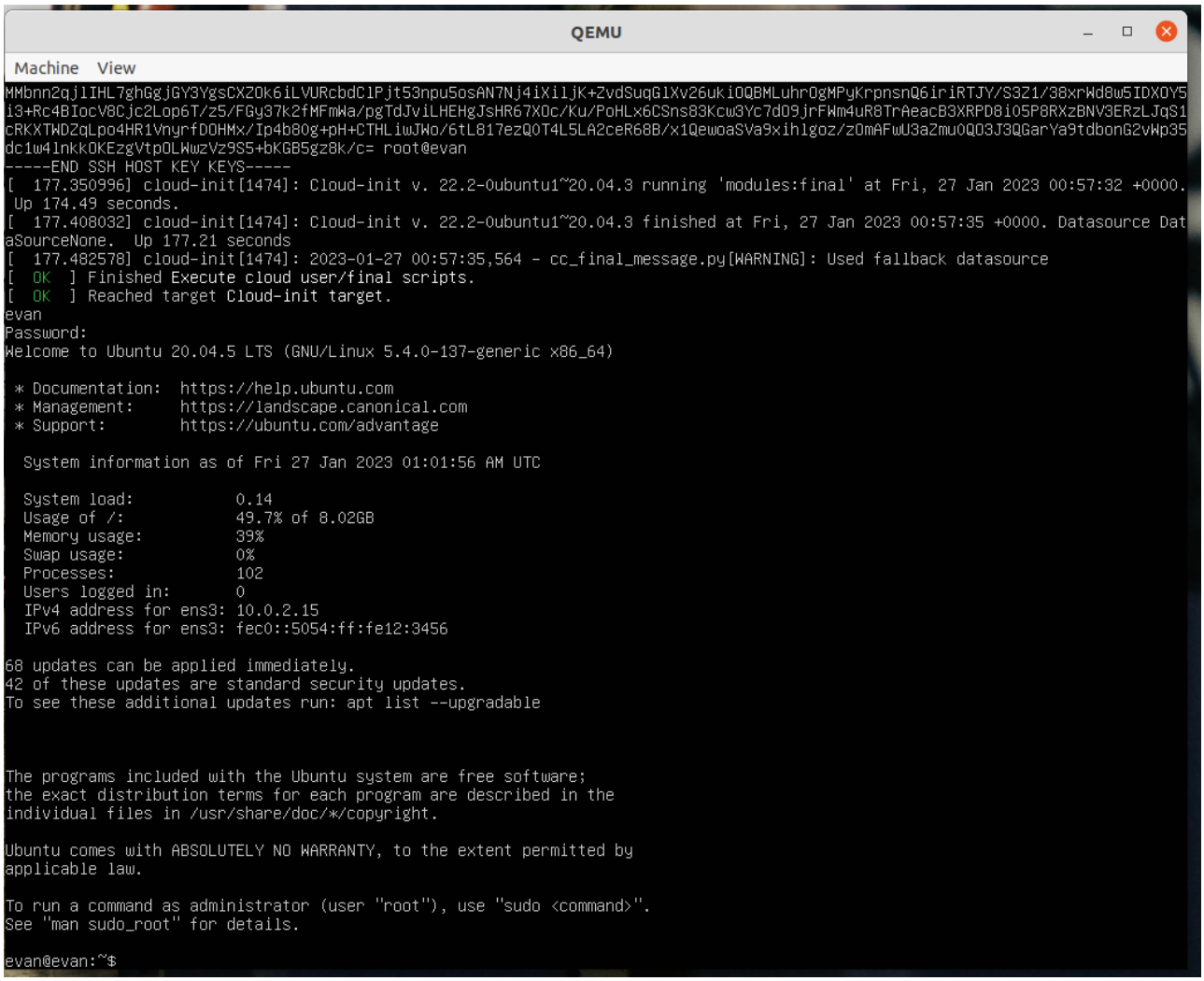
```
$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.5-live-server-amd64.iso -m 2048 -boot strict=on
```



5. Boot the VM created from the image ubuntu.img.

```
$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -m 512 -smp cores=2 -boot strict=on
```

6. Login to the QEMU Ubuntu with username and password. Then we can see the QEMU VM running.



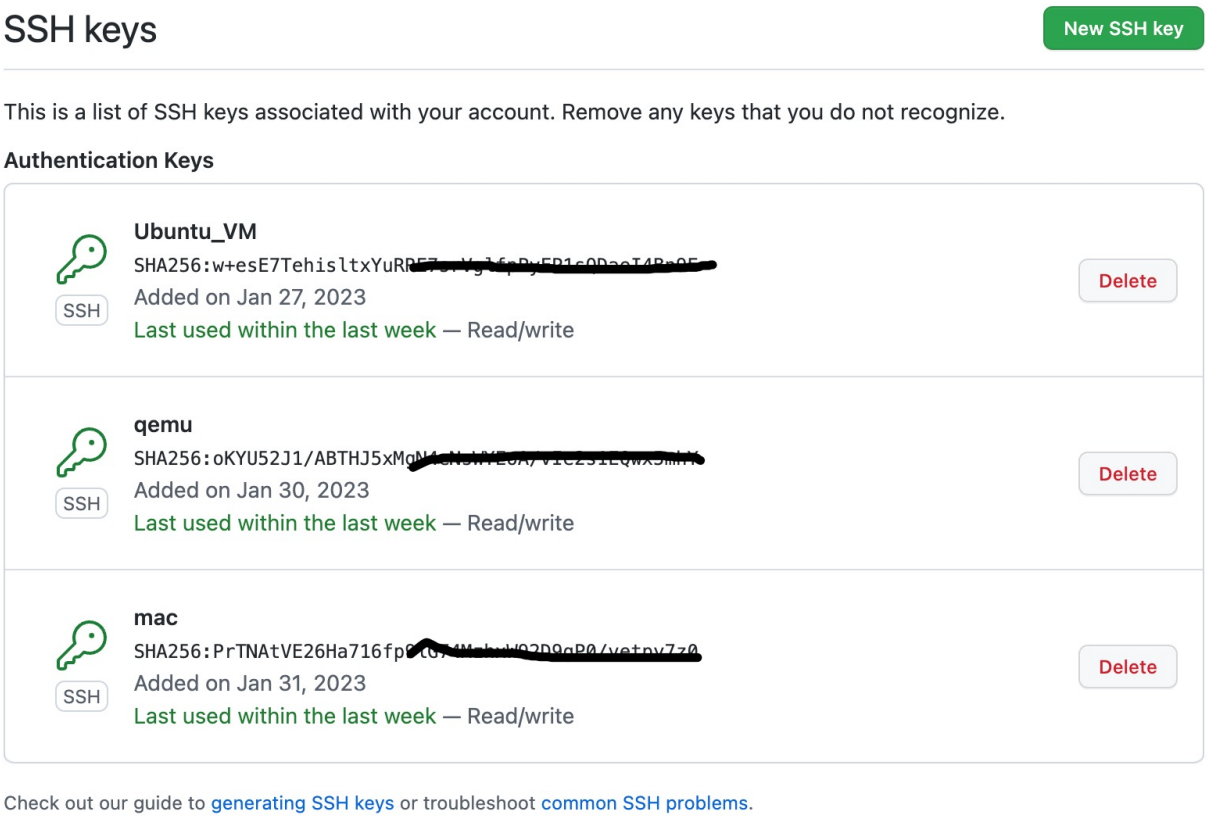
7. Install the sysbench in the QEMU.

   ```
   $ sudo apt update
   ```

   ```
   $ sudo apt install sysbench
   ```

   Sysbench version: 1.0.18

8. Set up the GitHub authorization with ssh and pull the repository to the local VM.



QEMU VM Performance Testing with Varying Configurations

the scenarios is that I will change QEMU VM memory from 512 to 2o48 MB ,and I will analyze their performance by getting the sysbench measurement.

Steps

1. Boot the VM, each time with different parameter.

```
$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -m 512 -smp cores=2 -boot strict=on

$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -m 1024 -smp cores=2 -boot strict=on

$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -m 2048 -smp cores=2 -boot strict=on
```

2. Show VM environment.

```
$ printenv
```

3. Use the 'top' command line tool in the host (Linux, Ubuntu desktop) terminal to get the CPU utilization, including user-level and kernel-level.

```
$ top
```

4. Run each shell script and save results.

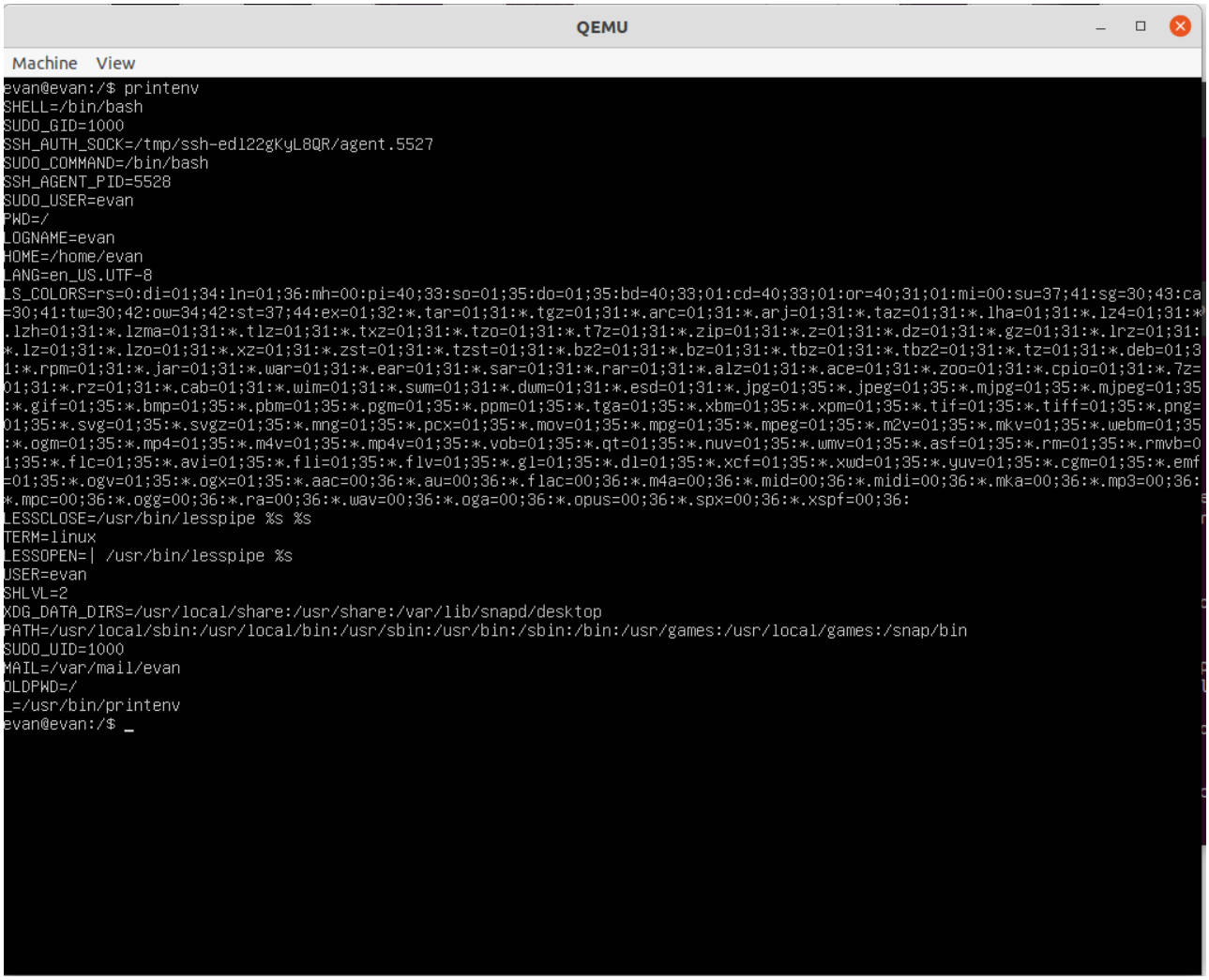
```
$ bash script_file_name.sh > output_file_name.txt
```

5. Push all output files to the repository.

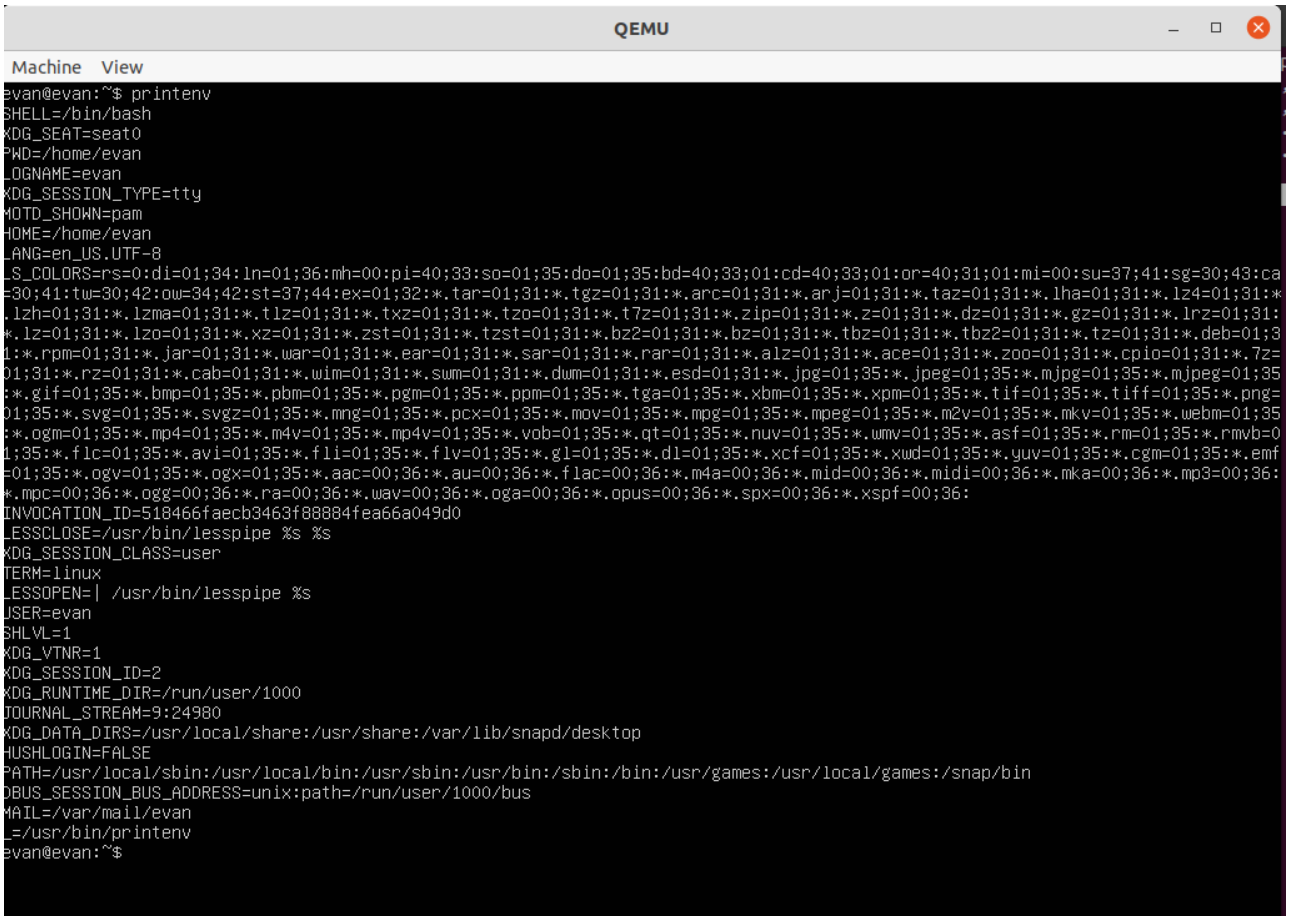
```
$ git add .

$ git commit -m "add files"

$ git push origin main
```

Running Environments

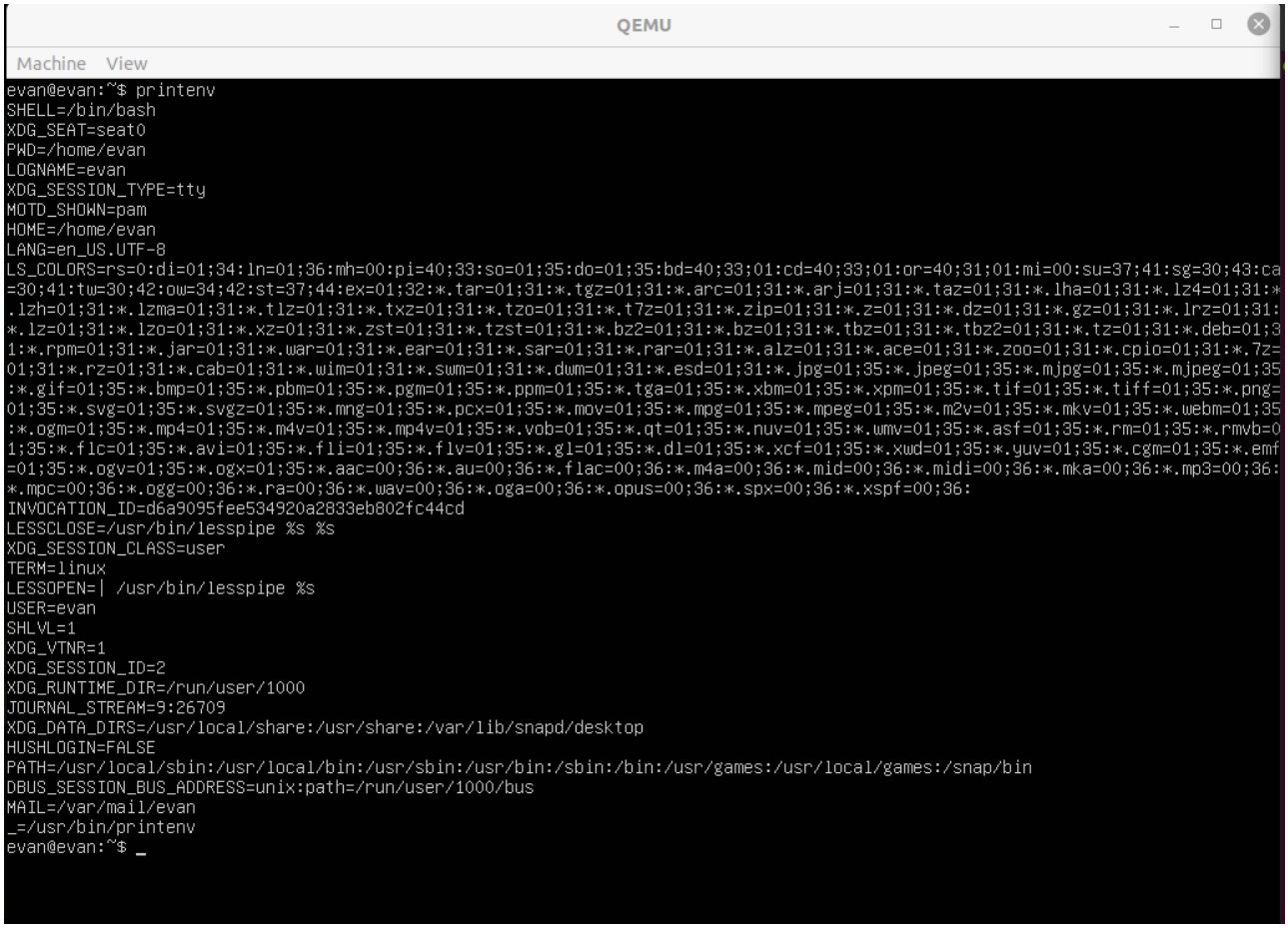
- Low-memory test



- Middle-memory test



- High-memory test



Shell Scripts

- VM Sysbench CPU Test Script

```
#!/bin/bash


mem=$(free -m | grep "Mem:" | awk '{print $2}')

if [ $mem -le 512 ];
then
    echo "Low-memory QEMU VM Sysbench CPU Tests: 2 core, 512 MB Memory"
    echo "============================================================="

elif [ $mem -le 1024 ];
then
    echo "Middle-memory QEMU VM Sysbench CPU Tests: 2 core, 1024 MB
Memory"
    echo
"============================================================="

elif [ $mem -le 2048 ];
then
    echo "High-memory QEMU VM Sysbench CPU Tests: 2 core, 2048 MB Memory"
    echo "============================================================="

else
    echo "Incorrect Memory to run sysbench, needs to match preset
configurations"
    exit 1
fi

# Run sysbench CPU test for 5 times on each VM configuration
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    echo "------"
    sysbench --test=cpu --cpu-max-prime=10000 --time=30 run
done
```

- VM Sysbench File IO Test Script

```
#!/bin/bash


mem=$(free -m | grep "Mem:" | awk '{print $2}')

if [ $mem -le 512 ];
then
    echo "Low-memory QEMU VM Sysbench File IO Tests: 2 core, 512 MB
Memory"
    echo
```

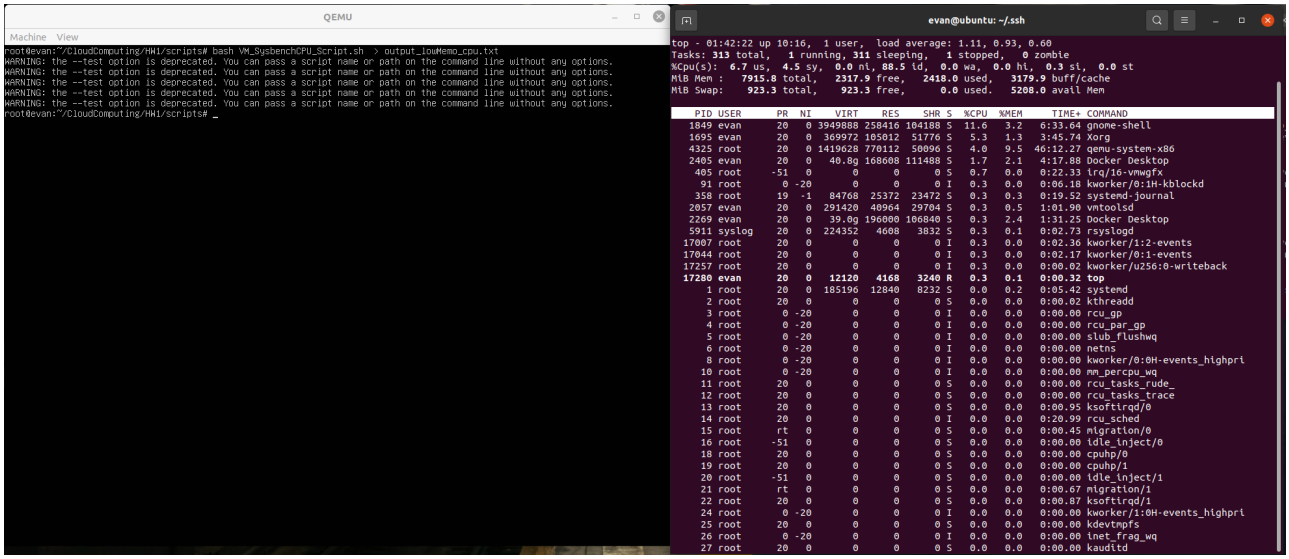
```
"============================================================="

elif [ $mem -le 1024 ];
then
    echo "Middle-memory QEMU VM Sysbench File IO Tests: 2 core, 1024 MB
Memory"
    echo
"============================================================="

elif [ $mem -le 2048 ];
then
    echo "High-memory QEMU VM Sysbench File IO Tests: 2 core, 2048 MB
Memory"
    echo
"============================================================="

else
    echo "Incorrect Memory to run sysbench, needs to match preset
configurations"
    exit 1
fi

# Run sysbench file io tests in sequential write mode and sequential read
mode
# Each test is run 5 times on each VM configuration
echo "File IO Test in Sequential Write Mode"
echo "-----------------------------------"
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    # Create the files to test
    sysbench --test=fileio --file-num=64 --file-total-size=4G prepare
    # Run the actual test
    sysbench --test=fileio --file-num=64 --file-total-size=4G --file-test-
mode=seqwr --max-time=60 run
    # Cleanup the test files
    sysbench --test=fileio cleanup
    # Drop the cache
    sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done

echo "File IO Test in Sequential Read Mode"
echo "-----------------------------------"
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    # Create the files to test
    sysbench --test=fileio --file-num=64 --file-total-size=4G prepare
    # Run the actual test
    sysbench --test=fileio --file-num=64 --file-total-size=4G --file-test-
mode=seqrd --max-time=60 run
    # Cleanup the test files
    sysbench --test=fileio cleanup
    # Drop the cache
```

```
    sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done
```

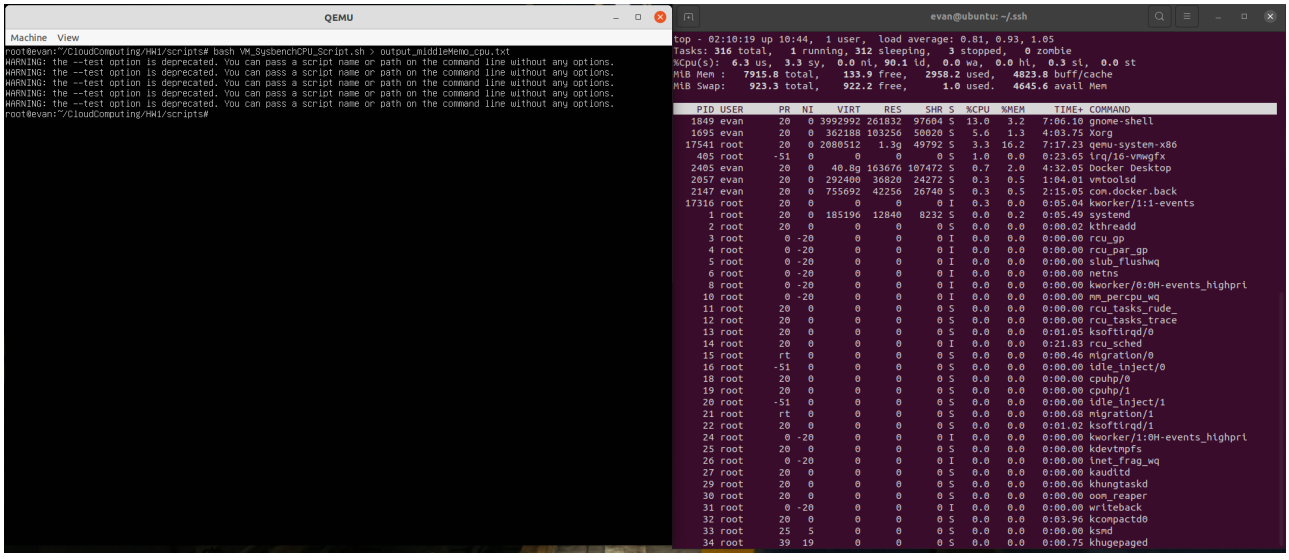
QEMU VM Performance Testing Results

CPU Tests

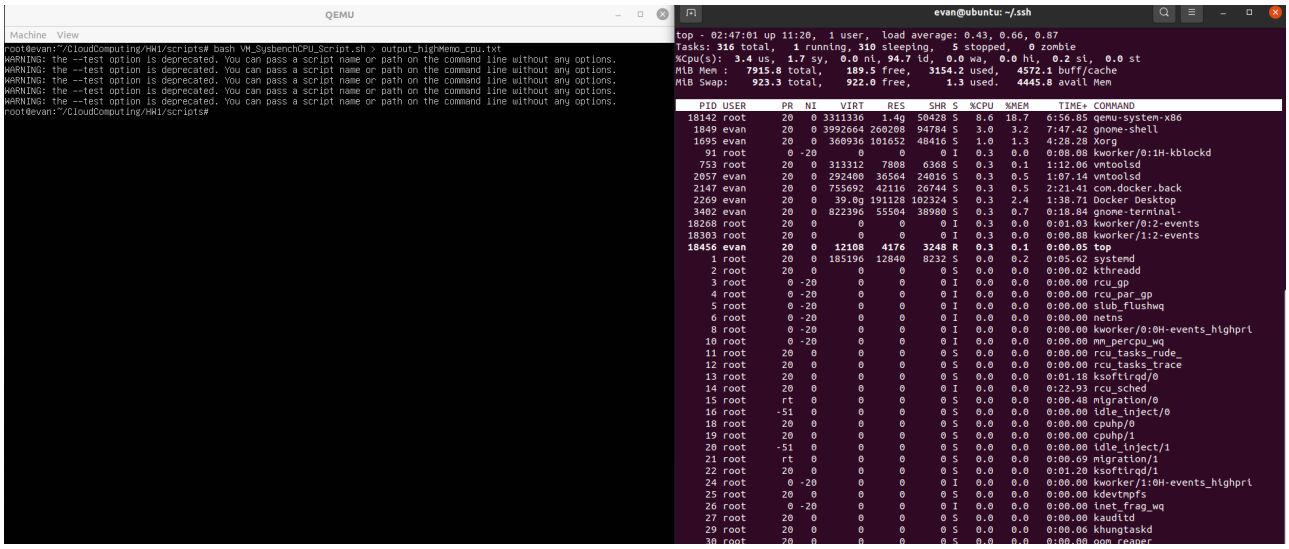
1. VM CPU Test with 512MB Memory



2. VM CPU Test with 1GB Memory



3. VM CPU Test with 2GB Memory



Low Memory (512 MB)

512 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	459.19	13348	22.10	0.83
min	434.37	13028	21.20	0.76
max	487.37	14625	22.10	1.1
std	5.83	189.80	0.23	0.23

Middle Memory (1GB)

1024 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	476.40	14876	22.12	0.90
min	475.46	14267	22.00	0.80
max	481.77	14457	22.30	1.20
std	6.25	176	0.49	0.62

High Memory (2GB)

2048 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	456.73	14406	25.16	1.16
min	478.00	14343	24.60	0.80
max	485.01	14556	25.90	1.70
std	4.92	132	0.49	0.34

From the data we can know that as the memory increases, the CPU performance improves slightly, but the difference is not large.

File I/O Tests

Low Memory (512 MB)

512 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	1046.86	0.57	0.98	5957.33	0.16	0.91
min	965.81	0.51	0.97	5502.24	0.15	0.91
max	1161.96	0.62	0.98	6198.67	0.17	0.91
std	76.26	0.04	0.00	293.42	0.01	0.00

Middle Memory (1GB)

1024 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	1144.23	0.52	0.97	5725.62	0.16	0.91
min	1134.34	0.51	0.97	4349.58	0.14	0.90
max	1155.40	0.52	0.98	6311.87	0.21	0.91
std	9.60	0.00	0.00	783.12	0.03	0.00

High Memory (2GB)

2048 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	1157.93	0.51	0.97	5585.14	0.17	0.91
min	1136.67	0.50	0.97	4338.04	0.15	0.89
max	1182.00	0.52	0.97	6146.01	0.21	0.93
std	18.79	0.01	0.00	779.56	0.03	0.01

From above data we can know that, reading speed is significantly faster than writing, and the disk utility of reading is also slightly lower than that of writing. As memory increases, writes are slightly faster and reads are slightly slower.

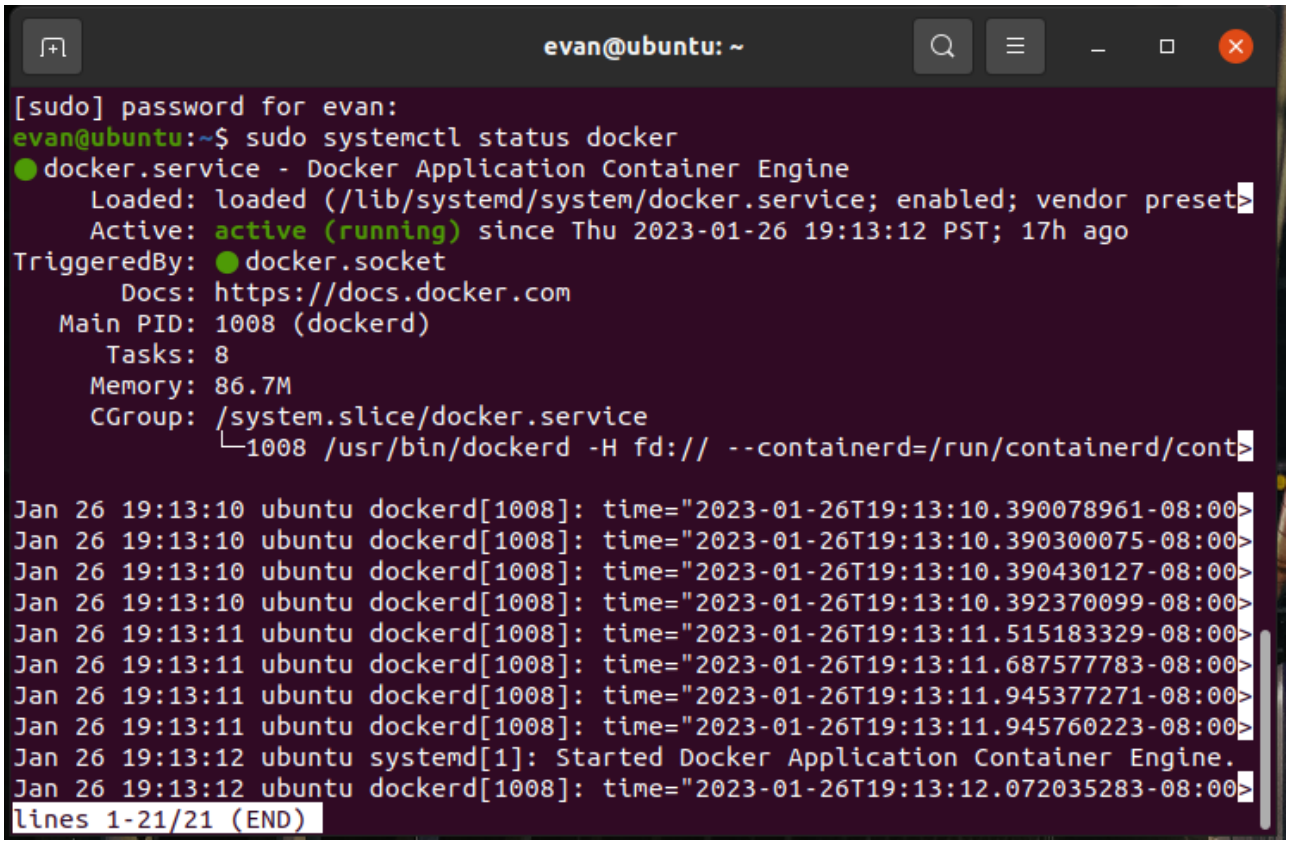
Docker Container Setup

1. Install Docker, including Docker Engine, containerd, and Docker Compose. I used steps from "Set up the repository" on https://docs.docker.com/engine/install/ubuntu/.

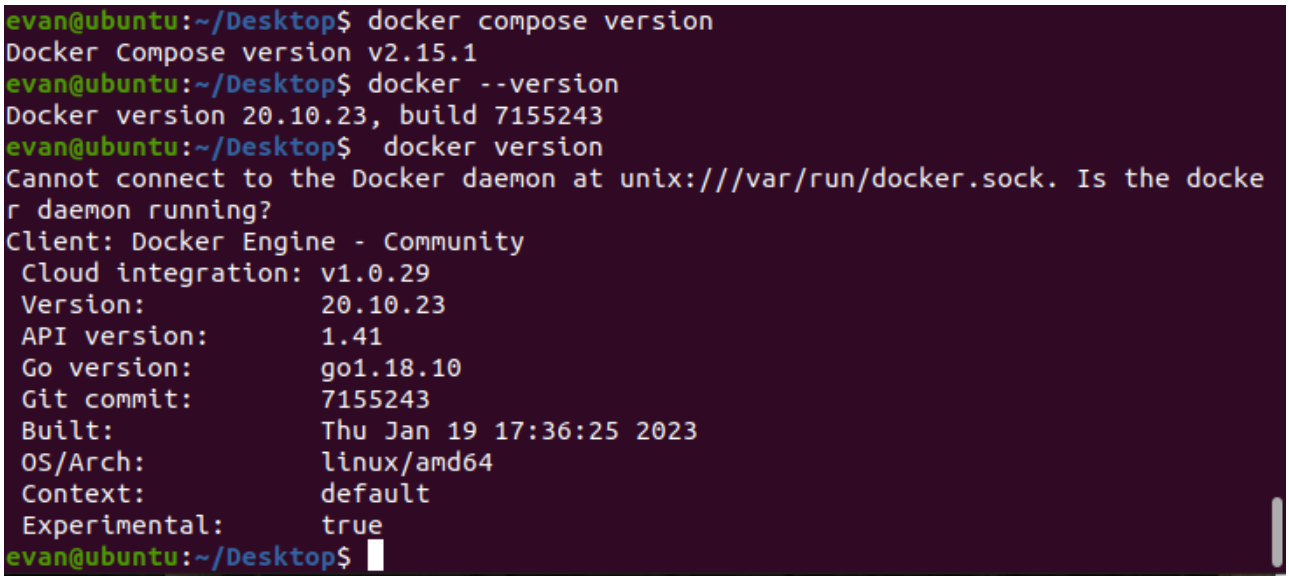
2. Verify Docker is installed and running correctly.

```
$ sudo systemctl start docker
```

```
$ sudo systemctl status docker
```



And also check the Docker version.



3. Create a Unix group called `docker` and add users to it. Then restart Linux.

   ```
   $ sudo groupadd docker

   $ sudo usermod -aG docker $USER

   $ newgrp docker
   ```

4. Pull ubuntu image. (https://hub.docker.com/_/ubuntu)

5. Create a container from ubuntu image and run it.

```
$ docker run -it ubuntu bin/bash
```

6. Install sysbench in container.

```
apt update

apt install sysbench
```

7. Create my own image.

```
$ docker commit -m "sysbenchcontainer" 350973ea83e6 ubuntu_sysbench:v1.0
```

8. Check image.

```
$ docker image ls
```

```
evan@ubuntu:~/Desktop$ docker ps
CONTAINER ID   IMAGE     COMMAND      CREATED             STATUS             POR
TS     NAMES
3d143f0988f4   ubuntu    "bin/bash"   About an hour ago   Up About an hour
       epic_pare
evan@ubuntu:~/Desktop$ docker commit 3d143f0988f4 ubuntu_sysbench:v1.0
sha256:55cad855ae6274b11eb9b090d391519a7c543c50f0c20cddfe56691825c4e317
evan@ubuntu:~/Desktop$ docker image ls
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
ubuntu_sysbench   v1.0      55cad855ae62   20 seconds ago   129MB
ubuntu            latest    6b7dfa7e8fdb   7 weeks ago      77.8MB
hello-world       latest    feb5d9fea6a5   16 months ago    13.3kB
evan@ubuntu:~/Desktop$
```

9. Operations for Docker container management:

`$ docker container ls` - List containers

`$ docker image ls` - List images

`$ docker run`, `$ docker container run` - Run a command in a new container

`exit` or `ctrl-D` - Exit out of the docker container bash shell

`$ docker container create` - Create a new container

`$ docker container update` - Update configuration of one or more containers

`$ docker container commit` - Create a new image from a container's changes

`$ docker container cp` - Copy files/folders between a container and the local filesystem

`$ docker container exec` - Run a command in a running container

10. Install git on the Linux host(VM on my windows PC), set up the GitHub authorization with ssh and pull the repository to the local Linux host.

Docker Container Performance Testing with Varying Configurations

the scenarios is that I will change docker container memory from 512 to 2o48 MB ,and I will analyze their performance by getting the sysbench measurement.

Steps

1. Update the contain_config in scripts and pull updated repository to the local host.

```
$ git pull origin main
```

2. Run Docker container, each time with different configurations. I used `--privileged` to drop cache.

```
$ docker run --rm -it --privileged --cpus="2" -m 512m ubuntu_sysbench:v1.0
bin/bash

$ docker run --rm -it --privileged --cpus="2" -m 1024m ubuntu_sysbench:v1.0
bin/bash

$ docker run --rm -it --privileged --cpus="2" -m 2048m ubuntu_sysbench:v1.0
bin/bash
```

3. Run the `env` command in container to display container environment.

4. Open another terminal, and copy the script files to be executed to the container.

```
$ docker cp ./SCU_COEN241/HW1/Scripts/script_file_name.sh
container_id:/script_file_name.sh
```

5. For CPU test, open another terminal in the host (Linux, Ubuntu desktop) and use the 'top' command line tool to get the CPU utilization, including user-level and kernel-level.

```
$ top
```

6. Run each shell script and save results.

```
bash /script_file_name.sh > /output_file_name.txt
```

7. Copy the output files from the container. Then exit container.

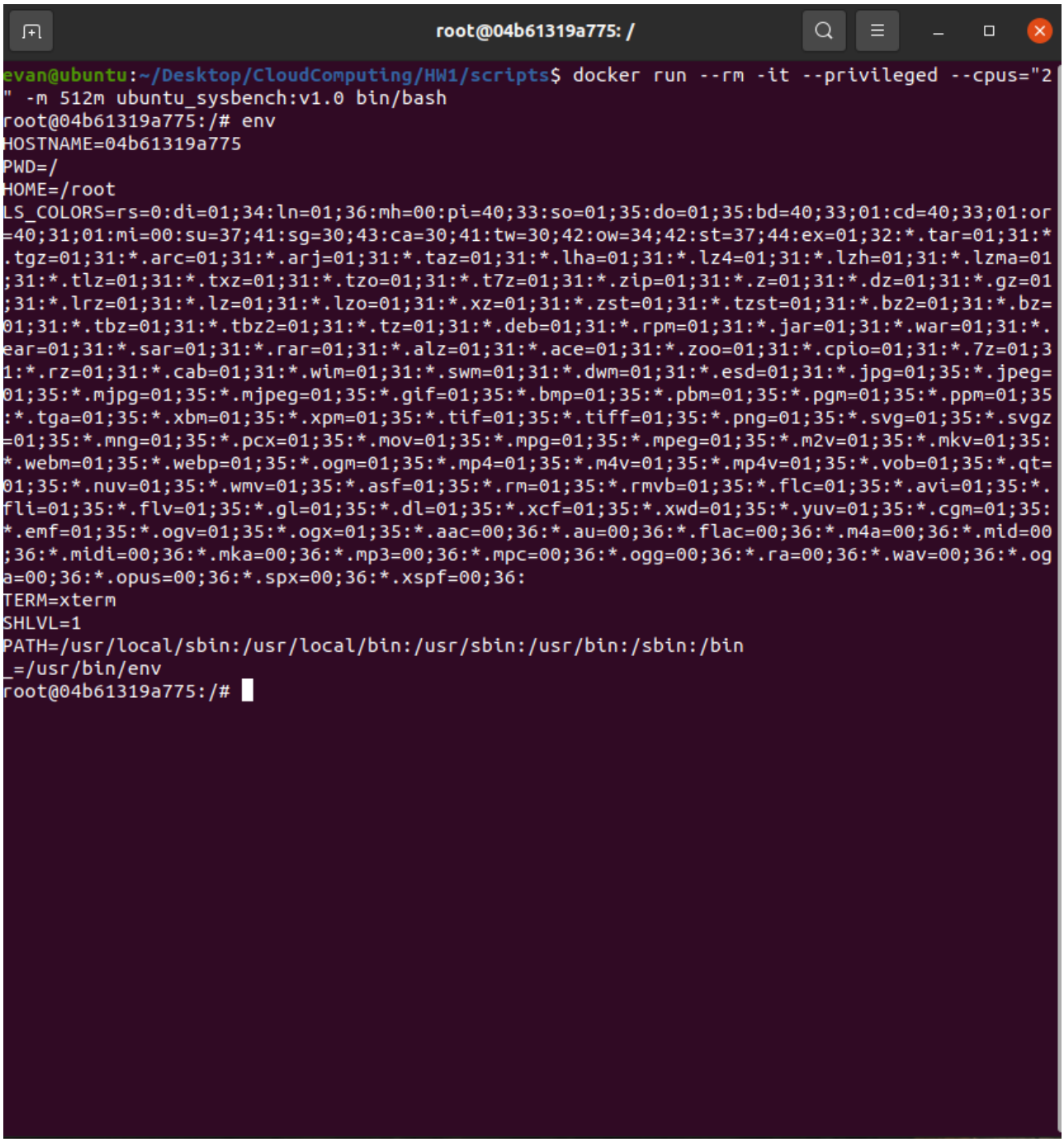
```
$ docker cp container_id:/output_file_name.txt
./SCU_COEN241/HW1/Container_Results/output_file_name.txt

exit
```

8. Push all output files to the repository.

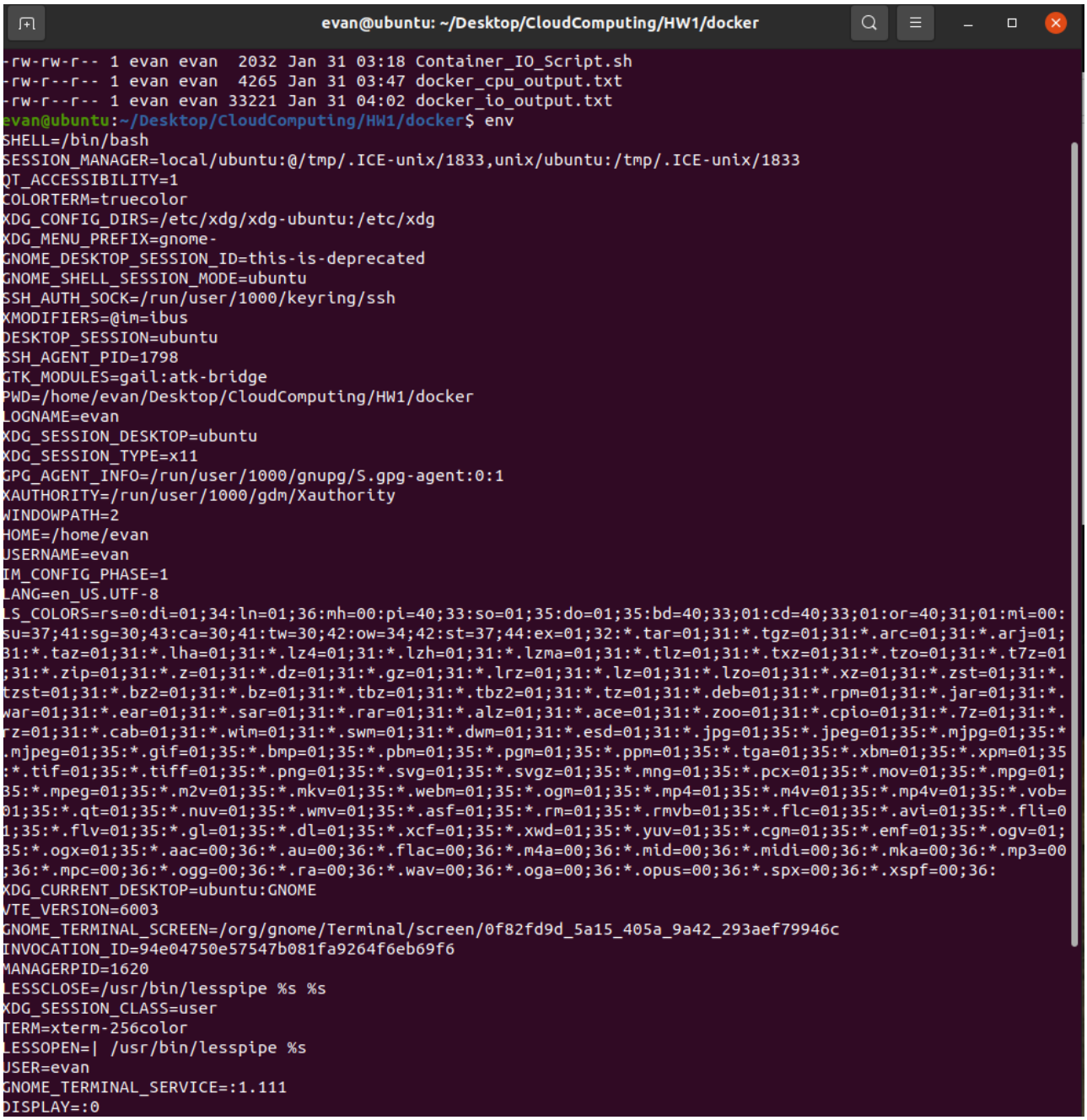
```
$ cd SCU_COEN241

$ git add .

$ git commit -m "Add result"

$ git push origin main
```

Running Environments

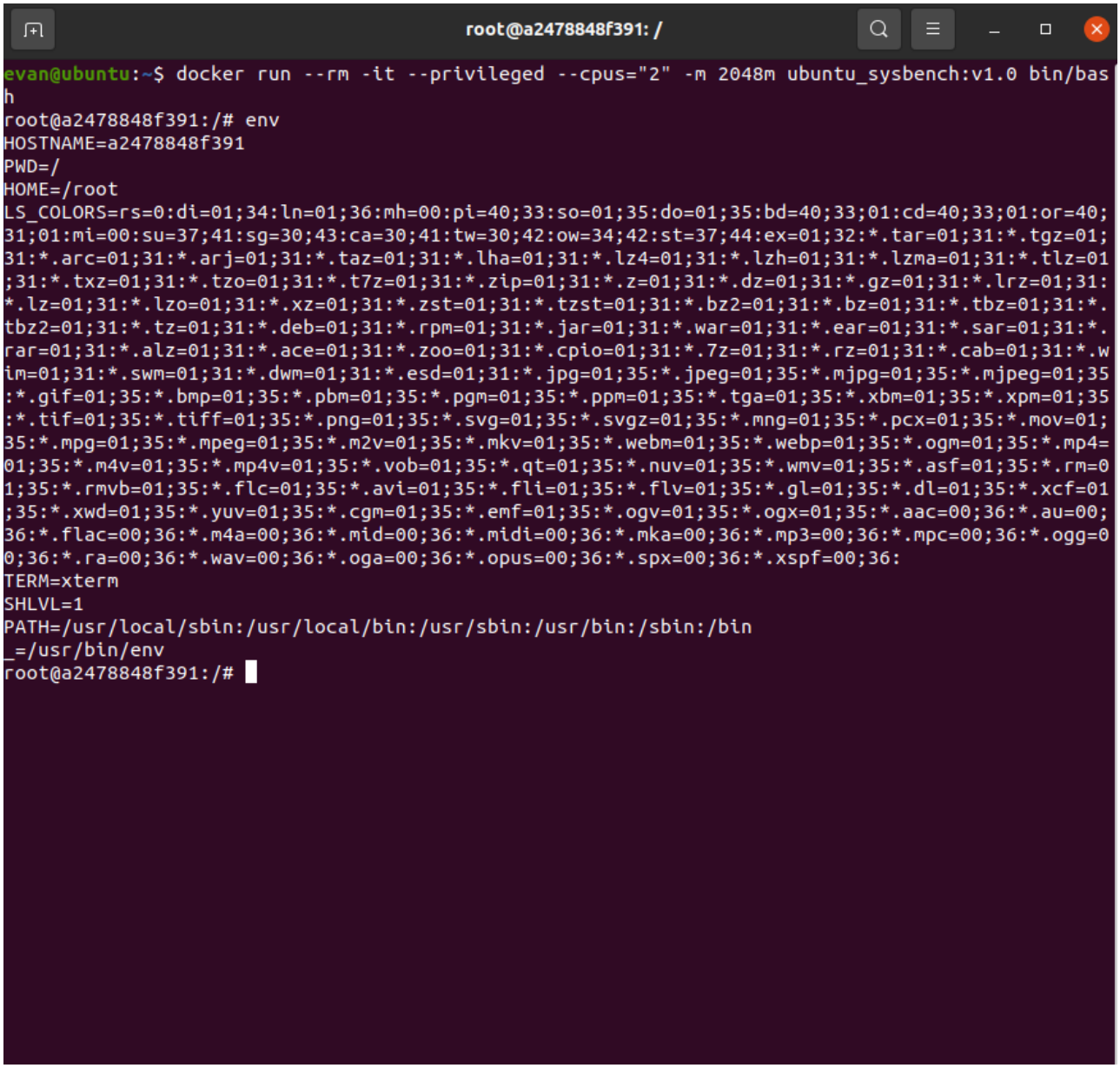
- Low-memory test



- Middle-memory test



- High-memory test



Shell Scripts

- Container Sysbench CPU Test Script

```
#!/bin/bash



#Arg = 0 - low-memory config, 512 MB
#Arg = 1 - middle-memory config, 1 GB
#Arg = 2 - high-memory config, 2GB

contain_config=$(( 0 ))

if [ $contain_config -eq 0 ];
then
    echo "Low-memory Docker Container Sysbench CPU Tests: 2 core, 512 MB Memory"
    echo "============================================================="
```

```
elif [ $contain_config -eq 1 ];
then
    echo "Middle-memory Docker Container Sysbench CPU Tests: 2 core, 1024
MB Memory"
    echo
"========================================================================"

elif [ $contain_config -eq 2 ];
then
    echo "High-memory Docker Container Sysbench CPU Tests: 2 core, 2048 MB
Memory"
    echo "========================================================================"

else
    echo "Incorrect Memory to run sysbench, needs to match preset
configurations"
    exit 1
fi

# Run sysbench CPU test for 5 times on each VM configuration
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    echo "------"
    sysbench --test=cpu --cpu-max-prime=10000 --time=30 run
done
```

- Docker Sysbench File IO Test Script

```
#!/bin/bash



#Arg = 0 - low-memory config, 512 MB
#Arg = 1 - middle-memory config, 1 GB
#Arg = 2 - high-memory config, 2GB

contain_config=$(( 0 ))

if [ $contain_config -eq 0 ];
then
    echo "Low-memory Docker Container Sysbench CPU Tests: 2 core, 512 MB
Memory"
    echo "========================================================="

elif [ $contain_config -eq 1 ];
then
    echo "Middle-memory Docker Container Sysbench CPU Tests: 2 core, 1024
MB Memory"
    echo
"========================================================================"
```

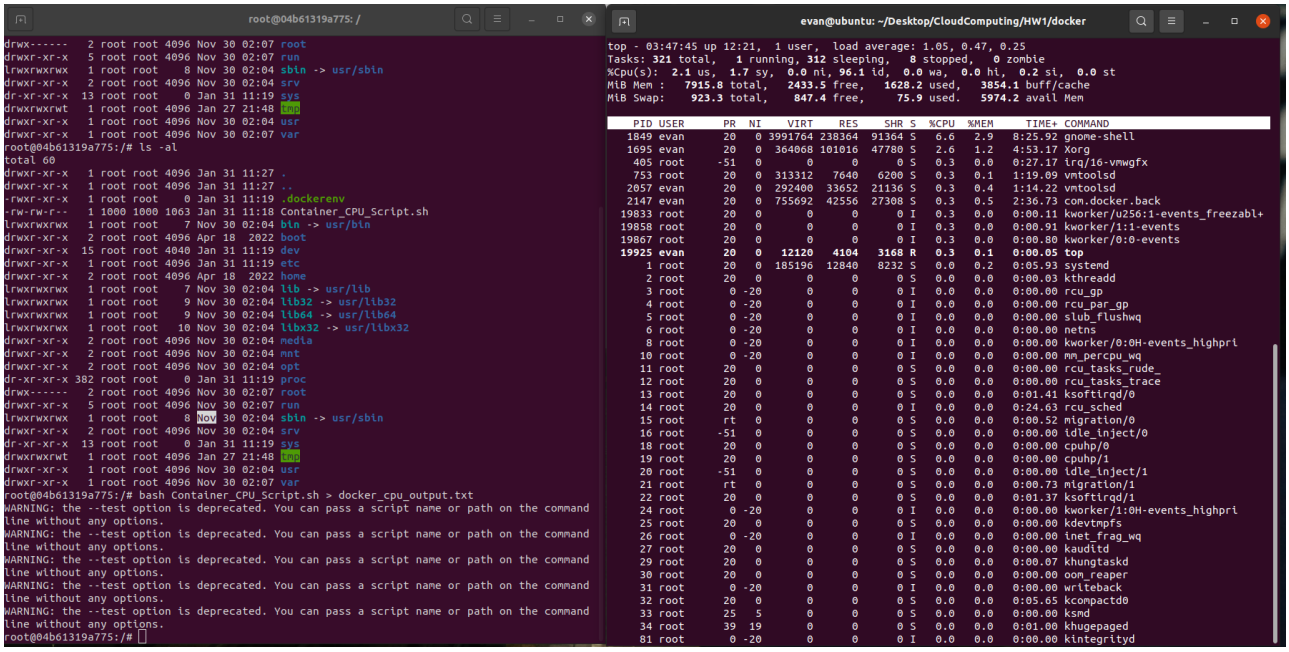
```
elif [ $contain_config -eq 2 ];
then
    echo "High-memory Docker Container Sysbench CPU Tests: 2 core, 2048 MB
Memory"
    echo "========================================================================"

else
    echo "Incorrect Memory to run sysbench, needs to match preset
configurations"
    exit 1
fi

# Run sysbench file io tests in sequential write mode and sequential read
mode
# Each test is run 5 times on each VM configuration
echo "File IO Test in Sequential Write Mode"
echo "-------------------------------------"
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    # Create the files to test
    sysbench --test=fileio --file-num=64 --file-total-size=4G prepare
    # Run the actual test
    sysbench --test=fileio --file-num=64 --file-total-size=4G --file-test-
mode=seqwr --max-time=60 run
    # Cleanup the test files
    sysbench --test=fileio cleanup
    # Drop the cache
    sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done

echo "File IO Test in Sequential Read Mode"
echo "------------------------------------"
for ((counter = 1; counter < 6; counter++))
do
    echo "Test $counter"
    # Create the files to test
    sysbench --test=fileio --file-num=64 --file-total-size=4G prepare
    # Run the actual test
    sysbench --test=fileio --file-num=64 --file-total-size=4G --file-test-
mode=seqrd --max-time=60 run
    # Cleanup the test files
    sysbench --test=fileio cleanup
    # Drop the cache
    sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done
```

Docker Container Performance Testing Results

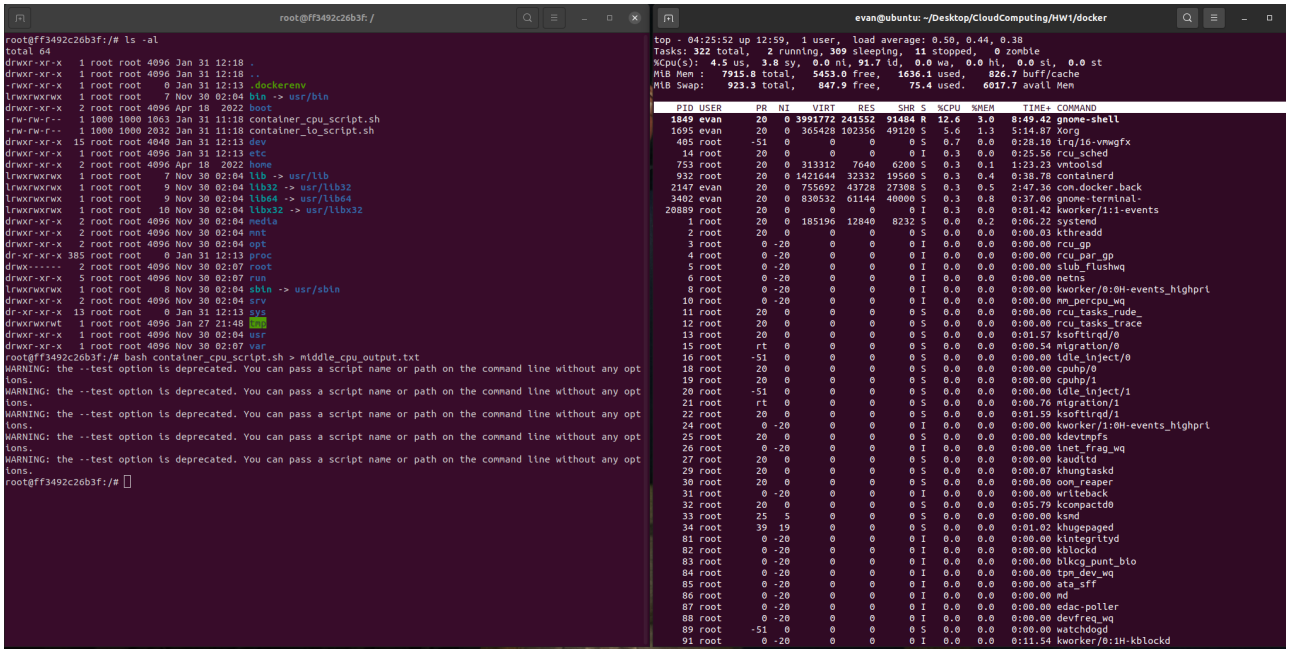
CPU Tests

Similar to the VM testing, a total of 15 sysbench were conducted across all 3 scenarios for the container testing. Screenshots of each test are shown below:

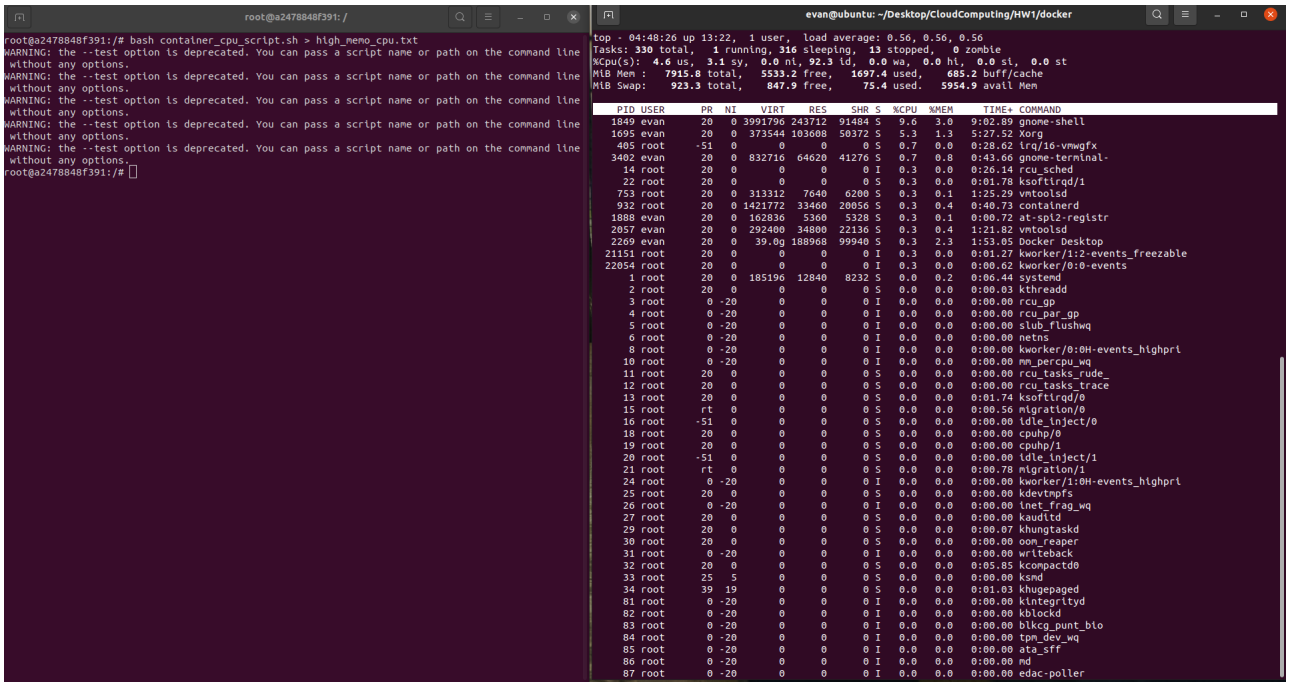
1. Container CPU Test with 512MB Memory



2. Container CPU Test with 1GB Memory



3. Container CPU Test with 2GB Memory



Low Memory (512 MB)

512 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	3162.26	97867.20	25.44	0.12
min	3263.62	97915.00	25.10	0.10
max	3305.28	99164.00	25.90	0.20
std	36.82	3104.79	0.30	0.04

Middle Memory (1GB)

1024 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	3278.23	31209.40	25.64	0.22
min	3220.13	96610.00	25.10	0.10
max	3300.57	99023.00	27.80	0.70
std	38.74	2842.29	1.21	0.27

High Memory (2GB)

2048 MB mem	Events per second	Total events	Kern. CPU (%)	User CPU (%)
average	3287.03	31414.60	25.34	0.14
min	3282.01	98466.00	25.10	0.10
max	3297.62	98935.00	26.20	0.20
std	40.01	2899.17	0.48	0.05

As shown in the above tables, when running a Docker container, the CPU performance is basically the same with different memory size. Perhaps because of the good performance of Docker, experiments with smaller memory are required to show the impact on CPU performance.

File I/O Tests

Similar to the VM testing, the file I/O tests, a total of 30 sysbench file I/O tests were conducted for the container testing. The performance data collected for each test is read and write speed (MiB/s), latency (ms), and disk utilization (%). Sysbench reports the I/O throughput and latency, while disk utilization was calculated by dividing the latency by the total time.

Low Memory (512 MB)

512 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	36882.15	0.01	0.99	52702.23	0.05	0.99
min	35043.52	0.01	0.99	51852.94	0.05	0.99
max	37393.95	0.01	0.99	54379.08	0.05	0.99
std	252.75	0.001	0.00	451.97	0.00	0.00

Middle Memory (1GB)

1024 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	37975.52	0.01	0.99	53276.41	0.02	0.99
min	37634.99	0.01	0.99	55026.58	0.02	0.98
max	38025.09	0.01	0.99	52779.33	0.02	0.99
std	323.50	0.01	0.00	1479.53	0.00	0.00

High Memory (2GB)

2048 MB mem	Write (MiB/s)	Latency (ms)	Disk Util. (%)	Read (MiB/s)	Latency (ms)	Disk Util. (%)
average	38920.53	0.01	0.99	55103.32	0.02	0.99
min	38543.29	0.01	0.99	54163.94	0.02	0.99
max	39342l39	0.01	1.00	56806.30	0.02	0.99
std	842.34	0.02	0.00	3765.24	0.03	0.00

From above data we can get understanding that reading is also significantly faster than writing in the docker.

QEMU VM vs. Docker Container Performance

Comparing QEMU virtual machines's data and containers's data, we can know that the Docker container performance is so much better than the QEMU VM performance.