

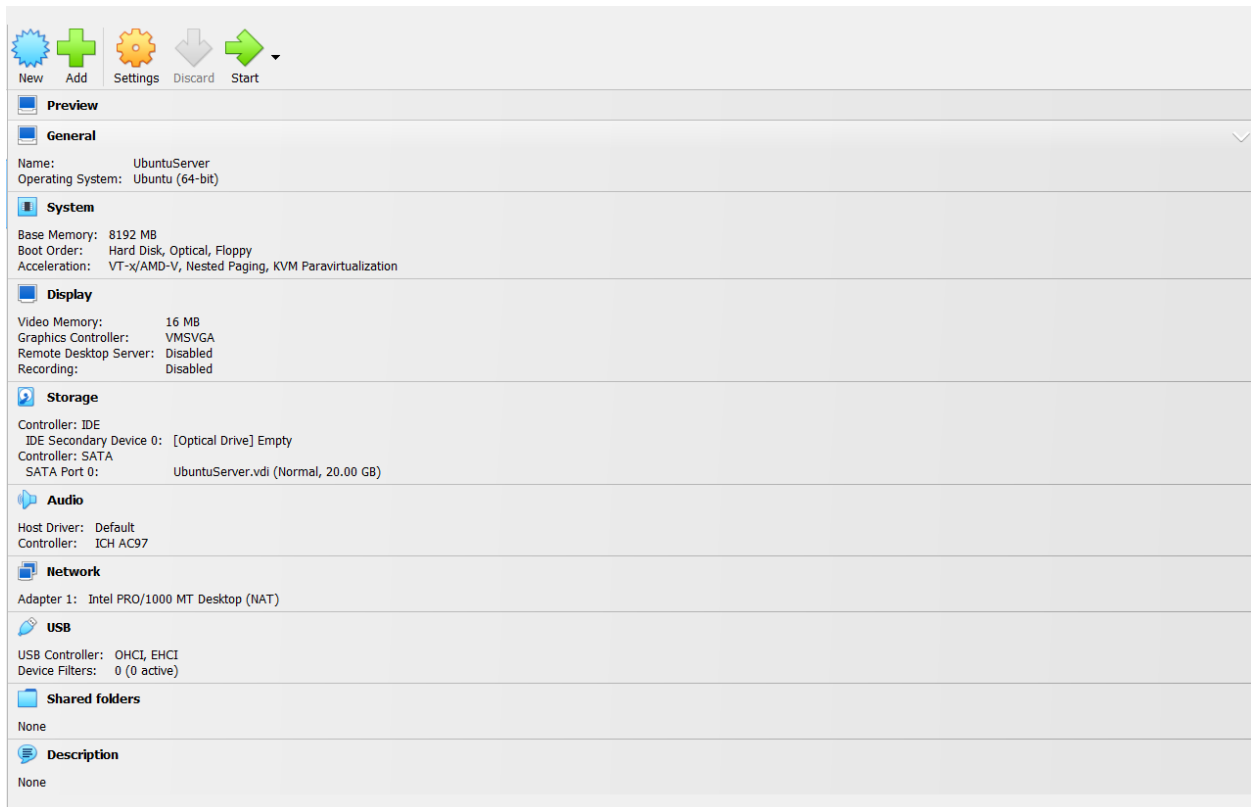
COEN 241 HW 1

System vs. OS Virtualization

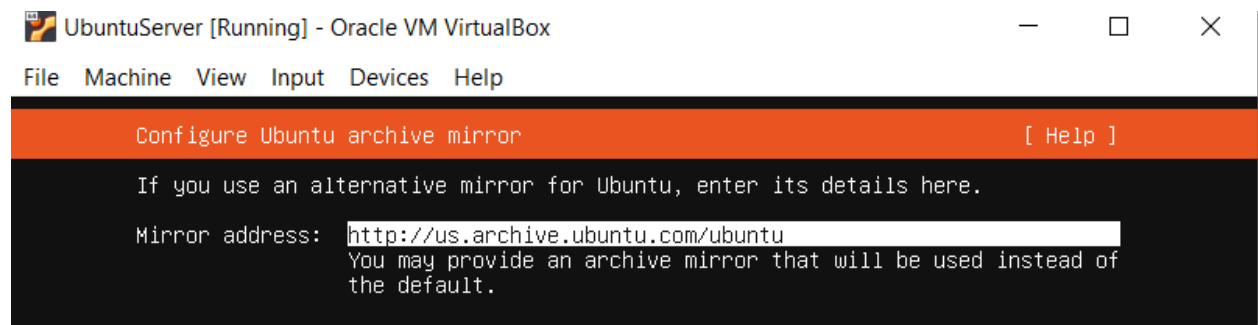
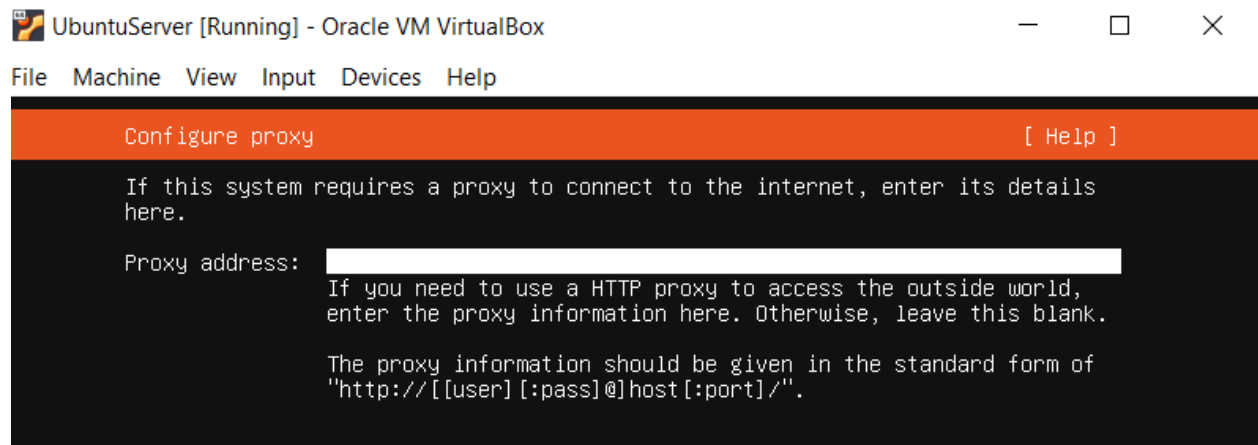
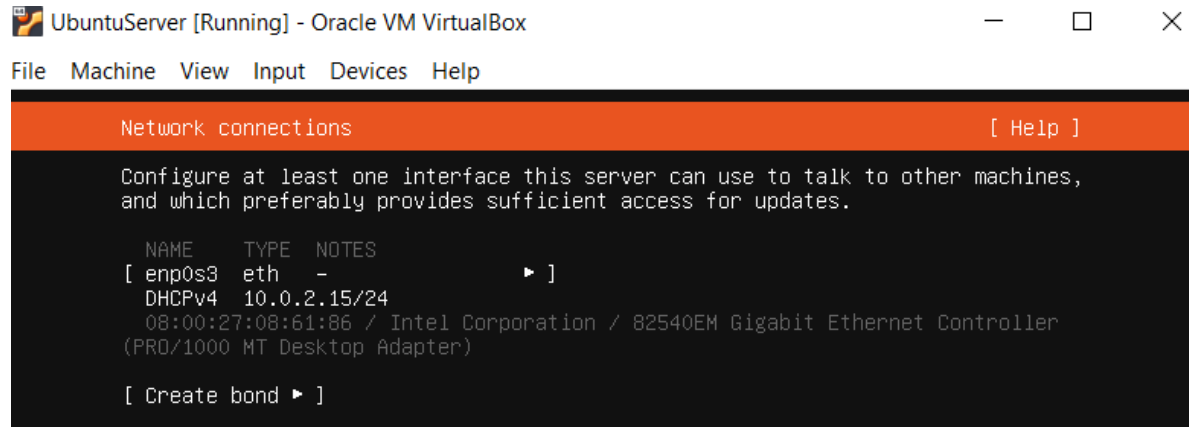
Justin Li

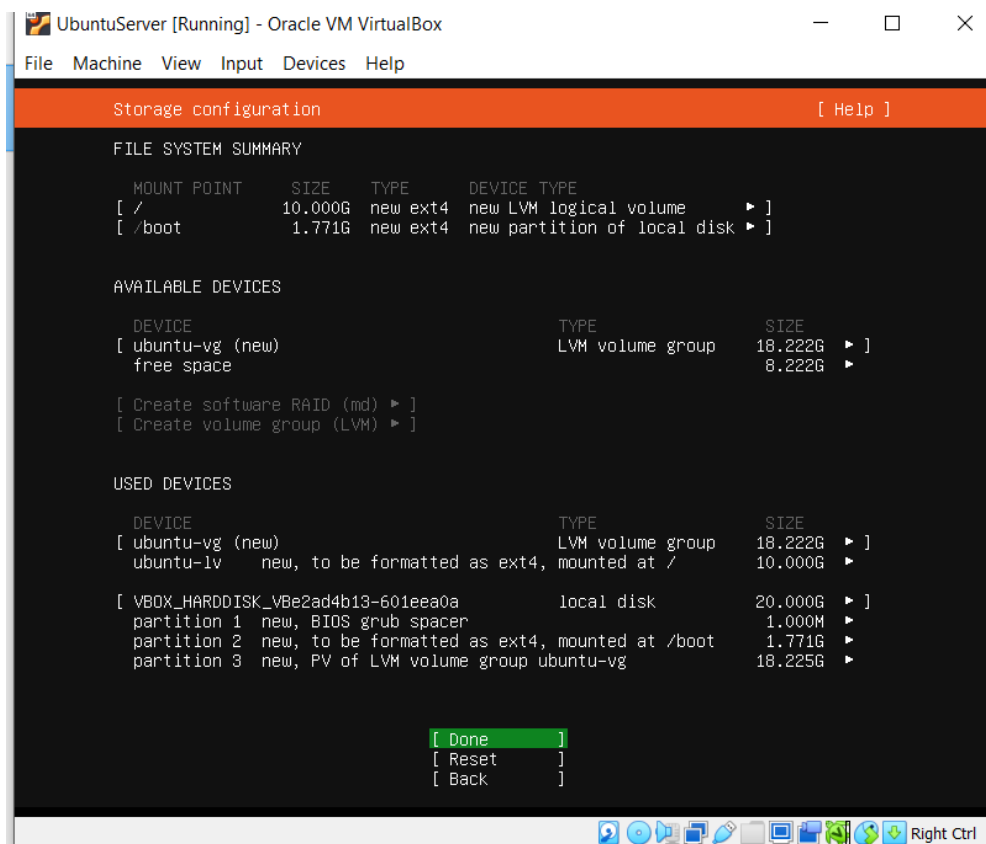
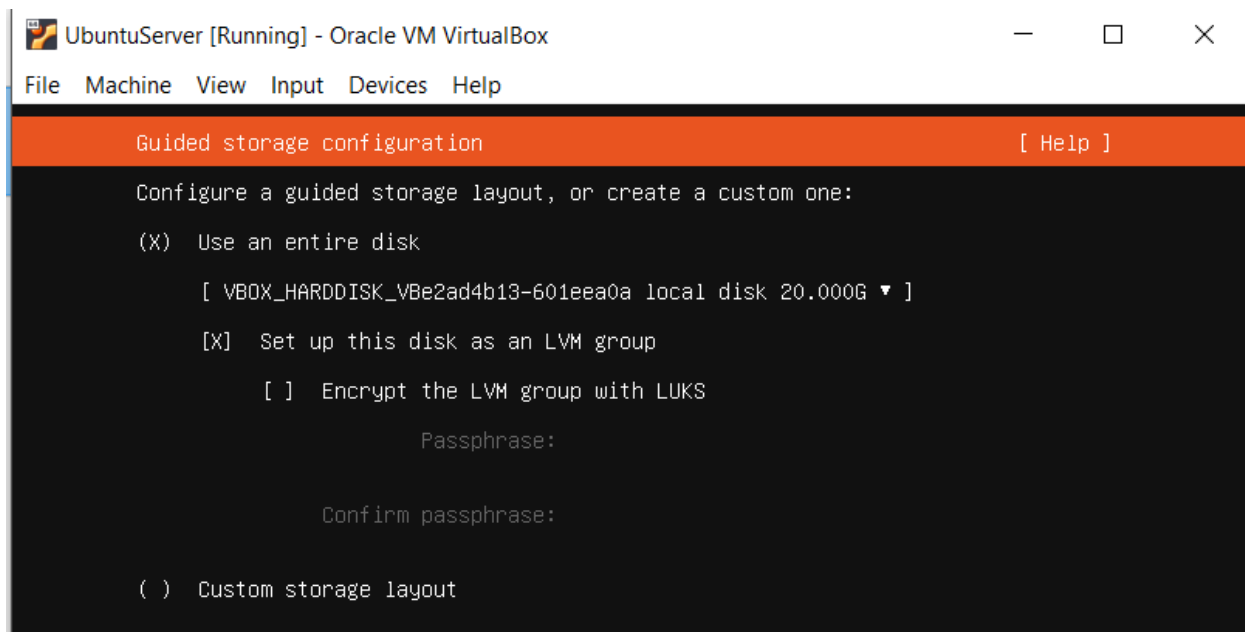
VM Setup and Configurations:

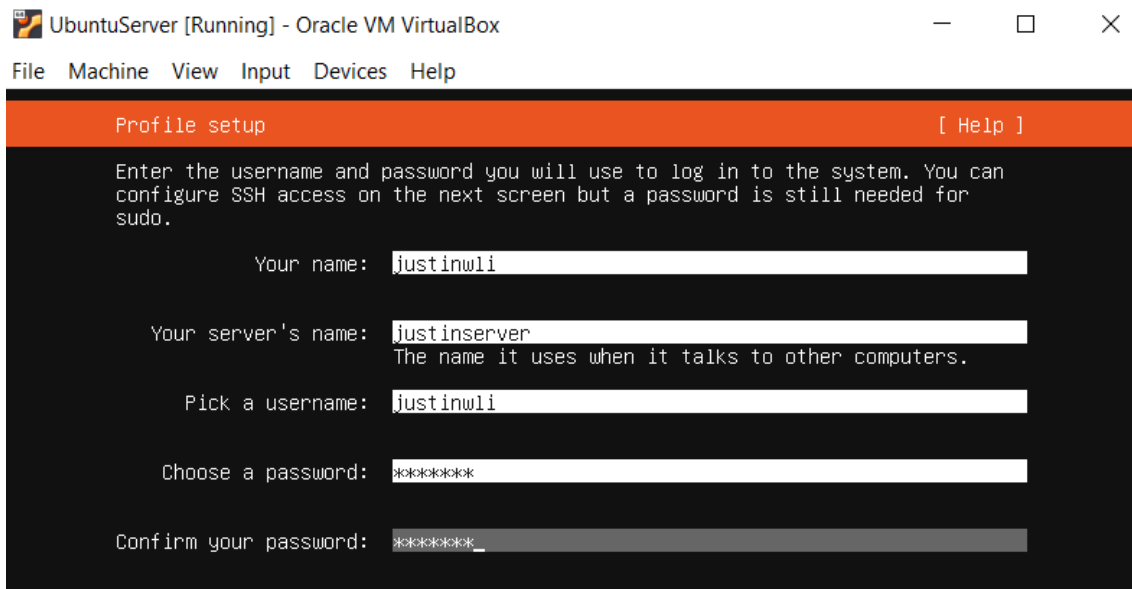
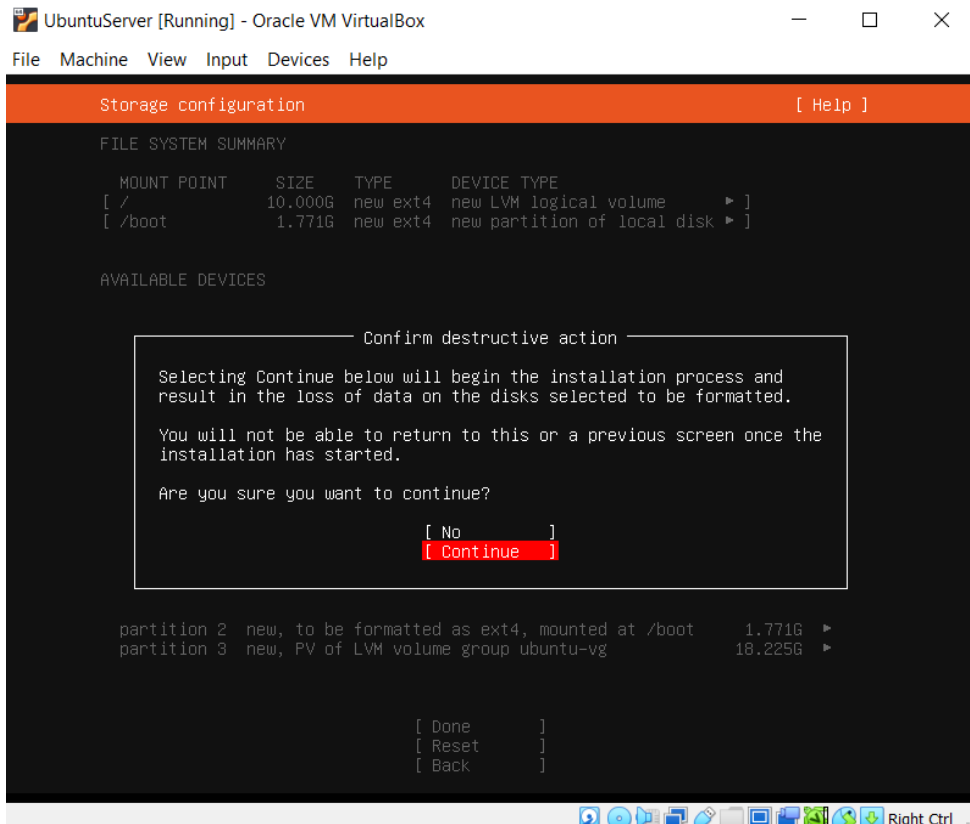
Because I was running on Windows and had difficulty setting up and running QEMU, I ended up using VirtualBox 7.0.2 in order to run the 20.04 Ubuntu Server ISO. Below is a screenshot of my configurations for my virtual machine. (I changed my base memory value according to the experiment test cases that I set up, as well as CPUs which I started at 1.)



Here are the steps that I used to run the Ubuntu server:







SSH Setup

[Help]

You can choose to install the OpenSSH server package to enable secure remote access to your server.

[] Install OpenSSH server

Import SSH identity: [No ▼]
You can import your SSH keys from GitHub or Launchpad.

Import Username:

[X] Allow password authentication over SSH

Featured Server Snaps

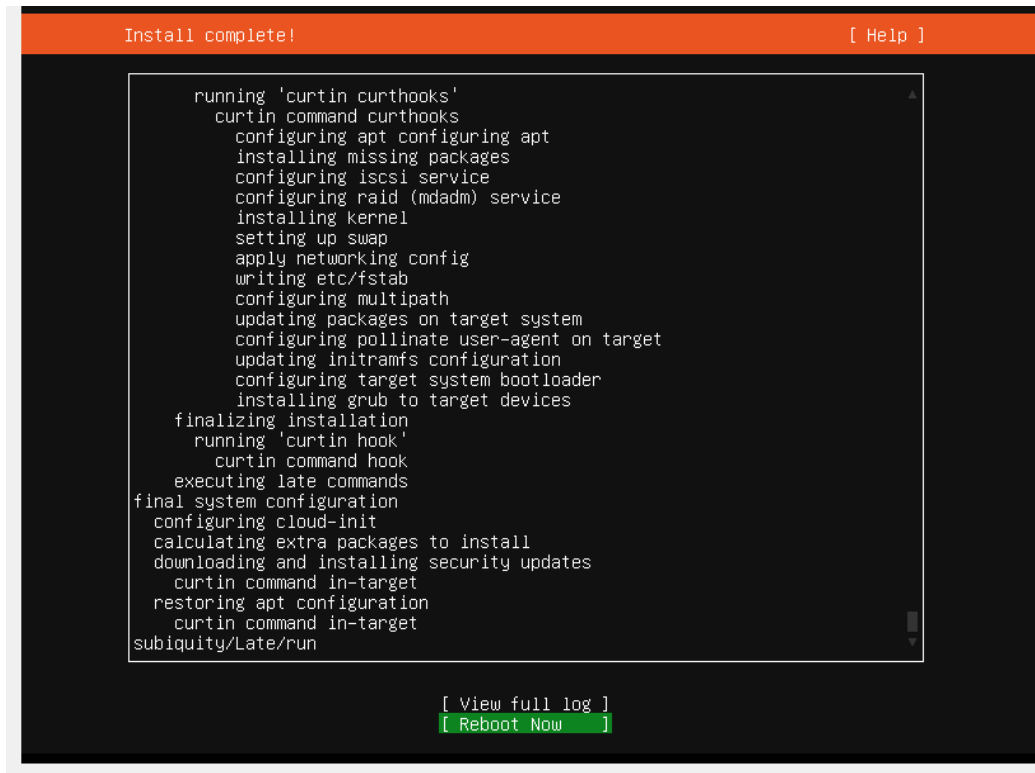
[Help]

These are popular snaps in server environments. Select or deselect with SPACE, press ENTER to see more details of the package, publisher and versions available.

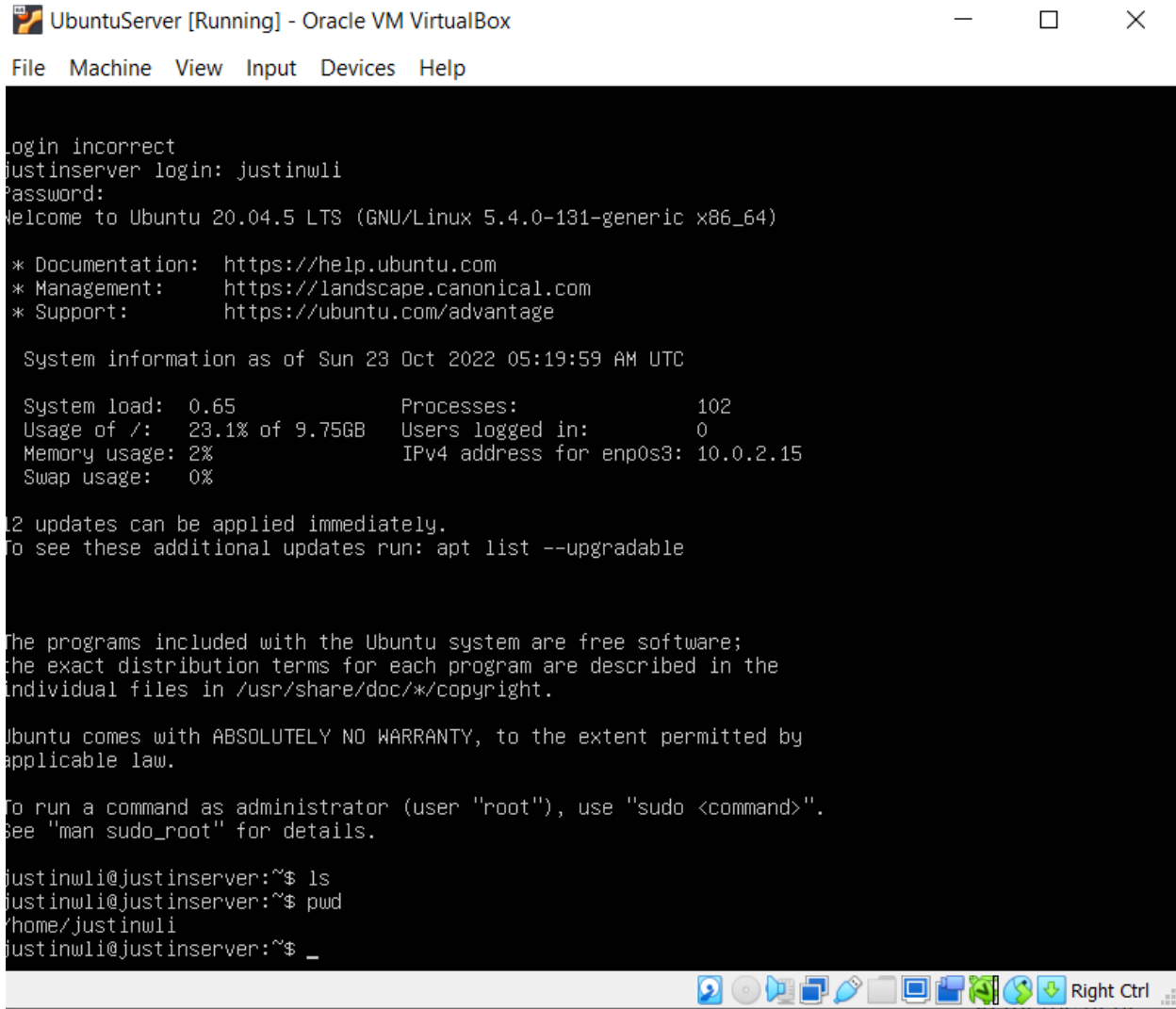
[] microk8s	Kubernetes for workstations and appliances	▶
[] nextcloud	Nextcloud Server - A safe home for all your data	▶
[] wekan	The open-source kanban	▶
[] kata-containers	Build lightweight VMs that seamlessly plug into the c	▶
[] docker	Docker container runtime	▶
[] canonical-livepatch	Canonical Livepatch Client	▶
[] rocketchat-server	Rocket.Chat server	▶
[] mosquitto	Eclipse Mosquitto MQTT broker	▶
[] etcd	Resilient key-value store by CoreOS	▶
[] powershell	PowerShell for every system!	▶
[] stress-ng	tool to load and stress a computer	▶
[] sabnzbd	SABnzbd	▶
[] wormhole	get things from one computer to another, safely	▶
[] aws-cli	Universal Command Line Interface for Amazon Web Servi	▶
[] google-cloud-sdk	Google Cloud SDK	▶
[] slcli	Python based SoftLayer API Tool.	▶
[] doctl	The official DigitalOcean command line interface	▶
[] conjure-up	Package runtime for conjure-up spells	▶
[] postgresql10	PostgreSQL is a powerful, open source object-relatio	▶
[] heroku	CLI client for Heroku	▶
[] keepalived	High availability VRRP/BFD and load-balancing for Lin	▶
[] prometheus	The Prometheus monitoring system and time series data	▶
[] juju	Juju - a model-driven operator lifecycle manager for	▶

[Done]

[Back]



Here is proof of my VM running environment:



```
login incorrect
justinserver login: justinwli
Password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-131-generic x86_64)

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

System information as of Sun 23 Oct 2022 05:19:59 AM UTC

System load:  0.65          Processes:           102
Usage of /:   23.1% of 9.75GB Users logged in:          0
Memory usage: 2%           IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

2 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

justinwli@justinserver:~$ ls
justinwli@justinserver:~$ pwd
/home/justinwli
justinwli@justinserver:~$ _
```

Docker Setup:

I downloaded Docker Desktop on Windows and used the command line in order to configure my image with sysbench. Here are the following commands which were used:

```
$ docker run --rm -it --entrypoint /bin/sh ubuntu:22.04
# apt update
# apt install sysbench
$ docker ps
$ docker commit 216c552ea5ba my_img_with_sysbench
$ docker images
$ docker history my_img_with_sysbench
```

Below is proof of my image creation on docker:

```
operable program or batch file.

C:\Users\ziban>docker commit b76be70e75ed my_img_with_sysbench
sha256:abad365955ab05f7e7a59b80e8461de076495b24275a41f6e6702f6106ccf527

C:\Users\ziban>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
my_img_with_sysbench latest          abad365955ab    6 seconds ago  133MB
docker101tutorial   latest         057fcfc77770c  44 minutes ago  28.9MB
alpine/git          latest         42a1cda0ba24   3 days ago     43.6MB
ubuntu              20.04          817578334b4d   2 weeks ago    72.8MB
hello-world         latest         feb5d9fea6a5    13 months ago  13.3kB

C:\Users\ziban>docker history my_img_with_sysbench
IMAGE              CREATED          CREATED BY
SIZE              COMMENT
abad365955ab      47 seconds ago
59.8MB
817578334b4d      2 weeks ago     /bin/sh -c #(nop)  CMD ["bash"]
0B
<missing>         2 weeks ago     /bin/sh -c #(nop)  ADD file:8faed18d471598732...
72.8MB
```

Experiments:

For this assignment we are testing and comparing the performance of system virtualization (via Virtualbox/QEMU) vs. OS virtualization (via Docker). For each environment, I also changed around some of the settings to produce different test cases. For instance, I changed the CPU count and the amount of RAM allocated to see if they would produce different results. In Docker, I set the argument “--memory=[“memory_limit”]” to change the amount of RAM, and set the cpus=[“CPU_count”] to change the amount of CPUs. For VirtualBox, I simply changed these settings using the GUI before booting up the Ubuntu server. The results of each individual test can be viewed in the Github in the respective folders.

In each environment, I tested under the CPU and FileI/O test modes of sysbench. For CPU, I set the value of --cpu-max-primes=10000. For FileI/O, I set the value of --file-total-size=“1G.” For both modes, I let each run for 15 seconds before returning the sysbench results. For the CPU tests, I decided to collect the events/sec to measure CPU speed. For the FileI/O tests, I kept track of both the read and write throughputs as well as the latency to measure overall performance, which are shown in the tables below.

Test Case #1: 1 CPU & 8GB RAM

This was my default starting point for my experiments in both virtualization environments.

Test Case #2: 1 CPU & 1GB RAM

For my 2nd case, I wanted to see the results of lowering the amount of RAM available, and how the performance would be affected.

Test Case #3: 4 CPUs & 8GB RAM

And for the last case, I increased the CPU count to 4 and raised the RAM back to its starting value to compare how adding more CPUs would affect the performance.

Shell Scripts:

For CPU tests:

```
#!/bin/sh
```

```
sysbench --test=cpu --time=15 --cpu-max-prime=10000 run
```

For FileI/O tests:

```
#!/bin/sh
```

```
sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw prepare
```

```
sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --time=15 --max-requests=0 run
```

```
sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw cleanup
```

For windows:

```
sync
```

For linux:

```
echo3 > /proc/sys/vm/dropcaches
```

Results:

Docker - Case #1: 1 CPU & 8GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	988.5	33.45	22.3	0.11
max	1045.92	36.07	24.05	0.12
avg	1013.05	34.96	23.31	0.11
std	18.52	1.16	0.77	~0

VM - Case #1: 1 CPU & 8GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	891.47	6.33	4.22	0.59
max	928.68	6.94	4.62	0.64

avg	907.2	6.7	4.47	0.61
std	15.46	0.21	0.14	0.02

Docker - Case #2: 1 CPU & 1GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	975.12	32.32	21.84	0.12
max	1060.07	34.81	23.21	0.13
avg	1019.1	33.81	22.46	0.12
std	29.42	0.96	0.64	~0

VM - Case #2: 1 CPU & 1GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	823.94	4.89	3.26	0.68
max	916.92	6.03	4.02	0.84
avg	886.32	5.54	3.7	0.74
std	32.5	0.51	0.34	0.07

Docker - Case #3: 4 CPU & 8GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	999.91	32.44	21.5	0.11

max	1027.84	37.7	25.13	0.13
avg	1012.96	33.78	22.52	0.12
std	9.33	2	1.34	0.01

VM - Case #3: 4 CPU & 8GB RAM				
	CPU	FileI/O		
	Events/sec	Rd Throughput (Mib/sec)	Wr Throughput (Mib/sec)	Avg Latency (ms)
min	971.71	6.56	4.37	0.58
max	1024.93	7	4.67	0.62
avg	989.61	6.81	4.54	0.6
std	18.34	0.16	0.11	0.01

Analysis:

Overall, in comparing the CPU performance of Docker vs VirtualBox, we can see that Docker outperforms by a little bit in each test case, having a slightly higher CPU speed each time. However, when it comes to FileI/O performance, Docker has a significantly higher throughput value for both reads and writes and a much lower latency in every run, making it the clear winner for FileI/O performance.

When it comes to comparing the individual test cases, we can see that regarding the CPU performance, Docker actually didn't seem to improve when increasing the CPU count from 1 to 4, as each of the CPU runs ended up with almost the same results. But in VirtualBox, there was definitely a higher performance, raising the average by about 80 events/second when increasing the CPU count. Regarding the FileI/O test cases, again in docker, there seemed to be very little difference between each of the experimental runs. However, when we lowered the amount of RAM allocated to the VM, we can clearly see a drop in throughput in the second test case, having about a 20% decrease compared to the other cases.

In conclusion, Docker seemed to outperform VirtualBox in terms of both CPU and FileI/O performance, having slightly higher CPU speed, while having much higher throughput and much lower latency compared to the VM. In addition, Docker seemed to have very little change when changing the memory allocation and CPU count, implying that it needs very little in order to run effectively. Meanwhile, changing the RAM and CPU count for VirtualBox made a significant difference for both CPU speed and FileI/O performance, suggesting that these amounts make a major difference when running a VM and its overall performance.