System vs OS Virtualization

Rohnie Baskar

Student ID: 1629049

Introduction

This report includes the real world use case of two virtualization techniques / technologies:

- System Virtualization via QEMU
- OS Virtualization via Docker

The operations related to each technology are discussed and their performance differences are demonstrated using benchmarks and the results are finally compared.

Environment

The environment for this assignment is use of personal computer with the following configurations

QEMU

QEMU is an open source machine emulator and a virtualizer. It's a hypervisor and emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems. For this experiment, we will be using Ubuntu as the guest OS.

SETUP - INSTALLATION

The below were the steps followed to install QEMU as the hypervisor and have Ubuntu as Guest OS on top of it.

1. Downloaded the .iso file for ubuntu version 20.04.5
2. Installed qemu using the following command

   brew install qemu

```
(base) Rohnies-MacBook-Air:~ rohnie$ qemu-system-aarch64 --version
QEMU emulator version 7.2.0
Copyright (c) 2003-2022 Fabrice Bellard and the QEMU Project developers
(base) Rohnies-MacBook-Air:~ rohnie$
```

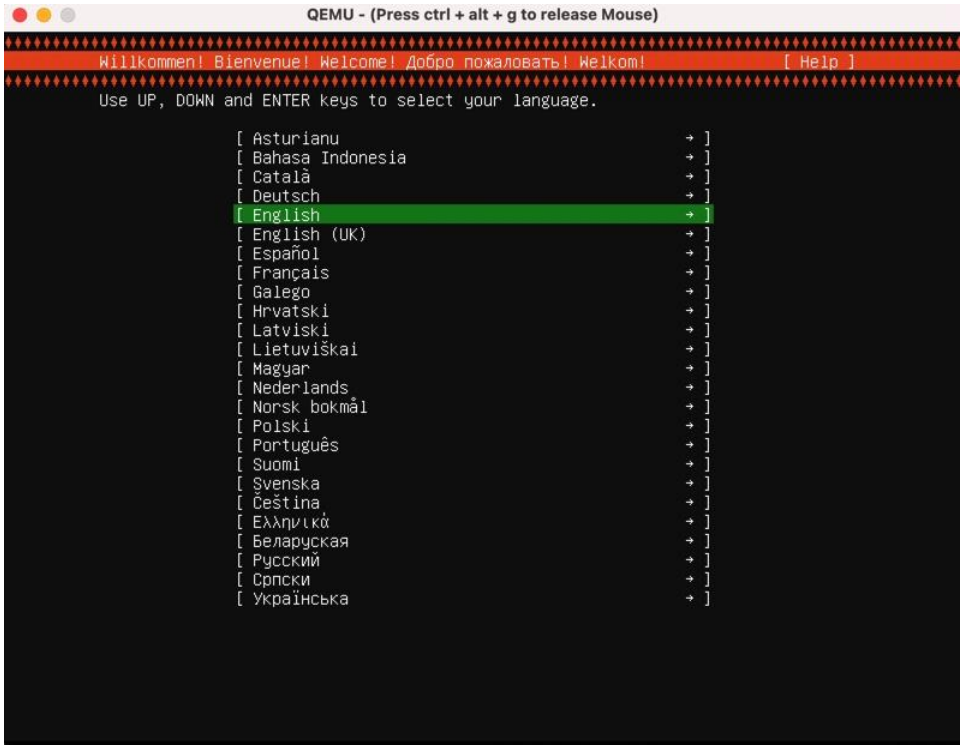
3. Finally Ubuntu was setup in the qemu using the following commands

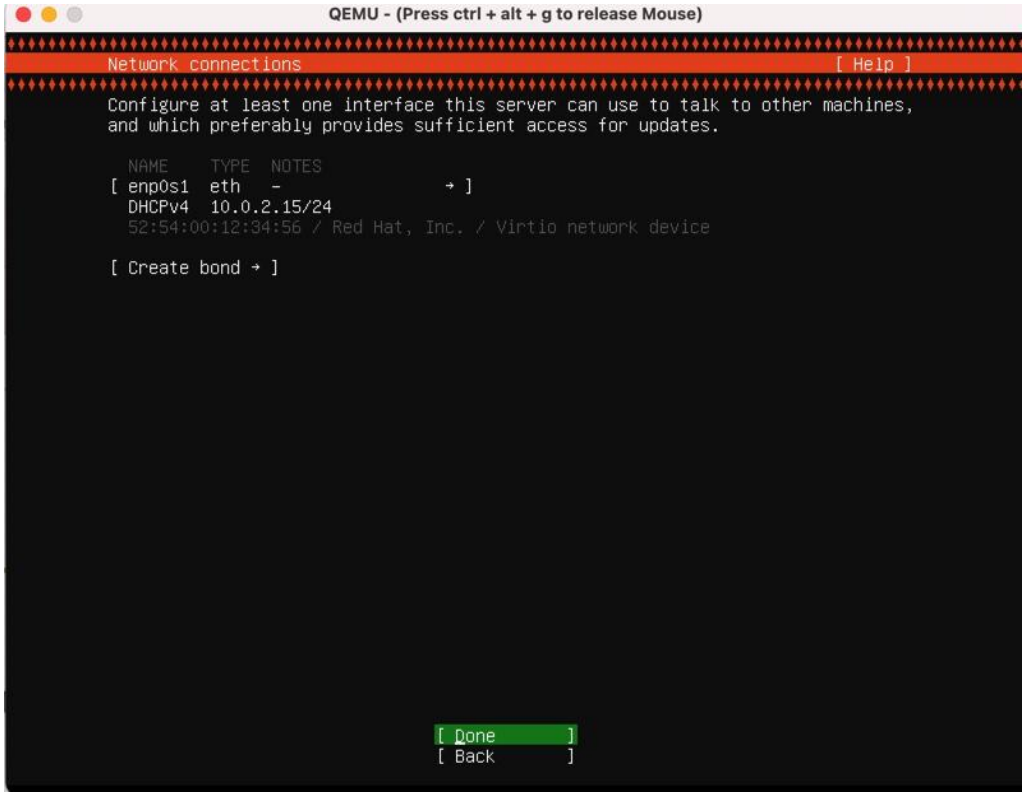
   cd /usr/local/share/qemu
   qemu-img create -f raw ubuntu-latest.raw 40G
   Download pre-built EDK2 UEFI image for QEMU from the below link
   https://gist.github.com/theboreddev/5f79f86a0f163e4a1f9df919da5eea20#:~:text=QEMU_EFI%2Dcb438b9%2Dedk2%2Dstable202011%2Dwith%2Dextra%2Dresolutions.tar.gz
   mv ~/Downloads/QEMU_EFI-*.tar.gz /usr/local/share/qemu
   tar xzvf QEMU_EFI-*.tar.gz
   mv ~/Downloads/ubuntu-20.04.5-live-server-arm64.iso /usr/local/share/qemu

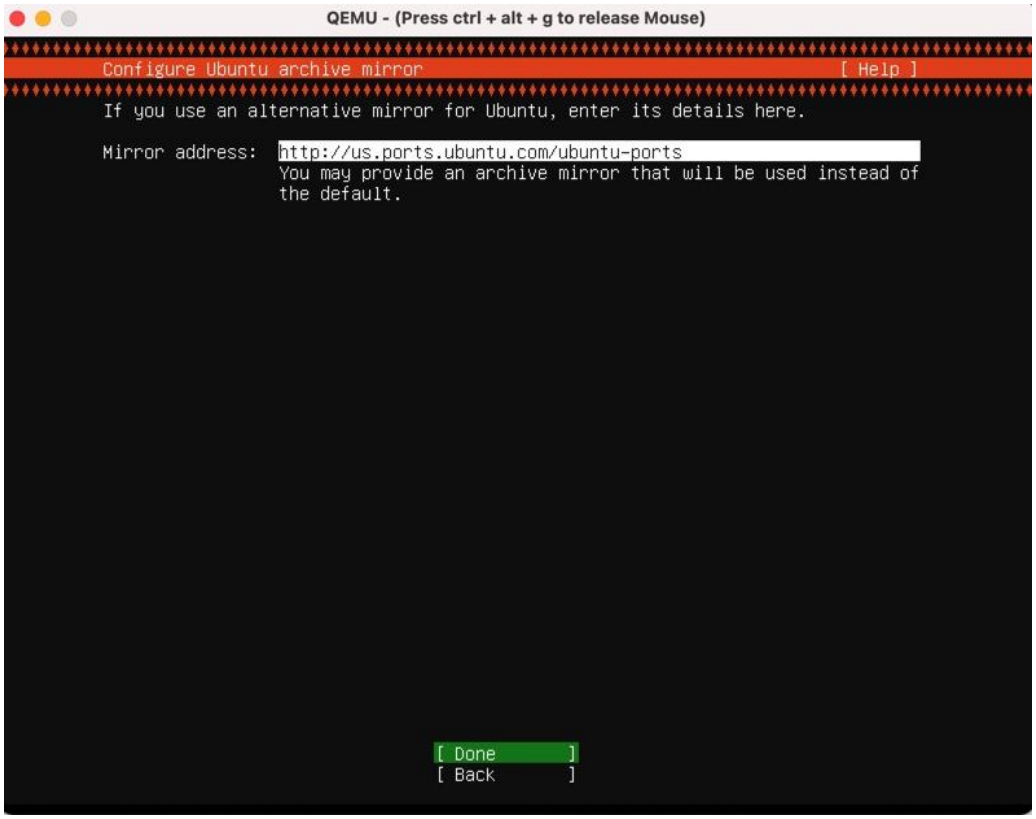
   **qemu-system-aarch64 **

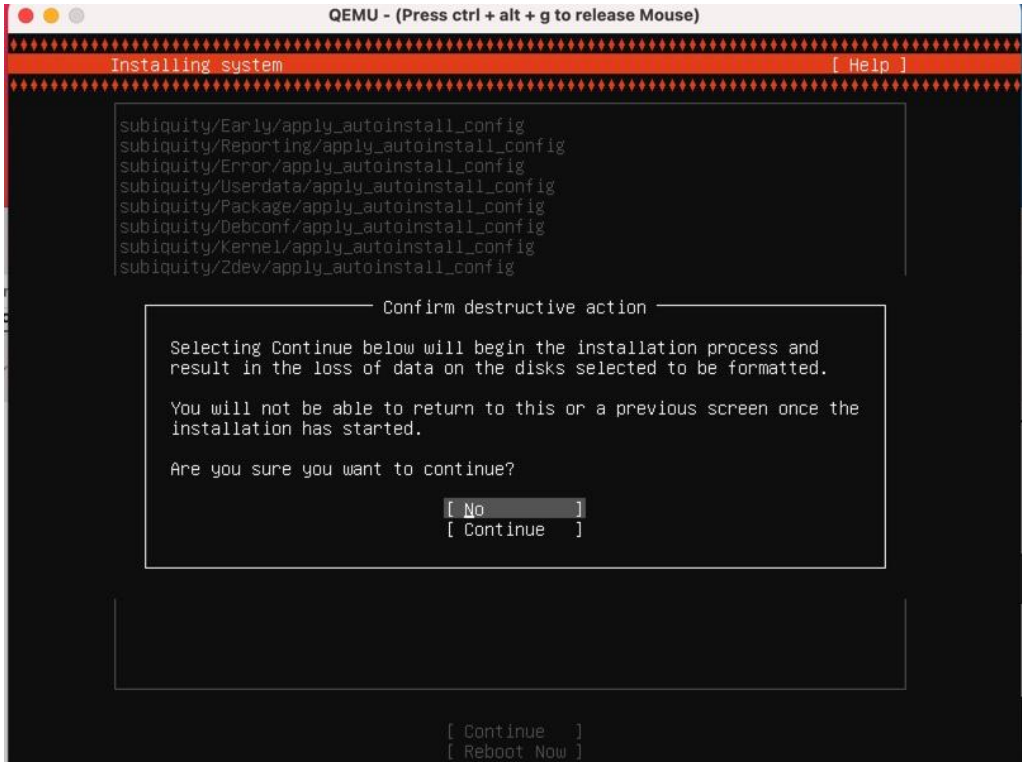
```
  -monitor stdio \
  -M virt,highmem=off \
  -accel tcg \
  -cpu cortex-a57 \
  -smp 4 \
  -m 2048 \
  -bios QEMU_EFI.fd \
  -device virtio-gpu-pci \
  -display default,show-cursor=on \
  -device qemu-xhci \
  -device usb-kbd \
  -device usb-tablet \
  -device intel-hda \
  -device hda-duplex \
  -drive
file=ubuntu-latest.raw,format=raw,if=virtio,cache=writethrough \
  -cdrom ubuntu-20.04.5-live-server-arm64.iso
```

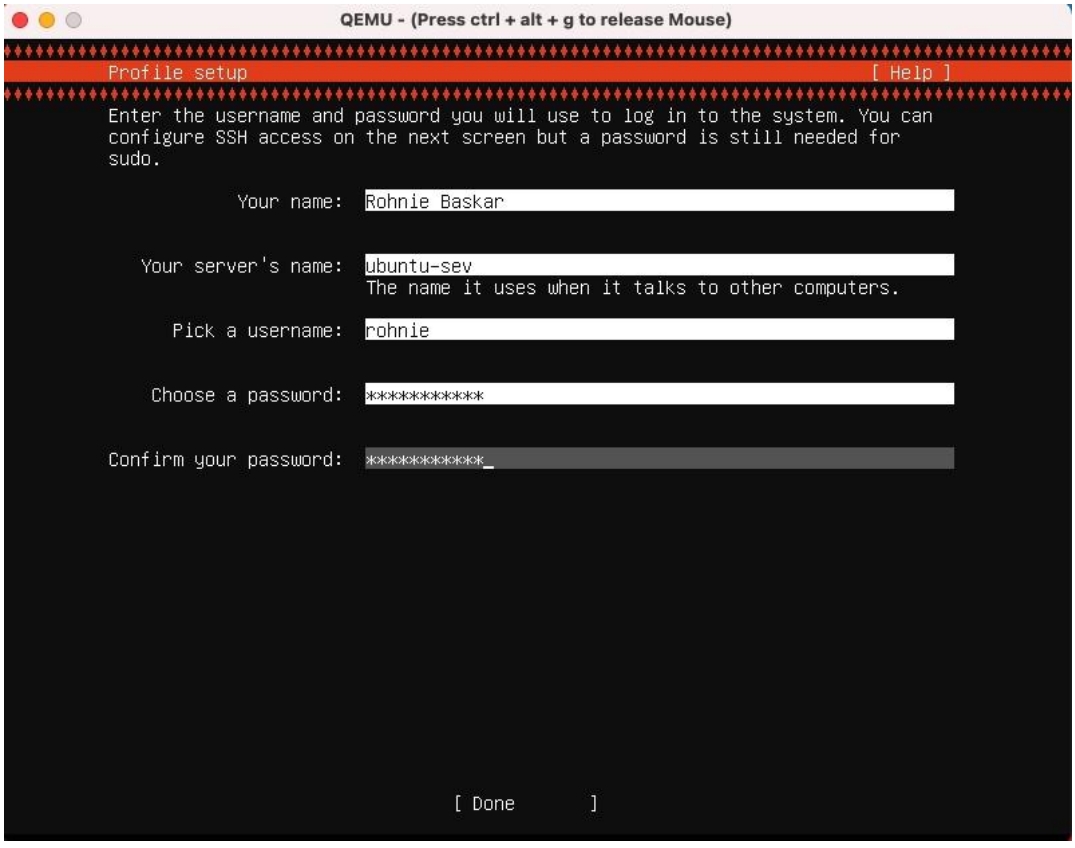
A few screenshots of the installation process is shown below

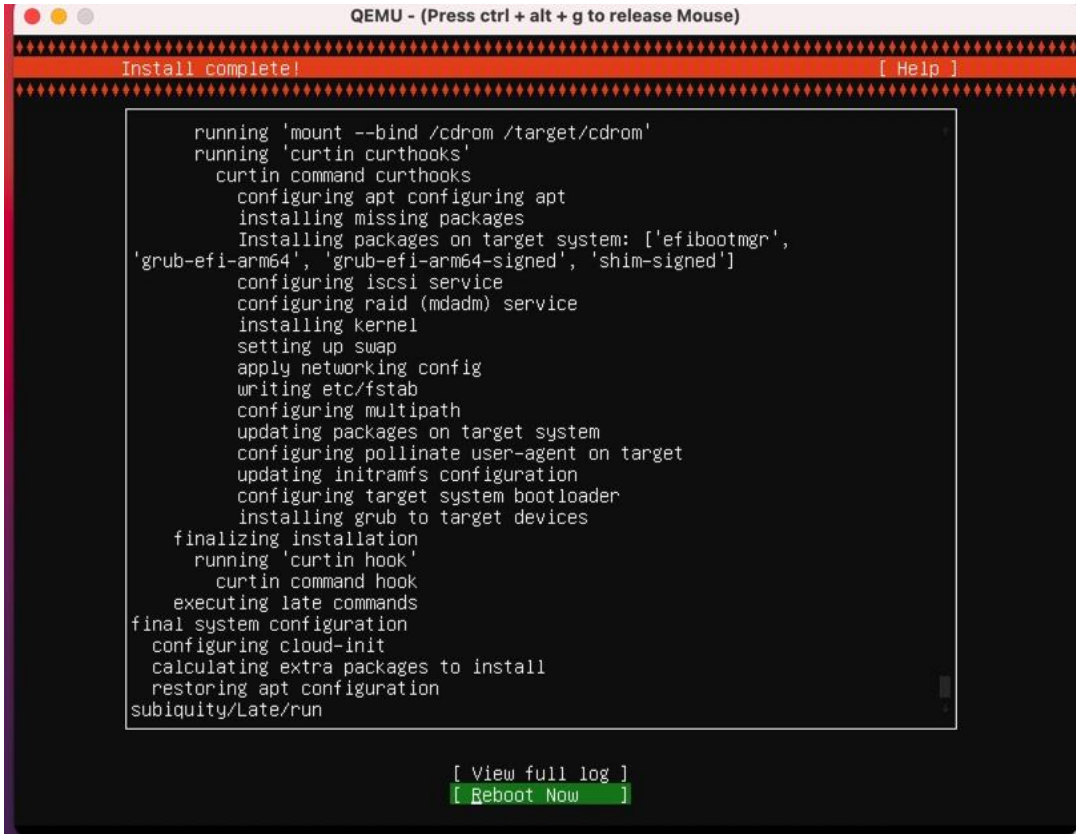



QEMU - (Press ctrl + alt + g to release Mouse)
Network connections
[Help]
Configure at least one interface this server can use to talk to other machines,
and which preferably provides sufficient access for updates.
NAME TYPE NOTES
[enp0s1 eth - →]
DHCPv4 10.0.2.15/24
52:54:00:12:34:56 / Red Hat, Inc. / Virtio network device
[Create bond →]
[Done]
[Back]


QEMU - (Press ctrl + alt + g to release Mouse)
Configure Ubuntu archive mirror
[Help]
If you use an alternative mirror for Ubuntu, enter its details here.
Mirror address: http://us.ports.ubuntu.com/ubuntu-ports
You may provide an archive mirror that will be used instead of
the default.
[Done]
[Back]

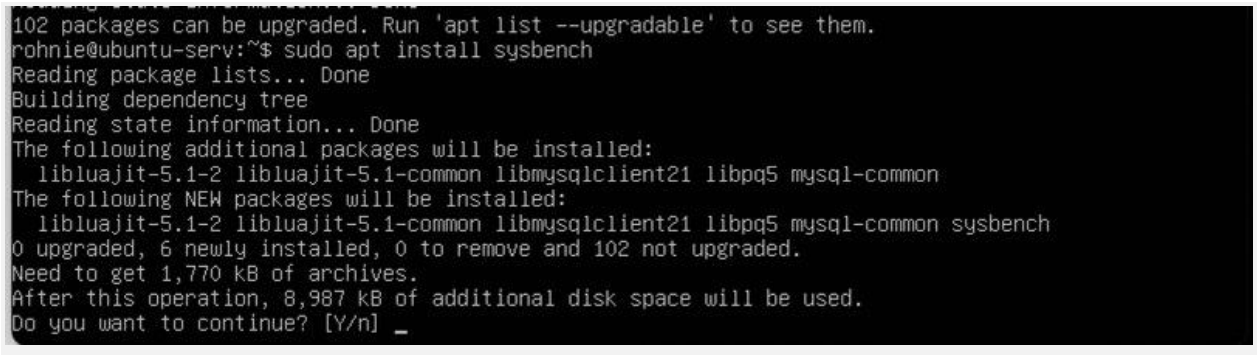

QEMU - (Press ctrl + alt + g to release Mouse)
Installing system
[Help]
subiquity/Early/apply_autoinstall_config
subiquity/Reporting/apply_autoinstall_config
subiquity/Error/apply_autoinstall_config
subiquity/Userdata/apply_autoinstall_config
subiquity/Package/apply_autoinstall_config
subiquity/Debconf/apply_autoinstall_config
subiquity/Kernel/apply_autoinstall_config
subiquity/Zdev/apply_autoinstall_config
Confirm destructive action
Selecting Continue below will begin the installation process and result in the loss of data on the disks selected to be formatted.
You will not be able to return to this or a previous screen once the installation has started.
Are you sure you want to continue?
[No]
[Continue]
[Continue]
[Reboot Now]


QEMU - (Press ctrl + alt + g to release Mouse)
Profile setup
[Help]
Enter the username and password you will use to log in to the system. You can configure SSH access on the next screen but a password is still needed for sudo.
Your name: Rohnie Baskar
Your server's name: ubuntu-sev
The name it uses when it talks to other computers.
Pick a username: rohnie
Choose a password: **********
Confirm your password: **********
[Done]


QEMU - (Press ctrl + alt + g to release Mouse)
Install complete!
[Help]
running 'mount --bind /cdrom /target/cdrom'
running 'curtin curthooks'
curtin command curthooks
configuring apt configuring apt
installing missing packages
Installing packages on target system: ['efibootmgr',
'grub-efi-arm64', 'grub-efi-arm64-signed', 'shim-signed']
configuring iscsi service
configuring raid (mdadm) service
installing kernel
setting up swap
apply networking config
writing etc/fstab
configuring multipath
updating packages on target system
configuring pollinate user-agent on target
updating initramfs configuration
configuring target system bootloader
installing grub to target devices
finalizing installation
running 'curtin hook'
curtin command hook
executing late commands
final system configuration
configuring cloud-init
calculating extra packages to install
restoring apt configuration
subiquity/Late/run
[View full log]
[Reboot Now]


QEMU - (Press ctrl + alt + g to release Mouse)
Ubuntu 20.04.5 LTS ubuntu-serv tty1
ubuntu-serv login: Rohnie Baskar
Password:

4. Sysbench installation on Ubuntu is very simple, It can be done with the following commands:
   $ sudo apt update
   $ sudo apt install sysbench



Docker

Docker is a set of platforms that enable OS-level virtualization to deliver software in packages called containers. Docker is the most popular container management platform.

SETUP - INSTALLATION

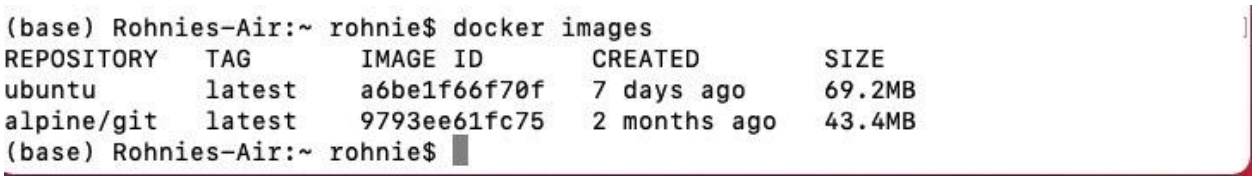
The below were the steps followed to install Docker and have Ubuntu image running on top of it.

1. Downloaded docker desktop executable file from ARM (Apple Silicon):: Install on Mac | Docker Documentation and followed the wizard for completion.

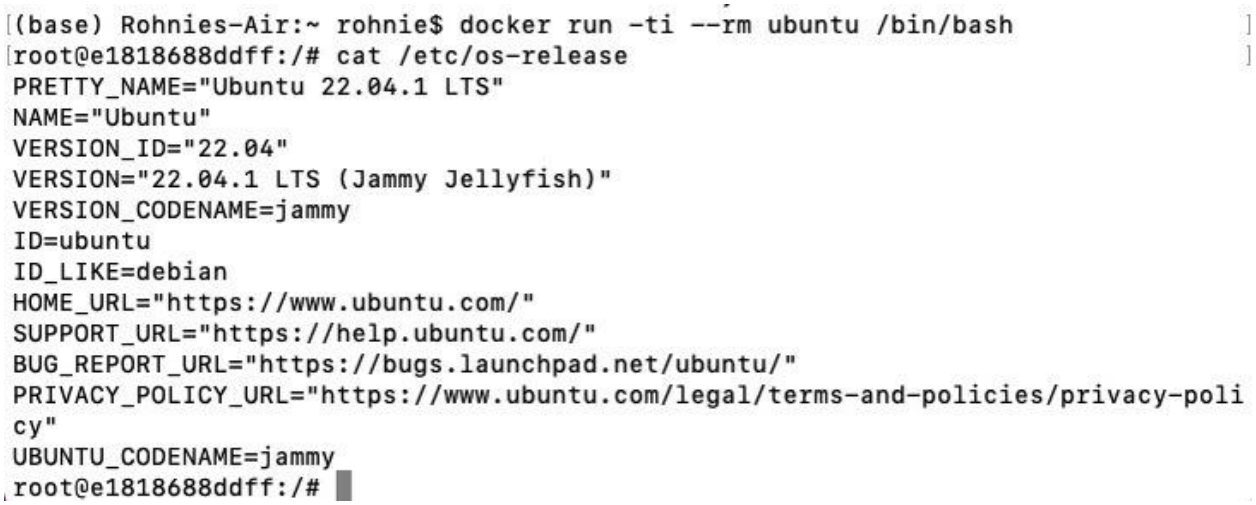
2. Fetch the image ubuntu from docker hub repository and store it in the system using the below command

   docker pull ubuntu

3. Check images to reconfirm



4. Docker run command with -ti flag lets us get an interactive terminal in the container. This will be used later with sysbench. The /bin/bash argument is a way of telling the container to run the bash shell terminal. Finally, the --rm flag instructs Docker to automatically remove the container after we stop it.



5. Docker ps will list all the running containers

```
(base) Rohnies-Air:~ rohnie$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS
NAMES
16f0aa0e71f8   ubuntu    "/bin/bash"   7 seconds ago   Up 6 seconds
confident_proskuriakova
(base) Rohnies-Air:~ rohnie$
```

6. Now, to install sysbench, the following commands are used as shown.

```
(base) Rohnies-Air:~ rohnie$ docker run -ti --rm ubuntu /bin/bash
root@16f0aa0e71f8:/# apt update
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease [270 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates InRelease [114 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports jammy-backports InRelease [107 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports jammy-security InRelease [110 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports jammy/universe arm64 Packages [17.2 MB]
Get:6 http://ports.ubuntu.com/ubuntu-ports jammy/restricted arm64 Packages [24.2 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports jammy/multiverse arm64 Packages [224 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 Packages [1758 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 Packages [976 kB]
Get:10 http://ports.ubuntu.com/ubuntu-ports jammy-updates/multiverse arm64 Packages [3452 B]
Get:11 http://ports.ubuntu.com/ubuntu-ports jammy-updates/restricted arm64 Packages [305 kB]
Get:12 http://ports.ubuntu.com/ubuntu-ports jammy-updates/universe arm64 Packages [867 kB]
Get:13 http://ports.ubuntu.com/ubuntu-ports jammy-backports/universe arm64 Packages [22.4 kB]
Get:14 http://ports.ubuntu.com/ubuntu-ports jammy-backports/main arm64 Packages [49.0 kB]
Get:15 http://ports.ubuntu.com/ubuntu-ports jammy-security/restricted arm64 Packages [292 kB]
Get:16 http://ports.ubuntu.com/ubuntu-ports jammy-security/main arm64 Packages [661 kB]
Get:17 http://ports.ubuntu.com/ubuntu-ports jammy-security/universe arm64 Packages [663 kB]
Fetched 23.7 MB in 4s (5991 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@16f0aa0e71f8:/#
```

```
root@16f0aa0e71f8:/# apt install sysbench
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libaio1 libldap-2.5-0 libldap-common libluajit-5.1-2 libluajit-5.1-common libmysqlclient21 libpq5
  libsasl2-2 libsasl2-modules libsasl2-modules-db mysql-common
Suggested packages:
  libsasl2-modules-gssapi-mit | libsasl2-modules-gssapi-heimdal libsasl2-modules-ldap libsasl2-modules-otp
  libsasl2-modules-sql
The following NEW packages will be installed:
  libaio1 libldap-2.5-0 libldap-common libluajit-5.1-2 libluajit-5.1-common libmysqlclient21 libpq5
  libsasl2-2 libsasl2-modules libsasl2-modules-db mysql-common sysbench
0 upgraded, 12 newly installed, 0 to remove and 0 not upgraded.
Need to get 2175 kB of archives.
After this operation, 9582 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 libaio1 arm64 0.3.112-13build1 [7018 B]
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 libsasl2-modules-db arm64 2.1.27+dfsg2-3ub
untu1.1 [21.2 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 libsasl2-2 arm64 2.1.27+dfsg2-3ubuntu1.1 [
55.6 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 libldap-2.5-0 arm64 2.5.13+dfsg-0ubuntu0.2
2.04.1 [181 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports jammy-updates/main arm64 libldap-common all 2.5.13+dfsg-0ubuntu0.22
.04.1 [15.9 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports jammy/universe arm64 libluajit-5.1-common all 2.1.0~beta3+dfsg-6 [4
4.3 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports jammy/universe arm64 libluajit-5.1-2 arm64 2.1.0~beta3+dfsg-6 [221
kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports jammy/main arm64 mysql-common all 5.8+1.0.8 [7212 B]
```

7. When we stop the container at this stage we lose all the changes made including any installations done. For this we commit this container's changes to a new docker image. We can then use the new image as our base image with sysbench.
8. We can finally see the image history of the new image 'myubuntu'. The steps are shown in this screenshot.

```
(base) Rohnies-Air:~ rohnie$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
16f0aa0e71f8   ubuntu    "/bin/bash"   7 seconds ago   Up 6 seconds             confident_proskuriakova
(base) Rohnies-Air:~ rohnie$ docker commit -p 16f0aa0e71f8 myubuntu
sha256:461ab448195179a6440882156cb2351191242e4050a496d41886698c499b28dc
(base) Rohnies-Air:~ rohnie$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
myubuntu      latest    461ab4481951   5 seconds ago   117MB
ubuntu        latest    a6be1f66f70f   7 days ago      69.2MB
alpine/git    latest    9793ee61fc75   2 months ago    43.4MB
(base) Rohnies-Air:~ rohnie$ docker image history myubuntu
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
461ab4481951   42 seconds ago   /bin/bash                                       48MB
a6be1f66f70f   7 days ago       /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      7 days ago       /bin/sh -c #(nop) ADD file:55486a91f732042dd…   69.2MB
<missing>      7 days ago       /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      7 days ago       /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      7 days ago       /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      7 days ago       /bin/sh -c #(nop)  ARG RELEASE                  0B
(base) Rohnies-Air:~ rohnie$
```

Sysbench - Docker

CPU Tests:

Have run max prime cpu test for 3 different test scenarios. Each has run 5 times. Automated scripts available in github repo
Script: cpuTest_Docker.sh

The consolidated metrics of the different test runs are below

Test 1: Docker, max prime cpu=20000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	3495.79	0.28	0.29	1.17	0.30	0
2	3551.02	0.27	0.28	0.66	0.30	0
3	3559.84	0.28	0.28	1.02	0.29	0
4	3558.03	0.28	0.28	0.82	0.29	0
5	3559.99	0.28	0.28	0.76	0.29	0

Test 2: Docker, max prime cpu=50000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	1105.59	0.89	0.90	1.55	0.92	0
2	1100.65	0.89	0.91	1.71	0.94	0
3	1096.64	0.89	0.91	4.18	0.95	0
4	1102.17	0.89	0.91	1.29	0.94	0
5	1100.09	0.89	0.91	1.78	0.95	0

Test 3: Docker, max prime cpu=90000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	514.38	1.91	1.94	2.75	2.00	0
2	515.13	1.91	1.94	2.53	2.00	0
3	513.27	1.90	1.95	2.83	2.00	0
4	512.68	1.91	1.95	3.04	2.03	0
5	515.23	1.91	1.94	2.60	2.00	0

FileIO Tests:

The fileio tests were performed for three different modes with 16 threads and 1GB file size. Automated scripts available in github repo
Script: fileioTest_Docker.sh

The consolidated metrics of the different test runs are below

Test 1: Docker, Mode=rndwr

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	73.15	4566	0.01	0.15	0.76	91.92
2	56.35	3678	0.01	0.16	22.69	71.70
3	44.85	2995	0.01	0.13	7.55	80.01
4	40.83	2767	0.01	0.18	26.02	104.92
5	65.83	4236	0.01	0.18	8.94	91.77

Test 1: Docker, Mode=seqrd

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	1245	76437	0.01	0.15	143	451
2	1125	75667	0.01	0.16	125	645
3	989	76437	0.01	0.15	311	342
4	1268	74871	0.01	0.18	145	621
5	1139	73281	0.01	0.186	145	451

Test 1: Docker, Mode=seqrewr

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	200.1	12845.53	0.03	0.78	234	466
2	201	12945	0.03	0.71	671	475
3	151	9663	0.04	1.35	323	367
4	215	13462	0.03	0.71	411	612
5	186	11946	0.03	0.56	225	492

Sysbench - QEMU

CPU Tests:

Have run max prime cpu test for 3 different test scenarios. Each has run 5 times. Automated scripts available in github repo
Script: benchmarkcpu.sh

The consolidated metrics of the different test runs are below

Test 1: QEMU, max prime cpu=20000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	96.12	9.83	10.32	12.52	10.84	0
2	92.59	9.79	10.73	23.06	12.52	0
3	90.77	9.97	10.95	21.05	12.75	0

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
4	91.85	9.96	10.82	14.93	12.30	0
5	94.10	9.95	10.56	13.32	11.24	0

Test 2: QEMU, max prime cpu=50000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	31.63	29.74	31.46	38	33.12	0
2	31.38	30.01	31.71	37.67	3312	0
3	31.32	30.67	31.77	35.55	33.12	0
4	30.92	30.12	32.17	46.31	36.89	0
5	31.16	30.57	31.93	43.29	33.72	0

Test 3: QEMU, max prime cpu=90000

S.No	Events/Sec	Min	Avg	Max	95th Percentile	Std
1	14.19	68.45	70.19	73.35	71.83	0
2	12.40	67.44	78.63	388.22	71.83	0
3	3.04	158.27	323.76	413.79	383.33	0
4	2.95	165.02	328.48	422.61	390.30	0
5	3.49	169.03	279.34	393.67	376.49	0

FileIO Tests:

The fileio tests were performed for three different modes with 16 threads and 1GB file size. Automated scripts available in github repo
Script: benchmarkfileio.sh

The consolidated metrics of the different test runs are below

Test 1: QEMU, Mode=rndwr

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	48.84	3125.93	0.06	2.17	108.21	111.04
2	51.11	3314	0.06	2.14	111.2	110.2
3	46.66	3245	0.06	2.17	108.2	124
4	48.84	3567	0.06	2.17	124	135.4
5	49.92	3222	0.06	2.17	122.1	118.2

Test 1: QEMU, Mode=seqrd

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	366.3	23443	0.13	0.64	59.91	110.99
2	355.6	23467	0.13	0.65	60.45	111.3
3	345.3	23556	0.13	0.64	61.45	123.45
4	364.8	23478	0.12	0.64	57.44	110.45
5	360.4	23563	0.13	0.65	70.34	111.43

Test 1: QEMU, Mode=seqrewr

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
1	36.49	2335.35	0.06	2.96	71.84	252.29

S.No	Transfer Rate, Mb/Sec	Requests/sec	Min ms	Avg ms	Max ms	Std
2	38	2367.45	0.06	2.97	70.45	259.33
3	40.5	2453.55	0.06	2.97	84.33	278.94
4	35.5	2348.33	0.06	2.96	79.79	259.69
5	45.3	2357.97	0.06	2.96	67.33	268.44

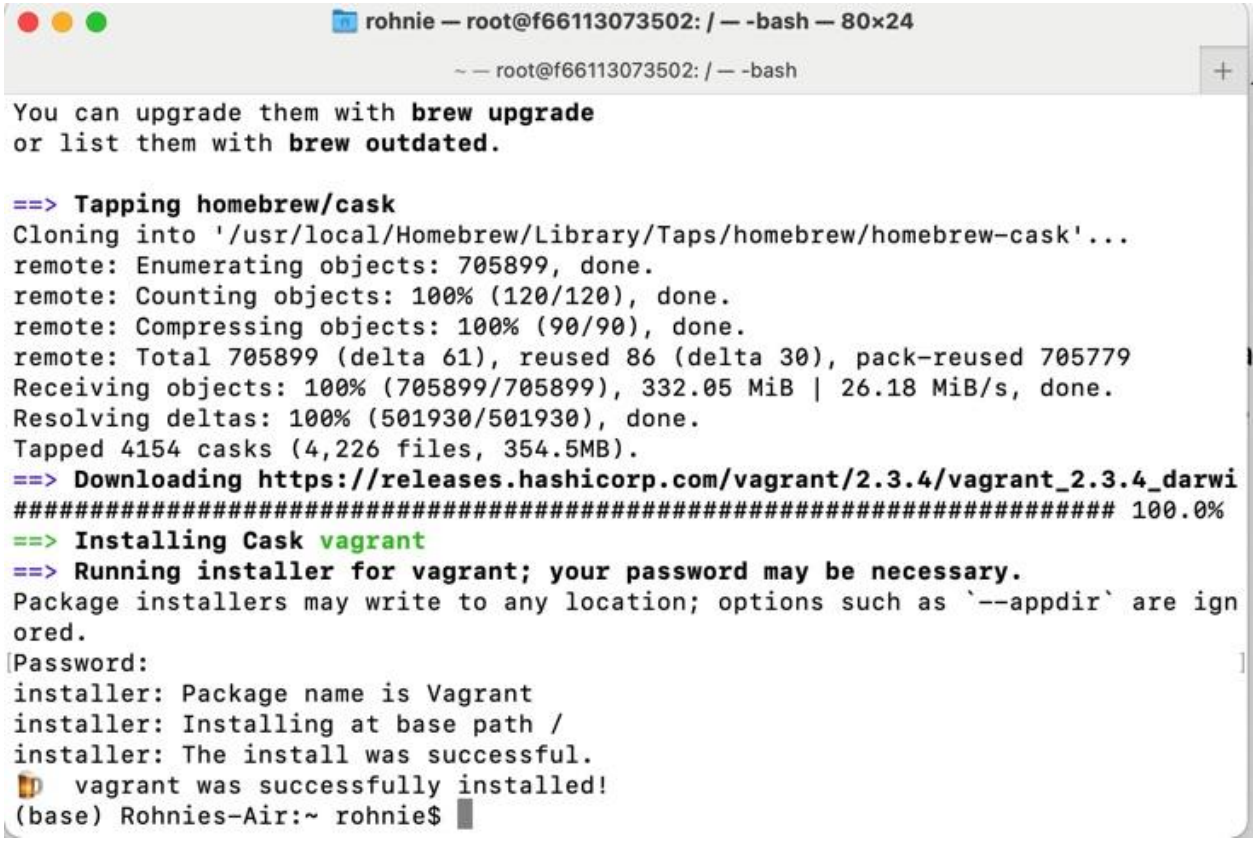
Inference

1. Booting up of QEMU is faster, when memory and cores are increased.
2. The CPU events per second is higher in OS virtualization when compared to system virtualization.
3. As the cpu max prime increases the cpu events decreases for both system and OS virtualizations
4. Despite the decrease in cpu's per vent OS virtualization is much faster than system virtualization
5. The transfer rate and events per second is higher in OS virtualization compared to System Virtualization
6. OS virtualization performs better than system virtualization based on the cpu and fileio benchmark tests.

Vagrant File

Vagrant is an open source tool for creating and configuring VMs. In order to automate the bring up of QEMU and ubuntu as guest os in it, we require virtualization management library libvirt.

1. Install QEMU - Refer QEMU chapter above
2. Install vagrant



3. Install libvirt
4. Find the box compatible with libvrt from

   https://app.vagrantup.com/boxes/search?page=1&provider=libvirt
5. Vagrant file shared in github: VagrantFile

Docker File

1. Placed the docker file in the github repo
2. docker build .

```
(base) Rohnies-Air:Temp rohnie$ docker build -f ~/Documents/Cloud\ Computing/Assignment\ 1/Temp/Dockerfile .
[+] Building 0.1s (7/7) FINISHED
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 166B                                          0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [internal] load metadata for docker.io/library/myubuntu:latest            0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 600B                                             0.0s
 => [1/2] FROM docker.io/library/myubuntu                                     0.0s
 => [2/2] COPY sysbenchTestDocker.sh /sysbenchTest.sh                         0.0s
 => exporting to image                                                        0.0s
 => => exporting layers                                                       0.0s
 => => writing image sha256:a10f65d5c01f6430da1d1f0002538d99db2b122238fc4     0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
(base) Rohnies-Air:Temp rohnie$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    a10f65d5c01f   About a minute ago   117MB
myubuntu     latest    2f396bec83a1   15 minutes ago       117MB
ubuntu       latest    a6be1f66f70f   8 days ago           69.2MB
(base) Rohnies-Air:Temp rohnie$
```