

Amanda Taylor

Sean Choi

October 19, 2021

COEN 241: Cloud Computing

Homework 1

Git Repository Information

<https://github.com/amandarito/coen241-hw1>

Host Environment

I used my old MacBook for this assignment (retina, late 2012 model).

Processor:	2.5 GHz Intel Core i5
Memory:	8 GB 1600 MHz DDR3
Disk Space:	23 GB / 121 GB total (barely enough, I know)
OS:	Mac OS X Mojave

QEMU Setup

I used homebrew to install QEMU.

```
$ brew install qemu
```

Once it's done we can create the image, with 10G of disk space. -f qcow specifies the image format, QEMU copy-on-write v2.

```
$ qemu-img create ubuntu.img 10G -f qcow2
```

Then boot it up with an ubuntu iso for the cd rom, and any other desired specifications.

```
$ qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./Desktop/ubuntu-16.04.7-server-amd64.iso -m 2046 -boot strict=on
```

-hda specifies the virtual hard drive (the image). -boot d says to boot from the virtual cdrom drive first. I use the -m flag to give it some memory, and have the cdrom set to the iso so we can install onto our image. After this step it boots up and shows a QEMU window with ubuntu installer inside, prompting for install. I just followed the instructions to install it.

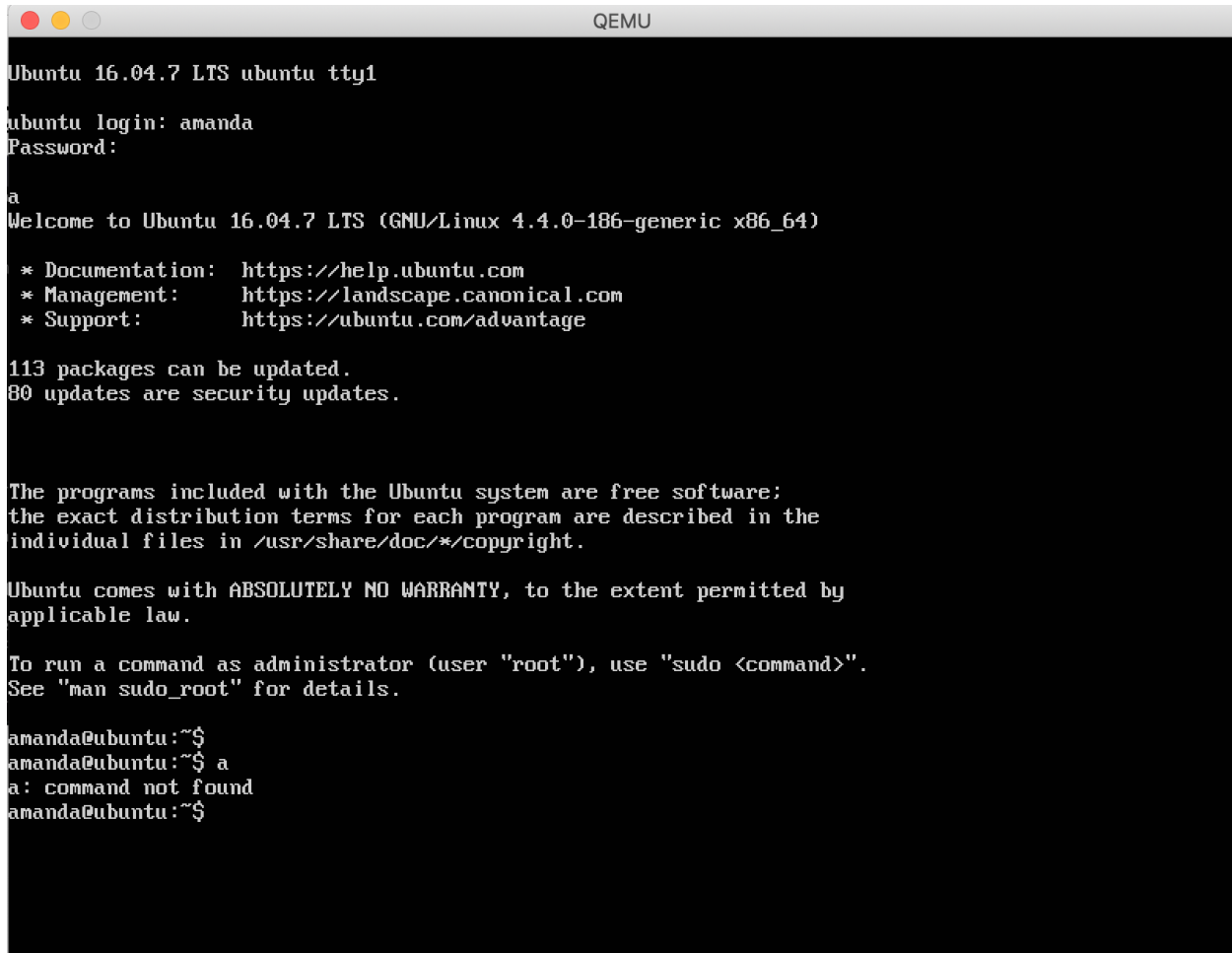
After install is complete, it will tell you. I then closed and booted up the system using this command, which now is booting off of our image.

```
$ qemu-system-x86_64 -hda ubuntu.img -boot d -m 2046 -boot strict=on
```

Then I logged in with username and password I set during installation. For sysbench, I downloaded it via command line:

```
$ sudo apt update
$ sudo apt install sysbench
```

Then it's ready for testing.



```
QEMU
Ubuntu 16.04.7 LTS ubuntu tty1
ubuntu login: amanda
Password:
a
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-186-generic x86_64)

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

113 packages can be updated.
80 updates are security updates.

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.

amanda@ubuntu:~$
amanda@ubuntu:~$ a
a: command not found
amanda@ubuntu:~$
```

Running QEMU setup.

Docker Setup

First I downloaded Docker Desktop from the docker website. After running the app, it installs Docker CLI and engine.

On command line, I created a container with the desired image by using this command

```
docker run -it -m 6m --name=ubuntu1 csmhpp/ubuntu-sysbench
```

giving the container 6mb of memory (the minimum), and naming it ubuntu. I actually wanted to give it less memory to be a better comparison with my QEMU environment, but it wasn't allowed. The image is pulled because I didn't already pull it. I am now interacting with the container and we can do our tests.

Other useful docker commands I ended up using:

```
docker ps
```

to see list of containers

```
docker ps -a
```

to see list of containers, including stopped

```
docker images
```


to see list of images

```
docker container rm
```

to delete containers, use id

```
docker image rm
```

to delete images. if a container is using this image, you would need to delete it first or use the -f flag



```
amandataylor — root@68c4b439e4b8: / — com.docker.cli • docker run -it -m 6m --name=ubuntu1 csmnpp/ubuntu-sysbench — 142x37
root@68c4b439e4b8:/usr/src/hw1# ls
README.md  cpu_prime.sh  docker_cpu_test.txt  docker_fileio_test.txt  file_io.sh  qemu_cpu_test.txt  qemu_fileio_test.txt
root@68c4b439e4b8:/usr/src/hw1# cd ..
root@68c4b439e4b8:/usr/src# cd ..
root@68c4b439e4b8:/usr# cd ..
root@68c4b439e4b8:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  sysbench  tmp  usr  var
root@68c4b439e4b8:/#
```

Running docker environment.

Tests

The shell scripts I used were doing sysbench CPU max prime tests, and file i/o tests with random read/write. I kept things mostly constant but used 3 different number of threads for each test, to see how it would affect performance. For each different configuration, I ran the test 5 times using a loop in the script. I wrote the results into 4 separate files (1 file i/o test & 1 CPU test per environment). My “3 different scenarios” in this case are the number of threads.

CPU max prime

For the script, a blank file is created with the first line. Then 5 times each, sysbench runs the test with a different number of threads (1, 2, 8). For my machine, I found setting -cpu-max-prime to 5000 usually let the machine finish the test, while still taking enough time to have some consistent results.

```
echo -n > cpu_test.txt
for i in 1 2 3 4 5; do sysbench --test=cpu --cpu-max-prime=5000 --num-threads=1 --max-time=45 run >> cpu_test.txt; done
for i in 1 2 3 4 5; do sysbench --test=cpu --cpu-max-prime=5000 --num-threads=2 --max-time=45 run >> cpu_test.txt; done
for i in 1 2 3 4 5; do sysbench --test=cpu --cpu-max-prime=5000 --num-threads=8 --max-time=45 run >> cpu_test.txt; done
```

Here is a table of the average total time, in seconds:

Threads / Environment	QEMU	Docker
1	32.4s	6.76s
2	34.85	3.62s
8	21.79s	3.23s

The docker setup is much faster. It's possible this could be because of the different memory amounts given, but by the time I'm writing this it's a little late to change it. They both have a very small amount. It is a little interesting how the QEMU setup sees much more change between 2 and 8 threads, whereas the Docker setup has a bigger gap between 1 and 2 threads.

File I/O

The script is a lot like the last one but with more lines that have to do with the I/O test. Here it is:

```
echo -n > file_io_test.txt
sysbench --test=fileio --num-threads=1 --file-test-mode=rndrw --file-total-size=1G prepare
for i in 1 2 3 4 5; do sysbench --test=fileio --num-threads=1 --file-test-mode=rndrw --file-total-size=1G run >> file_io_test.txt; done
```

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

```
sysbench --test=fileio --num-threads=2 --file-test-mode=rndrw --file-total-size=1G prepare
for i in 1 2 3 4 5; do sysbench --test=fileio --num-threads=2 --file-test-mode=rndrw --file-total-size=1G run >> file_io_test.txt; done
sysbench --test=fileio --num-threads=2 --file-test-mode=rndrw --file-total-size=1G cleanup
```

```
sysbench --test=fileio --num-threads=8 --file-test-mode=rndrw --file-total-size=1G prepare
for i in 1 2 3 4 5; do sysbench --test=fileio --num-threads=8 --file-test-mode=rndrw --file-total-size=1G run >> file_io_test.txt; done
sysbench --test=fileio --num-threads=8 --file-test-mode=rndrw --file-total-size=1G cleanup
echo finished
```

It's using the random read/write mode. I felt this would be a good indicator of general file I/O.

Here is a table of the average total time, in seconds:

Threads / Environment	QEMU	Docker
1	8.39s	3.88s
2	7.84s	3.50s
8	7.58s	2.08s

In this test, Docker seems to run much faster on average. In both cases, more threads speeds up performance of the task.

Based on these tests, it's clear that the Docker environment is faster (at least on my machine with these configurations). I would have tried giving more memory and disk space but my host machine is running out, so this is what I can do for now.

Conclusion

Overall, Docker seems to give better performance by miles. This could be due to slightly different memory given to the Docker container and the QEMU image. In the future, I'd like to check to see if that's the case. In terms of what was easier to set up, I think Docker was a lot easier. It's also nice that there are a lot of images to pull from on Docker Hub. This assignment was helpful in learning a lot of things, such as writing shell scripts, using git, and understanding sysbench tests (or just computer tests in general).