Ryan Skipp, Alex Garibaldi, Alex Kuhn, and David Katz
Dr. Ngo
CSC496 Spring 2020
1 May 2020

Course Project: Deliverable #2 Technical Paper

In this project, what we are doing is setting up an Alpine Linux image, installing Docker onto that image, installing our testing programs through Docker, and then using them through Docker to analyze performance in various benchmarks such as disk read/write speed, network speed and bandwidth, and more. We have documented the steps below our group has utilized to do this:

Step 1: Preparing the Alpine Linux Image.

To prepare the image, we follow Ngo's slides he has posted, only increasing our disk size to 20GB and our RAM size to 8GB.

Step 2: Update and Prepare Alpine Linux

This section is also largely covered by Ngo's slides, up to and including creating a non-root user (in our case, the "student" user) to safely run the benchmarks, as leaving everything running under root is a recipe for disaster once this image is running on CloudLab. Afterwards, the following steps are performed with root privileges:

`apk update` (updates list of programs from repos)
`apk upgrade` (upgrades the installed versions of these programs)
`apk add nano` (installs nano text editor, our personal preference for command line text editing)

Step 3: Installing Docker in Alpine

For our benchmarks, it is necessary to install Docker. To do this, we have to allow access to the community repository of Alpine, and then install the Docker program and ensure its services are running. The steps are as follows, and should be performed with root privileges:

`nano /etc/apk/repositories` (opens up file to edit in nano)
Add the line to the file for the repository we need to add: `http://dl-cdn.alpinelinux.org/alpine/latest-stable/community`
`apk update` (updates list of programs from repos, as we just added a repo)
`apk upgrade` (upgrades your installed versions of these programs)
`apk add docker` (installs docker)
`service docker start` (starts docker run-time service)
`rc-update add docker boot` (starts docker service at boot from now on so it no longer manually needs to be started)
`addgroup student docker` (adds your non-root user to docker group, so you can use docker as non-root)

`docker run -it ubuntu` (starts up ubuntu in Docker)
`apt-get update` (update list of programs from repo, this time in Ubuntu however)
`apt-get install fio` (installs fio testing benchmark)
`apt-get install nuttcp` (installs netccp testing benchmark)

Then, everything else should work as a non-root user.

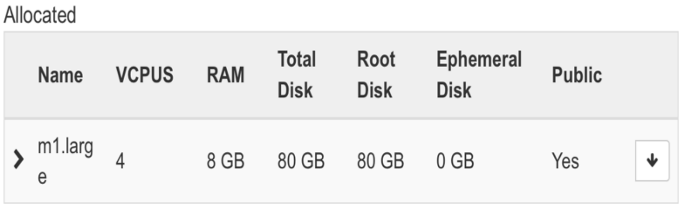
`exit` (log out of ubuntu/Docker)
`docker commit (ID) ubuntu-csc496` (this saves ubuntu image + testing programs)

Step 4: Now, booting up Alpine from scratch, you should be able to:

Login as a non root user, and then execute:
`docker run -it ubuntu-csc496` to start up Docker image
Now, Docker is running and all the benchmarks that we installed are available and ready to be used!

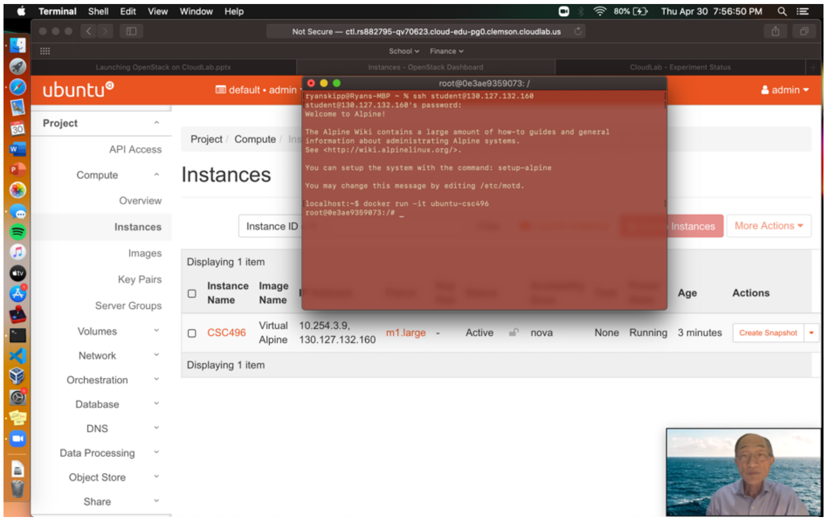
Step 5: Uploading to CloudLab and starting the server

The process for uploading and launching our Alpine server is very similar to what is shown on Ngo's slides. However, we chose the following specs for our server, which we believe to be similar to an "average" web server:



Allocated

Name	VCPUS	RAM	Total Disk	Root Disk	Ephemeral Disk	Public
m1.large	4	8 GB	80 GB	80 GB	0 GB	Yes



```
root@0e3ae9359073: /
ryanskipp@Ryans-MBP ~ % ssh student@130.127.132.160
student@130.127.132.160's password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~$ docker run -it ubuntu-csc496
root@0e3ae9359073:/# _
```

Displaying 1 item

Instance Name	Image Name									Age	Actions
CSC496	Virtual Alpine	10.254.3.9, 130.127.132.160	m1.large	-	Active		nova	None	Running	3 minutes	Create Snapshot

Displaying 1 item

Consequently, the local machine used for benchmark testing is Ryan's MacBook Pro. His laptop is running the latest macOS 10.15 and the benchmarks run natively. The specs for his machine are below, and can be considered a typical laptop:

CPU	RAM	Disk Space
2.6 GHz Dual Core Intel i5	8GB 1600MHz DDR3	256GB Flash Storage

Step 6: Testing using fio

This program is used to test disk writing and reading speed. To document our testing results, the plan is to compare an average of our local, native machine disk read/write speeds compared to the benchmarks running in Docker in order to get an idea of the difference in performance that the default Docker environment gives us:

Test #1: A measure of random write speeds. This command will write a total of 2GB of files [4 jobs x 512 MB = 2GB] through running 4 processes at a time:

```
fio --name=randwrite --ioengine=libaio --iodepth=1 --rw=randwrite
--bs=4k --direct=0 --size=512M --numjobs=4 --runtime=60 --
group_reporting
```

Three results from our CloudLab Server:

```
WRITE: bw=96.1MiB/s (101MB/s), 96.1MiB/s-96.1MiB/s (101MB/s-
101MB/s), io=2048MiB (2147MB), run=21319-21319msec
WRITE: bw=87.5MiB/s (91.8MB/s), 87.5MiB/s-87.5MiB/s (91.8MB/s-
91.8MB/s), io=2048MiB (2147MB), run=23403-23403msec
WRITE: bw=85.4MiB/s (89.5MB/s), 85.4MiB/s-85.4MiB/s (89.5MB/s-
89.5MB/s), io=2048MiB (2147MB), run=23983-23983msec
```

Three results from the personal laptop:

```
WRITE: bw=356MiB/s (374MB/s), 356MiB/s-356MiB/s (374MB/s-
374MB/s), io=2048MiB (2147MB), run=5747-5747msec
WRITE: bw=435MiB/s (456MB/s), 435MiB/s-435MiB/s (456MB/s-
456MB/s), io=2048MiB (2147MB), run=4711-4711msec
WRITE: bw=377MiB/s (396MB/s), 377MiB/s-377MiB/s (396MB/s-
396MB/s), io=2048MiB (2147MB), run=5426-5426msec
```

Test #2: A measure of random read speeds. This command will read a total of 2GB of files through running 4 processes at a time:

```
fio --name=randread --ioengine=libaio --iodepth=16 --rw=randread
--bs=4k --direct=0 --size=512M --numjobs=4 --runtime=240 --
group_reporting
```

Three results from our CloudLab Server:

```
READ: bw=1154KiB/s (1182kB/s), 1154KiB/s-1154KiB/s (1182kB/s-
1182kB/s), io=271MiB (284MB), run=240024-240024msec
READ: bw=1429KiB/s (1464kB/s), 1429KiB/s-1429KiB/s (1464kB/s-
1464kB/s), io=335MiB (351MB), run=240015-240015msec
READ: bw=1424KiB/s (1458kB/s), 1424KiB/s-1424KiB/s (1458kB/s-
1458kB/s), io=334MiB (350MB), run=240014-240014msec
```

Three results from the personal laptop:

```
READ: bw=79.6MiB/s (83.5MB/s), 79.6MiB/s-79.6MiB/s (83.5MB/s-
83.5MB/s), io=2048MiB (2147MB), run=25725-25725msec
READ: bw=174MiB/s (182MB/s), 174MiB/s-174MiB/s (182MB/s-182MB/s),
io=2048MiB (2147MB), run=11798-11798msec
READ: bw=181MiB/s (190MB/s), 181MiB/s-181MiB/s (190MB/s-190MB/s),
io=2048MiB (2147MB), run=11287-11287msec
```

As can be seen through the data, there is a clear indication that the server is running at significantly slower disk/read write speeds. This is, however, not a reflection on CloudLab. *Most likely this is due to* Docker placing default restrictions on the amount of resources allocated to it compared to the machine running it natively without such restrictions. For future tests, we will try to remove these Docker restrictions.

Step 6: Testing using nuttcp

This program can be used to test network bandwidth. To document our testing results, the plan is to compare the results of our four members connecting to the server (hosting nuttcp) and documenting the bandwidth, determining the average bandwidth that we are getting out of the server. As of right now, we are restricted by port forwarding issues we encountered when dealing between OpenStack and Docker. We hope to resolve this in time for Deliverable 3, and we will document the exact commands and results then.

Step 7: Establish additional benchmarks

Additionally, we are aiming to install the netperf and stream benchmarks for the third deliverable. A similar manner compared to Steps 5 and 6 will be used to test and document the results compared between all four of our machines versus the server. As of right now, we are restricted by port forwarding issues we encountered when dealing between OpenStack and Docker. We hope to resolve this in time for Deliverable 3, and we will document the exact commands and results then.

Thoughts and Conclusions:

I do believe our project has met the second deliverable requirements. We have demonstrated the process for both installing, configuring, and using our benchmark testing programs. Additionally, we have begun to use and experiment with them, demonstrating results from our first benchmark (fio) and ultimately (depending on if OpenStack and Docker want to cooperate) we will be able to document our findings for those benchmarks as well.