Project 1 – Containers and VMs

(Due by Midnight, Oct. 8, 15% Grade)

1 Summary

In this project, you will get familiar with two virtualization techniques: one is Docker (a light-weight, container-based virtualization technique) and the other is QEMU (a user-level full virtualization technique). **Task (1)**: You will need to correctly deploy them, be familiar with the operations, measure the performance using benchmarks, and finally summarize the results in a report; **Task (2)** You will implement a Docker-like tool, which can create container instances with configurable namespaces and control groups.

2 Environment Setup

For the task (1), please follow this document [1] to start a Google cloud instance, but do not use the vCPU, memory and VM image configurations in the document. We will configure this instance using **2 vCPU, 4 GB memory, 30 GB disk size, and Ubuntu 18.04 LTS** image for this assignment. We refer to this instance as **the native system**,

3 Docker

3.1 Installation & Preparation

Please refer to this instruction [2] for Docker installation. In addition, you may want to get familiar with Docker basic operations by referring to [3].

3.2 Docker Measurement

You will measure the performance of Docker containers using the Sysbench benchmark. Please use the following container image with Sysbench pre-installed: **csminpp/ubuntu-sysbench**. Sysbench is a benchmark tool [4]. With it, we can have some basic ideas about measuring system performance by running such benchmark tools against the system under test. In this project, we will only focus on the **cpu and fileio** test modes (section 4.1 and 4.5 in [4]) of Sysbench. The following instructions present some ideas about how to conduct a meaningful performance measurement. Please read them carefully, and then produce measurement results (you will lose points, if the results are incorrectly generated and/or can not be well justified).

- You need to be familiar with the test cases (under the cpu and fileio test modes). You *only* need to pick up **one** test case for each test mode (cpu or fileio).
- For each test case, you need to figure out the“right” test parameters. For example, you need to select the parameter for “–cpu-max-prime” to ensure your test case won’t end in a short time period (too small) or will never end (too large). As another example, you need to decide the file size (e.g., –file-total-size) for Sysbench fileio test mode. Usually, it’s reasonable to make a test case lasting for at least 30 - 60 seconds. Please justify your configurations.
- You may want to repeat/re-run at least 3 times for each test case, and report the average value of your results (the Sysbench will report some user-level performance data, e.g., total time).

- To reduce test variation, you need to test each case under similar test environment. For example, for the fileio test, after one test, the operating system will cache the accessed files. If we don't manually drop such cache, the following tests will finish much faster, as most I/O data will be served directly form the memory cache instead of disks. You have to manually drop such cache, for example, using the command: $echo\ 3 > /proc/sys/vm/drop\_caches$ in the native (not the container). (Notice that, you can only run this command under the root user. Please figure out how to enable root user under Ubuntu and switch to the root user to drop the cache before each fileio test).

- In addition to the user-level performance data generated by the benchmark, you need to collect system performance data (from the native system). For example, you can choose to use iostat (a Linux tool for collecting cpu, I/O, network usage [5]). Please do this on your **native** system (outside containers) – think about why. **You may need to collect performance data covering the whole time period while your benchmark is running**. You must provide the cpu and disk I/O performance data (think about how to show/visualize this data — suppose you will have a series of performance data points). Please report the commands your use and the interpretation of your performance related data.

- Please well organize your experimental setup, configurations, and data (using figures and/or tables); and report them in your final report.

4 QEMU

4.1 Installation & Preparation

Under the native system, you can install qemu (under Ubuntu) simply by:

```
$sudo apt-get update
$sudo apt-get install qemu
```

You will download the Ubuntu iso image to install your QEMU VM:

```
$wget http://mirror.pnl.gov/releases/18.04/ubuntu-18.04.5-live-server-amd64.iso
```

You need to create an image before installing your QEMU VM. For your convenience, we provide the commands below. Please refer to the QEMU manual for more details [6].

```
$sudo qemu-img create ubuntu.img 10G
```

4.2 Install a QEMU VM

Because installing a VM require a graphical user interface (GUI). Please refer to Section 7 of this document to enable the GUI of your Google instance (i.e., the native). You will do the following operations within GUI (you should use a VNC client). Please install the QEMU VM using the command below (which takes the iso file as a "cdrom" and the qemu image as a "hard disk"):

```
$sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-16.04.6-server-amd64.iso -m 1536
```

Then, install the Ubuntu Linux system step by step. It may take **some time** – hours – to complete the whole installation. Please be patient.

After you install the OS to the image, you simple quit the above command. Next time when you want to start the QEMU VM, you should use the following command (omitting the "cdrom"):

$sudo qemu-system-x86_64 -hda ubuntu.img -m 1536

4.3 QEMU VM Measurement

Please report how long it takes to boot a QEMU based VM (assume you have installed the VM already). Please **answer** why it is so slow to run such a VM under QEMU in your report.

5 A mini-Dockedr container

In task (2), you will develop a Docker similar tool (in Python), which can start a container instance with configuration parameters (e.g., namespace and cgroups). We call it **miniDocker**. We suggest you to use the Watson College VM for this task (as you will need some time to write the program, and this VM is free).

1. Program Template: We provide a program template that you could start with (however, you are free to write your own program without following the template):
`http://www.cs.binghamton.edu/~huilu/slidesfall2020/miniDocker_template.py`

This template includes a simple parser, which takes configuration parameters to start a container including (1) the path to the new root file system; (2) hostname of the container; (3) ip address of the container's network interface; (4) memory size and cpu number that the container instance can use. Note that, we have set the default values for these parameters, however you can also pass new values to each of them.

2. Root Filesystem: We provide a root file system that you can download from: `http://www.cs.binghamton.edu/~huilu/slidesfall2020/ubuntu-rootfs.tar.gz`

Place "ubuntu-rootfs.tar.gz" under your working directory (where you run your miniDocker) and unzip it:
$mkdir new_root
$tar xvf ubuntu-rootfs.tar.gz -C new_root/"
Dowload the following two program binaries and place them under `./new_root/home/`
`http://www.cs.binghamton.edu/~huilu/slidesfall2020/loop`
`http://www.cs.binghamton.edu/~huilu/slidesfall2020/mem`

Now, the new root filesystem is ready. You can use it as the root file system of the container instance.

3. Namespaces and Cgroups: You can start writing the miniDocker.py. As listed in the program template, you are required to implement four namespaces, which are UTS, network, mount, and PID, and two control groups, which are cpuset and memory. Though you can implement them in different ways, your miniDocker is expected to have the following behaviors (expected output):

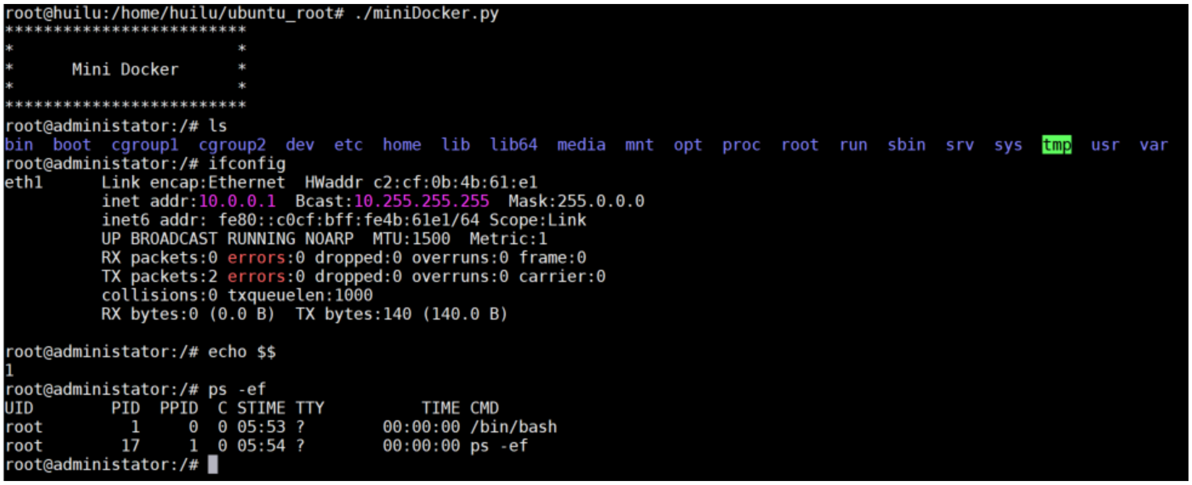
- As shown in Figure 1, once you execute your miniDocker.py, you will be dropped to a new bash process, which is the new container instance. In our template, we use "os.execle" to convert the child process of the miniDocker process to this new bash process. Please read the "exe_bash" function carefully.

  In the new container instance, the directory path would be under root "/". you can use "ls" command to check. You should have the same results as shown in Figure 1.

  The "hostname" of the container instance also changes to "administrator", which is the default one. Of course, you can pass a different hostname to miniDocker.py.

To check “network namespace”, you can use “ifconfig” command, which will display a network interface “eth1” (in Figure 1) with the associated IP address (“10.0.0.1”). You should be able to pass a different IP address to the container instance.

For the “PID namesapce”, you can use “echo $$” command, which will show PID **1**. This is because, in the new container instance, the PID starts from 1. Further, you can use “ps -ef” command to display all the processes in the container. You will see that it only lists two: one is the bash process and the other is the “ps” command. In contrast, if you run this command in the native host, it will show a lot of processes.

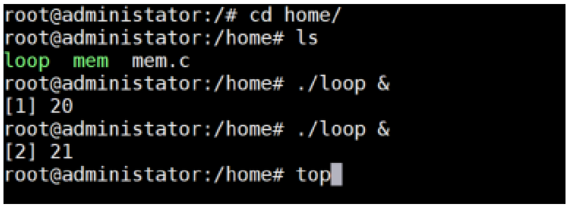


```
root@huilu:/home/huilu/ubuntu_root# ./miniDocker.py
************************
*                      *
*      Mini Docker     *
*                      *
************************
root@administator:/# ls
bin  boot  cgroup1  cgroup2  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@administator:/# ifconfig
eth1      Link encap:Ethernet  HWaddr c2:cf:0b:4b:61:e1
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::c0cf:bff:fe4b:61e1/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

root@administator:/# echo $$
1
root@administator:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 05:53 ?        00:00:00 /bin/bash
root        17     1  0 05:54 ?        00:00:00 ps -ef
root@administator:/# 
```

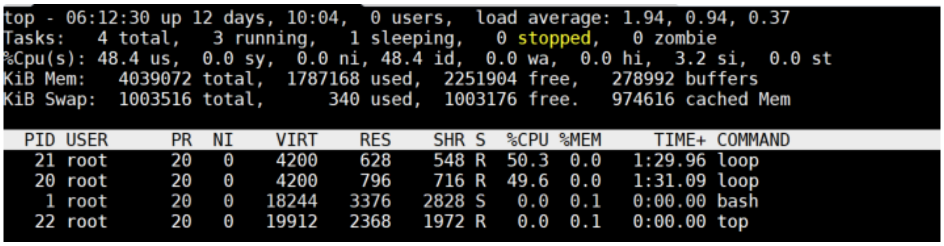
Figure 1: Execute your miniDocker.py.

- For the “cpuset cgroup”, you can check it by running the “loop” program. As the Watson School VM only has two CPUs, we configure the container instance to only use one CPU, which is also the default number in the template program. You can run two instances of the “loop” program as shown in Figure 2. Then, we use “top” command to check whether the cpuset cgroup works or not: If each of the “loop” processes consumes around 50% CPU usage, it means the cpuset cgroup is controlling the cpu usage, as illustrated in Figure 3.



```
root@administator:/# cd home/
root@administator:/home# ls
loop  mem  mem.c
root@administator:/home# ./loop &
[1] 20
root@administator:/home# ./loop &
[2] 21
root@administator:/home# top
```

Figure 2: Test cpuset Cgroup: command.



```
top - 06:12:30 up 12 days, 10:04,  0 users,  load average: 1.94, 0.94, 0.37
Tasks:   4 total,   3 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s): 48.4 us,  0.0 sy,  0.0 ni, 48.4 id,  0.0 wa,  0.0 hi,  3.2 si,  0.0 st
KiB Mem:   4039072 total,  1787168 used,  2251904 free,   278992 buffers
KiB Swap:  1003516 total,      340 used,  1003176 free.   974616 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
   21 root      20   0    4200    628    548 R  50.3  0.0   1:29.96 loop
   20 root      20   0    4200    796    716 R  49.6  0.0   1:31.09 loop
    1 root      20   0   18244   3376   2828 S   0.0  0.1   0:00.00 bash
   22 root      20   0   19912   2368   1972 R   0.0  0.1   0:00.00 top
```

Figure 3: Test cpuset Cgroup: resutls.

- Similarly, for the “memory cgroup”, you can check it by running the “mem” program. The default memory limit (in bytes) in the template program is set to 10M. Again, you should be able to pass a different memory limit to your container instance. You can use "top" command to check memory usage as well, reported in the "%MEM" field. As the Watson School's VM's memory is 4GB, the memory usage of the "mem" process will be **0.2%** given the 10MB memory limit.

4. Hints: 1) You probably can guess something from the "import" statements – we imported "unshare". Python has the unshare module for namespaces related configurations/operations. Please search this module a little bit and also refer to the linux manual: `https://man7.org/linux/man-pages/man2/unshare.2.html` for the unshare types. 2) We might need to rely on the "os" module to execute a lot of commands (e.g., `os.system()`) for configuring namespaces. 3) We did use "os.chroot" function. 4) We also need to deal with the "/proc" directory.

6 Submission & Grading

Submit your assignment report as one pdf file and the miniDocker.py code file (one file) on the blackboard. You will also schedule an appointment with TA to have a demo before the deadline. **Notice that it is your responsibility to schedule the demo time. You will lose points if you fail to do so**. We will grade your assignment based this report, code, and the demo. Basically, you should fully understand what you are doing (not simply follow the instructions).

The report should include (but not limited to):

- Present how you conduct your performance measurements in Docker for the CPU and fileio test cases, including detailed configurations of your experimental setup and main steps – 10 points
- Present how you use performance tools to collect performance data. For CPU utilization, you should at least divide them into two parts including user-level and kernel-level. For I/O, you should present I/O throughput, latency, and disk utilization – 10 points
- Please understand the performance data, analyze the data, and then present them in an understandable way (figures and/or tables) in your report. – 10 points
- Please answer the question, why is the QEMU based VM so slow (to install and to execute)? – 10 points
- Writing – 5 points

During the demo, you will be asked to:

- Run your docker container and QEMU VM – 5 points
- Run your miniDocker.py:
  Jump to bash – 5 points
  UTS namespace – 5 points
  network namespace – 10 points
  mount namespace – 10 points
  pid namespace – 5 points
  cpuset cgroup – 10 points
  memory cgroup – 5 points

7 Appendix

Please follow the instructions in this article [7] to enable GUI for the native (your Google instance). Notice that, however, there are two places you need to pay attention, and following the instructions below may solve possible problems.

7.1 .vnc/xstartup

Replace the whole content of .vnc/xstartup with the following content (DO NOT follow the article above).

```
#!/bin/sh
def
export XKL_XMODMAP_DISABLE=1
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

metacity &
gnome-settings-daemon &
gnome-panel &
nautilus &
gnome-terminal &
```

7.2 Open the firewall

Navigate to the “Firewall rules” item (in Google Cloud Console) shown as in Figure 4. Create a new firewall rule as in Figure 5. Notice that though this is an easy way to open ports for vncserver, it is not a safe solution (as we open too many ports). You can choose to use a safer way to do so.

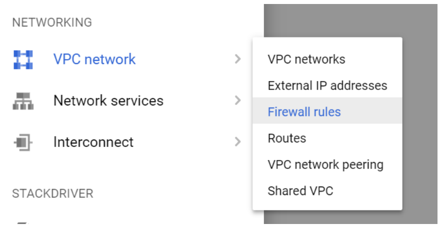


Figure 4: Open firewall rules.

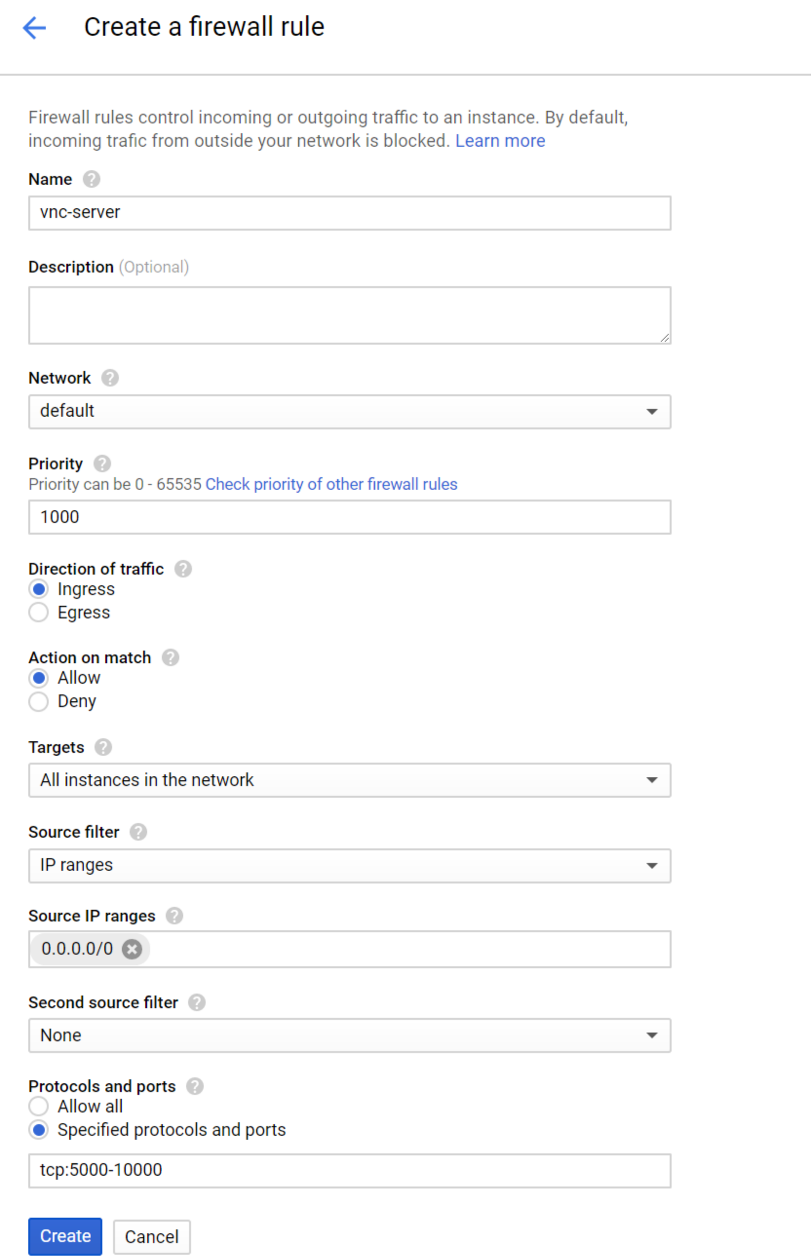
References

[1] How to Start an Instance in GCP. `http://www.cs.binghamton.edu/~huilu/slidesfall2020/How_to_create_VM_in_GCP.pdf`.

[2] Docker Installation. `https://docs.docker.com/install/linux/docker-ce/ubuntu/`.

[3] Running your first container. `https://github.com/docker/labs/blob/master/beginner/chapters/alpine.md`.

[4] Sysbench. `http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf`.



Create a firewall rule

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming trafic from outside your network is blocked. Learn more

Name
vnc-server

Description (Optional)

Network
default

Priority
Priority can be 0 - 65535 Check priority of other firewall rules
1000

Direction of traffic
● Ingress
○ Egress

Action on match
● Allow
○ Deny

Targets
All instances in the network

Source filter
IP ranges

Source IP ranges
0.0.0.0/0

Second source filter
None

Protocols and ports
○ Allow all
● Specified protocols and ports
tcp:5000-10000

Figure 5: Firewall rules.

[5] IOSTAT. https://linux.die.net/man/1/iostat.

[6] QEMU https://qemu.weilnetz.de/doc/.

[7] Graphical user interface (GUI) for Google Compute Engine instance. https://medium.com/google-cloud/graphical-user-interface-gui-for-google-compute-engine-instance-78fccda09e5c.