CSEN 241 HW 1
System vs OS Virtualization

Learning Objectives

The aims of the assignment include the following learning objectives:

- Understanding the real-world use case of two virtualization techniques / technologies:
  - System Virtualization via QEMU
  - OS Virtualization via Docker
- Understand the operations related to each technologies
- Understand and demonstrate the performance differences of each technology using benchmarks and finally compare the results in a comprehensive report.

Environment Setup

You are free to use your personal computer for this assignment. The set of requirements for using your personal computers is as follows:

- x64 CPU with at least 2 cores or a Mac Apple Silicon
- 4 GB memory
- 20 GB free disk space
- OS: Windows, Linux (Ubuntu preferred) or Mac OS X

If you are using a personal computer that does not support such requirements, please do reach out to me, so that we can discuss some options.

System Virtualization (QEMU) Setup

What is QEMU?

QEMU is a free and open-source hypervisor. It 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. It is more lightweight than virtualbox and allows near-native speed (when using QEMU with KVM on Linux). QEMU supports various types of CPUs, such as x64 and ARM, and also supports integrations with various host operating systems. It is less user-friendly and requires more effort to setup.

Installing QEMU

For All Operating Systems

For this assignment, you will be installing a Ubuntu guest virtual machine. Thus you must download the appropriate Ubuntu 20.04 or 16.04 server ISO image from one of the links below:

- x64 CPU on Linux or Mac: Ubuntu 20.04 Server
- x64 CPU on Windows: Ubuntu 16.04 Server
- ARM (Apple Silicon): Ubuntu 20.04 Server for ARM

From here, the installation instruction varies between which OS you are running.

Optional helpful tips:

- You can set `-nographic` to run directly in the terminal.
- Another option is to SSH from your host machine to the VM by adding the following flags to enable port mapping from 8888 to 22 and run the SSH command shown below.
    - `-netdev user,id=net0,hostfwd=tcp::8888-:22`
    - `$ ssh -p 8888 localhost`
- Do the above only if you have successfully launched a VM!

Linux Installation instructions

QEMU is the easiest to install if you have Ubuntu. You can install qemu (under Ubuntu) simply by typing the following command in the terminal:

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

You can then create the QEMU Image by running the following command.

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

Given the image, install the 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_SERVER_ISO_FILE_NAME] -m 2046 -boot strict=on
```

For each of these commands, please refer to the QEMU manual for more details [1].

Mac OS X (x64) Installation instructions

The instructions to install QEMU on the x64 Macs can be found in the following link https://wiki.qemu.org/Hosts/Mac. I suggest using the homebrew method as it is the easiest. You can find homebrew installation instructions in [2], if it is not already installed in your machine.

Once QEMU is installed, you can follow a similar procedure as the Linux installation to create a QEMU image and install the VM. Things to watch out for:

- Make sure to update homebrew
- Try 20.04, if it doesn't work 16.04 may work.

Mac OS X (Apple Silicon) Installation instructions

Given that Apple Silicon is a relatively new processor, there are not many resources available to simply install QEMU with a single command line. Once you install qemu using brew as similar to above, **first make sure you have version 8.2.0+**. If not, you should upgrade to 6.2.0.

Also, make sure you are not running your terminal via Rosetta. To check, type in `arch` in the terminal to make sure it is arm64, not i386.

Here is a set of commands that **may** work:

-
  ```
  $ qemu-system-aarch64 \
       -accel hvf -cpu cortex-a57 -M virt,highmem=off -m 2048 
  -smp 2 \
       -drive 
  file=/usr/local/share/qemu/edk2-aarch64-code.fd,if=pflash,format
  =raw,readonly=on \
       -drive if=none,file=ubuntu.img,format=qcow2,id=hd0 \
       -device virtio-blk-device,drive=hd0,serial="dummyserial" \
       -device virtio-net-device,netdev=net0 \
       -netdev user,id=net0 \
       -vga none -device ramfb \
       -cdrom ubuntu-20.04.4-live-server-arm64.iso \
       -device usb-ehci -device usb-kbd -device usb-mouse -usb \
       -nographic
  ```

Note that the VM image file names may be different based on which version you have downloaded. Make sure you download the ARM64 image.

Windows Installation Instructions

Running QEMU on Windows is **NOT RECOMMENDED**, as most servers are Linux/Unix based and most open source virtualization works best with Unix based OSes. However, it is certainly possible. Here is an article that describes the entire process in detail in [3], which requires you to be able to change some Window environments variables.

- You may need to install VcXsrv to connect to the VM instance. Once you have connected Ubuntu to the display, follow the Linux instructions on the homework.

OS Virtualization (Docker) Setup

What is 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.

Installing Docker

While we will be installing Docker Desktop (for Windows and Mac), we will be interacting mostly with Docker CLI. Docker Desktop installation will include Docker Engine, Docker CLI client, Docker Compose and more. You can learn more about the Docker CLI in [5]. I have included the corresponding links to find the installation instructions for each type of OSes. Installing Docker should be much simpler than installing QEMU.

Linux Installation instructions

Note: Ubuntu does not come with Docker Desktop, so we would just need to install the Docker Engine and the Docker CLI separately.

- https://docs.docker.com/engine/install/ubuntu/

Mac OS X Installation instructions

- x64 CPU: https://docs.docker.com/desktop/mac/install/
- ARM (Apple Silicon):: https://docs.docker.com/desktop/mac/apple-silicon/

Windows Installation instructions

- https://docs.docker.com/desktop/windows/install/

Installing sysbench

Sysbench[4] is an open-source and multi-purpose benchmark utility that evaluates the parameter features tests for CPU, memory, I/O and much more. We will use sysbench to understand the performance characteristics of each of the virtualization technologies.
https://github.com/akopytov/sysbench/issues/140

For Ubuntu Guest VM

Sysbench installation on Ubuntu is very simple. Just run the following commands:

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

Note: Do not be confused with the password for the guest VM.

For Docker

For Docker, you will be **creating your own image by adding sysbench to a base image**. Based on your choice of Ubuntu version and your architecture, you will choose a base image in Docker Hub. You can find the correct Docker docker images in https://hub.docker.com/_/ubuntu. Make sure to match the ubuntu version and your chip! Very important!
Once the base image is downloaded, you must install sysbench similar to how you installed it on the VM. Note that the base image is quite light, so it may not contain simple commands like sudo or ping.

You must describe the commands you ran to create your image, your final image ID, and the screenshot of your image history!

Note on Sysbench Versions!

The sysbench version on your linux machine and the Docker container may differ, as the sysbench version is updated continuously. Therefore, the output values may seem like they are different. To eliminate this issue, make sure you have the same version of sysbench on both experiments.

What to Experiment?

Given the sysbench tool, we can have some basic ideas about the system performance by running it against the system under test. In assignment, we will focus on the **cpu, memory 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 measurement. Please read them carefully, and then produce the measurement results. If the results are incorrectly generated and/or can not be well justified, you will not be able to get a full credit for this assignment.

The following should be done on **Docker and QEMU Separately**:

- For QEMU Only: Create different types of disk images. See https://www.qemu.org/docs/master/system/images.html
  - Create two images with different disk image types, raw, qcow2
  - Document each command you used.

- For QEMU only: Test different two arguments of QEMU for CPU and RAM each. See https://www.qemu.org/docs/master/system/qemu-manpage.html
  - Hint: Use -smp and -m flags.
  - Be sure to document your experiments and why you used them.
    - **Do not just copy other students' arguments.**

- For Docker only: Test two different arguments for docker run with CPU and RAM each. See https://docs.docker.com/engine/reference/commandline/run/
  - Hint: Use --cpus and --memory
  - Be sure to document your experiments and why you used them.
    - **Do not just copy other students' arguments.**

- You need to select the Sysbench test cases for the cpu, fileio and memory test modes.
  - You need a **total of two sysbench test cases for CPU, two test cases for fileio and two test cases for memory**.
  - For QEMU: For each case, you must run them each on a different disk image and four modes of QEMU CPU and RAM combination as provided above.
  - For Docker : For each case, you must run them each on four modes Docker CPU and RAM combination as provided above.

- For each sysbench test case, you need to figure out the"right" parameters. Some examples are,
  1. For CPU Test: Choose 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).
     a. Note that if your experiment always runs for 10 seconds, change the time parameter and report events/sec.
  2. For memory test: Change the --memory-block-size
  3. For fileIO test: Change the --file-test-mode flag
  - Usually, it's reasonable to make a test case lasting for at least 30 seconds, but for such test cases, please justify your configurations for doing so.

- Total test cases should be:
  - **QEMU: 2 disk drives x 2 QEMU CPU x 2 QEMU Memory x 6 sysbench = 48**
  - **Docker: 2 Docker CPU setting x 2 Docker Memory setting x 6 sysbench = 24**

- You need to repeat the sysbench measurement **at least 5 times** for each test case, and report the **average**, **min, max and std.** values 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 a similar test environment.
  - 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 from the 1 memory cache instead of disks. You have to manually drop such cache in the Host using the commands (in superuser or administrator mode):
    - For Linux:
      ```
      echo3 > /proc/sys/vm/dropcaches
      ```

        - For Mac OS:
          `sync && sudo purge`

        - For Windows (Windows, this is not readily available, thus you must download the tool from here):
          `sync`

- For each of the experimental runs, **create a bash script to automate the experiment.**

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

Submission Instruction

Submit a PDF for the assignment on the Camino containing the following pieces of information:

1. The Report content
2. A Github URL such that the command 'git clone URL' would download your repository (assuming that the repository has been shared with the requesting user);
3. A git commit ID within your repository that you want markers to assess that contains the following.
    a. Shell scripts to run your experiments
4. You will reuse this repository for the entire class, so do not name it something like CSEN-HW1.

Repository Requirements

- You are expected to work on your assignment within a git repository. The easiest way to acquire URLs that reach your repository is to push your work to a (free) cloud-based git service, such as GitHub, Bitbucket, GitLab, or Altitude (a GitLab server run locally within the Department of Computer Science that you can access). The aforementioned cloud services allow you to create private repositories.
- **You will create a SINGLE repository for all homework assignments and projects, so make sure you create a folder called 'hw1' to store all your HW 1 files.**
- You must invite teaching staff to collaborate on your repository, so that they can access the URL that you submit. Please send an invitation to sean.choi@scu.edu
- Do not apply git history-rewriting operations to your repository: it is important that the markers can see how your commits completed your assignment, and how you worked on the assignment over time.

Rubric / What to Write on the Report

- Your report should be a single PDF file. Most of the grade on this assignment is based on this report. The PDF file should be uploaded to camino and have the link to the repo containing the other scripts.
- You should not only follow the instructions to do your assignment, but also fully understand what you are doing
- The report should include (but not limited to):
    - Creation of three different disk images: **5 points**
    - Present main steps to enable a QEMU VM. In addition, please present the detailed QEMU commands, and VM configurations: **10 points**
    - Present main steps to create the Docker container. **This must show your steps in creating your own image and your image history! Do not copy any classmate's image.** In addition, please describe the operations you use to manage Docker containers (and some other operations which you think are also important): **10 points**
    - Proof of experiment. Include screen snapshots of your Docker and QEMU running environments for each experiment: **10 points**
    - Present your measurements in given different scenarios for each virtualization technology: **20 points**
    - Shell scripts for running the experiment: **10 points**
    - Presentation and analysis of the performance data: **20 points**
    - Understandability/Neatness of your report and Git: **5 points**
    - Extra Credit I: Run the all the QEMU experiment on disk image with encryption. Use encrypt.format=luks and provide all experiment details. **5 points**
    - Extra Credit II: Provide the correct Vagrant and Docker files for your VM and Docker Container. **5 points (3 for vagrant 2 for Dockerfile)**

About Extra Credit II

For cloud deployment, automation is critical. Thus there are tools that enable developers to easily store the configurations of their VMs & Containers and launch them on the cloud. One such tool is Vagrant and the other is Docker File. Writing the configurations for these platforms requires practice and may take multiple tries before getting them right. However, if you are ambitious, you can try to write both Vagrant and Docker File for this assignment for future uses (and be rewarded hefty for it!). Note that Vagrant was created to be used with VirtualBox, so it may be tricky to run and debug the Vagrant file with QEMU. My suggestion is to do the assignment using QEMU, but build the Vagrant file using Virtualbox. If you are ambitious, you can try running Vagrant with QEMU as well! However, note that the extra credit is the same regardless of which hypervisor you decide to test the Vagrant file on.

Here are some references to help you write Vagrant and Docker Files

- Writing Vagrant File https://www.vagrantup.com/docs/vagrantfile

- Writing Docker File
  https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

For those of you who want to run Vagrant with QEMU, this would be the first step to try.

- https://computingforgeeks.com/using-vagrant-with-libvirt-on-linux/
- https://github.com/ppggff/vagrant-qemu

References

[1] QEMU Documentation: https://www.qemu.org/docs/master/
[2] Homebrew: https://brew.sh/
[3] How do I run QEMU on Windows: https://linuxhint.com/qemu-windows/
[4] Sysbench manual: https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf
[5] Docker CLI Reference: https://docs.docker.com/engine/reference/commandline/cli/
[6] Docker exec: https://docs.docker.com/engine/reference/commandline/exec/
[7] Docker Tutorials: https://docker-curriculum.com/