



東南大學
SOUTHEAST UNIVERSITY

云计算技术及应用 课程实践报告

标 题	虚拟化技术实践
学 号	232349
姓 名	李志成

东南大学计算机科学与工程学院

二零二四年三月

目录

实验一 虚拟化技术实践	2
1 实验目的及要求	2
2 实验环境	2
3 虚拟机镜像制作	3
3.1 安装QEMU	3
3.2 准备磁盘镜像和 ubuntu 操作系统	4
3.3 安装虚拟机	5
3.4 使用XML文件管理虚拟机	6
3.5 图形化界面运行	8
3.6 比较虚拟机镜像格式 raw 和 qcow2的区别	10
4 容器镜像制作	12
4.1 安装 docker	12
4.2 从 docker-hub 上下载 ubuntu20.04 镜像	13
4.3 使用ubuntu20.04 镜像运行一个容器	14
4.4 使用docker commit命令从容器创建一个新镜像	16
4.5 使用 dockerfile 编译镜像	16
4.6 比较 docker build 和 docker commit 创建镜像的区别	19
5 虚拟机和容器的启动速度比较	20
6 虚拟机和容器的镜像大小比较	20
7 总结	21

实验一 虚拟化技术实践

1 实验目的及要求

实验目的：

深入理解虚拟化技术原理：通过实际操作和实践，加深对虚拟机和容器技术基础原理的理解，包括它们的工作机制、应用场景以及在云计算环境中的重要性。

掌握虚拟机和容器的创建与配置过程：学习如何使用QEMU创建和配置虚拟机环境，以及如何利用Docker创建和配置容器，理解二者在实际应用中的不同之处。

性能分析与比较：对虚拟机和容器在启动速度、资源占用等方面进行比较分析，通过实际测试数据来评估各自的性能优劣。

实验要求：

详细记录实验步骤：在实验过程中，详细记录每一步的操作过程、使用的命令以及遇到的问题 and 相应的解决方案，确保实验报告的完整性和准确性。

对比分析：基于实验观察到的数据和结果，进行虚拟机与容器在启动速度、资源效率等方面的对比分析，提供客观的数据支持。

反思与总结：根据实验结果，对比反思虚拟机和容器的优缺点，以及在不同场景下的应用适用性。

2 实验环境

宿主机：

处理器：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

操作系统：Windows 11 23H2 22631.3155

虚拟机：

虚拟化软件：VMware® Workstation 17 Pro 17.5.1 build-23298084

操作系统：ubuntu-22.04.4-desktop-amd64

内存：2GB

处理器数量：2

每个处理器的内核数量：2

硬盘最大可用大小：20GB

3 虚拟机镜像制作

3.1 安装QEMU

在终端依次输入以下命令：

```
sudo apt-get update
sudo apt-get install qemu-kvm
sudo apt-get install qemu
sudo apt-get install virt-manager
sudo apt-get install virt-viewer
sudo apt-get install libvirt-daemon-system libvirt-clients bridge-utils libvirt-daemon
```

输入命令‘kvm-ok’，终端显示：

```
lzc@lzc-vm:~$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

输入命令‘virsh -c qemu:///system list’，终端显示：

```
lzc@lzc-vm:~$ virsh -c qemu:///system list
 Id    Name    State
-----
```

解释：

‘sudo apt-get install qemu-kvm’：安装QEMU的KVM（Kernel-based Virtual Machine）扩展。KVM是一个Linux内核内建的虚拟化解决方案，它允许运行多个隔离的虚拟机。对于支持硬件虚拟化的CPU，KVM能够提供更高效的虚拟化性能。简而言之，这个命令用于在支持KVM的系统上启用QEMU的高效虚拟化功能。

‘sudo apt-get install qemu’：安装QEMU，一个独立于KVM的虚拟化软件，可以用来创建和运行虚拟机，但不依赖于硬件虚拟化支持。

‘sudo apt-get install virt-manager’：安装Virtual Machine Manager，提供一个图形界面来管理虚拟机，让管理虚拟机更为简单直观。

‘sudo apt-get install virt-viewer’：安装Virtual Machine Viewer，这是一个用来查看和与运行在QEMU或KVM上的虚拟机交互的工具。

‘sudo apt-get install libvirt-daemon-system libvirt-clients bridge-utils libvirt-daemon’：安装libvirt及其相关组件，libvirt提供了一个管理虚拟化平台

（如KVM，QEMU）的框架。这个命令安装了libvirt的守护进程和客户端工具，以及设置网络桥接的工具。这些组件使得管理和交互虚拟机变得更容易。

‘kvm-ok’：检测 ‘/dev/kvm’ 是否存在，这是使用KVM虚拟化所必需的；确认硬件是否支持KVM（硬件虚拟化支持）并且该功能是否已在系统上启用。向用户提供关于他们的系统是否能高效运行KVM虚拟机的信息。

‘/dev/kvm exists’：这表明 ‘/dev/kvm’ 设备文件存在。在Linux系统中，这个设备文件是KVM（Kernel-based Virtual Machine）虚拟化的关键部分。其存在表明系统已经安装了KVM，并且内核已经正确地加载了KVM模块。

‘virsh -c qemu:///system list’：查看系统上当前活跃或存在的虚拟机。

‘qemu:///system’ 表示使用系统级别的连接（相对于用户级别的连接），它会连接到由libvirt守护程序 libvirtd 管理的所有虚拟机。

返回结果解释：‘Id’、‘Name’、‘State’字段均为空，意味着当前没有任何由libvirt守护程序管理的虚拟机正在运行或存在。

3.2 准备磁盘镜像和 ubuntu 操作系统

创建文件夹并为镜像预留空间：

```
mkdir 3_4_vir_machine
cd 3_4_vir_machine
qemu-img create -f qcow2 ubuntu.img 10G
```

终端输出：

```
lzc@lzc-vm:~/3_4_vir_machine$ qemu-img create -f qcow2 ubuntu.img 10G
Formatting 'ubuntu.img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib
size=10737418240 lazy_refcounts=off refcount_bits=16
```

‘qemu-img create -f qcow2 ubuntu.img 10G’：该命令创建了一个10GB大小的qcow2格式（QEMU Copy On Write version 2）的虚拟硬盘文件，命名为 "ubuntu.img"。这个文件将被用作创建虚拟机时的磁盘映像。命令的输出是对正在进行的格式化操作的描述，包括使用的格式、大小以及其他相关参数。

‘cluster_size=65536’：文件系统上的“集群大小”或“块大小”，它决定了文件系统的基本存储单元的大小。在这里，它被设置为65536字节（即64KB）。

‘extended_l2=off’：禁用L2缓存表的扩展。

‘compression_type=zlib’: 表明qcow2映像文件中的数据压缩使用的是zlib压缩。

‘size=10737418240’: 虚拟磁盘的大小，以字节为单位。在这里，它表示的是大约10GB的大小。

‘lazy_refcounts=off’: 禁用延迟引用计数。启用延迟引用计数可以提高某些操作的性能，但可能降低数据的完整性保护。

‘refcount_bits=16’: 使用16位引用计数。

下载 server 版本 linux 操作系统：（借助清华源镜像）

```
wget https://mirrors.tuna.tsinghua.edu.cn/ubuntu-releases/20.04.6/ubuntu-20.04.6-live-server-amd64.iso
```

3.3 安装虚拟机

通过以下指令安装虚拟机：

```
qemu-system-x86_64 -hda ubuntu.img -cdrom ./ubuntu-20.04.6-live-server-amd64.iso -enable-kvm -boot d -m 2048
```

解释：

这个命令是用来启动一个使用 QEMU 虚拟化软件的虚拟机，并在其中安装 Ubuntu 操作系统。

‘qemu-system-x86_64’: 这是启动 QEMU 虚拟机的主命令。‘x86_64’ 表明这个虚拟机模拟的是一个64位的x86架构的处理器。

‘-hda ubuntu.img’: 指定虚拟硬盘。‘-hda’ 表示作为第一块 IDE 硬盘，‘ubuntu.img’ 是之前使用 ‘qemu-img’ 命令创建的虚拟硬盘文件。

‘-cdrom ./ubuntu-20.04.6-live-server-amd64.iso’: 指定了一个ISO镜像作为虚拟CD-ROM驱动器的内容。

‘-enable-kvm’: 启用KVM（Kernel-based Virtual Machine）硬件加速。这可以显著提高虚拟机的性能，但需要处理器支持硬件虚拟化（比如Intel VT或AMD-V）。

‘-boot d’: 指定虚拟机从CD-ROM（即之前指定的Ubuntu安装ISO）启动。

‘-m 2048’: 为虚拟机分配2048MB（即2GB）的内存。

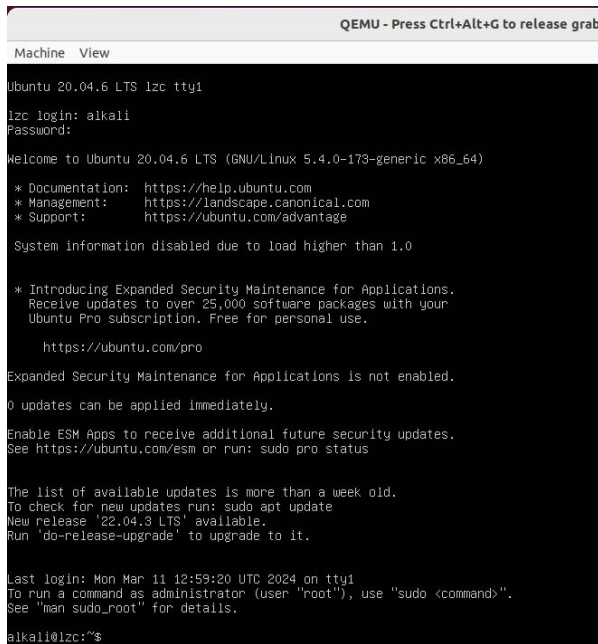
综合来看，这个命令将启动一个QEMU虚拟机，使用之前创建的 ubuntu.img 作为硬盘，从 Ubuntu 20.04.6 的安装盘启动，并尝试在其中安装Ubuntu操作系统。

安装成功后，输入以下指令退出系统：

```
shutdown -h now # 防止时间不同步
```

查看是否安装成功，登录后同样使用上一条指令关机：

```
qemu-system-x86_64 ubuntu.img -m 1024
```



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
Ubuntu 20.04.6 LTS lzc tty1
lzc login: alkali
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-173-generic x86_64)

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

System information disabled due to load higher than 1.0

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

   https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.
0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Mar 11 12:59:20 UTC 2024 on tty1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
alkali@lzc:~$
```

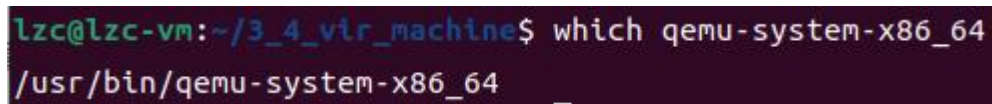
登录成功！

3.4 使用XML文件管理虚拟机

输入命令：

```
which qemu-system-x86_64
```

得到终端的输出：



```
lzc@lzc-vm:~/3_4_vir_machine$ which qemu-system-x86_64
/usr/bin/qemu-system-x86_64
```

在目录‘~/3_4_vir_machine’下，使用如下命令创建demo.xml：

```
touch demo.xml
```

使用文本编辑器编辑其内容：

```
<domain type='kvm'>
<name>demo</name>
<memory>1048576</memory> //1GB
<vcpu>8</vcpu> //8 个虚拟 cpu
<os>
<type arch='x86_64' machine='pc'>hvm</type>
<boot dev='cdrom'> //光盘启动
</os>
<features>
<acpi/>
<apic/>
```

```
<pae/>
</features>
<clock offset='localtime' />
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
<emulator>/usr/bin/qemu-system-x86_64</emulator>
<disk type='file' device='disk'>
<driver name='qemu' type='qcow2' />
<source file='/home/lzc/3_4_vir_machine/ubuntu.img' />
//目的镜像路径
<target dev='hda' bus='ide' />
</disk>
<input type='mouse' bus='ps2' />
<graphics type='vnc' port='-1' listen='0.0.0.0' autoport='yes' keymap='en-us' />
</devices>
</domain>
```

这个xml文件是一个完整的虚拟机配置文件，它定义了虚拟机的硬件配置、启动方式、存储、图形访问以及其他设置。通过libvirt工具（如virsh或virt-manager），可以使用这个文件来创建和启动一个名为“demo”的虚拟机。

进行准备工作——用户组绑定并开启权限：

```
sudo sed -i 's/#vnc_listen = "0.0.0.0"/vnc_listen = "0.0.0.0"/g' /etc/libvirt/qemu.conf
sudo sed -i 's/#group = "root"/group = "root"/g' /etc/libvirt/qemu.conf
sudo sed -i 's/#user = "root"/user = "root"/g' /etc/libvirt/qemu.conf
sudo service libvirtd restart
```

解释：

‘sed -i 's/#vnc_listen = "0.0.0.0"/vnc_listen = "0.0.0.0"/g' /etc/libvirt/qemu.conf’：将VNC监听地址设置为任意地址，允许从任何IP访问虚拟机的VNC服务。

‘sed -i 's/#group = "root"/group = "root"/g' /etc/libvirt/qemu.conf’：修改 libvirt 配置，将其运行组更改为 root。这意味着 libvirt 和所有相关的虚拟机将以 root 组的权限运行，这通常会提供更广泛的系统访问权限。

‘sed -i 's/#user = "root"/user = "root"/g' /etc/libvirt/qemu.conf’：将 libvirt 的运行用户更改为 root。这给予 libvirt 进程最高级别的访问权限。

‘service libvirtd restart’：重启 libvirtd 服务，这是必要的步骤来使之前所做的配置更改生效。重启服务后，任何更新或修改的配置将被应用，包括 VNC 监听设置和运行权限的更改。

定义、启动虚拟机：

```
virsh define demo.xml
virsh start demo
sudo virsh -c qemu:///system list # 应该可以看到 demo 在运行并有相应 id
```

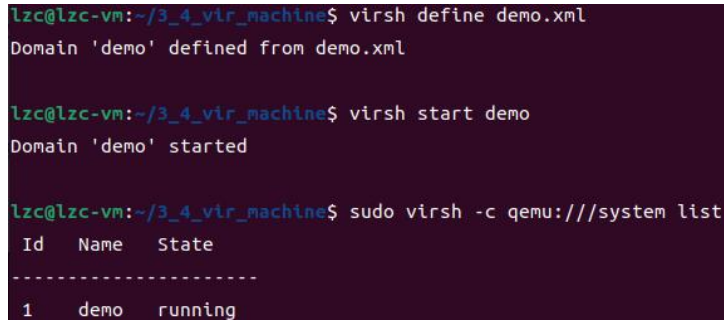
解释：

‘**virsh define demo.xml**’：从 demo.xml 文件定义（创建）一个新的虚拟机配置。define 操作只是在 libvirt 的虚拟机配置中添加了 this 虚拟机，但并不会启动它。虚拟机的配置信息会被存储，以便以后可以启动或修改。

‘**virsh start demo**’：启动一个名为“demo”的虚拟机。此虚拟机必须已经被定义过。启动命令会初始化虚拟机并运行其操作系统，根据定义的配置（如分配的内存、CPU和磁盘等）设置虚拟硬件。

‘**virsh -c qemu:///system list**’：列出当前所有由 libvirt 管理的虚拟机的状态。**‘-c qemu:///system’** 指定连接到 libvirt daemon，管理整个系统（而非单个用户）的所有虚拟机。list 参数显示所有定义的虚拟机及其当前状态（如运行、关闭等）。

终端效果：



```
lzc@lzc-vm:~/3_4_vir_machine$ virsh define demo.xml
Domain 'demo' defined from demo.xml

lzc@lzc-vm:~/3_4_vir_machine$ virsh start demo
Domain 'demo' started

lzc@lzc-vm:~/3_4_vir_machine$ sudo virsh -c qemu:///system list
Id   Name   State
-----
1    demo   running
```

我们可以看到虚拟机 demo 在运行，Id 为 1。

3.5 图形化界面运行

在终端输入以下命令启动虚拟机：

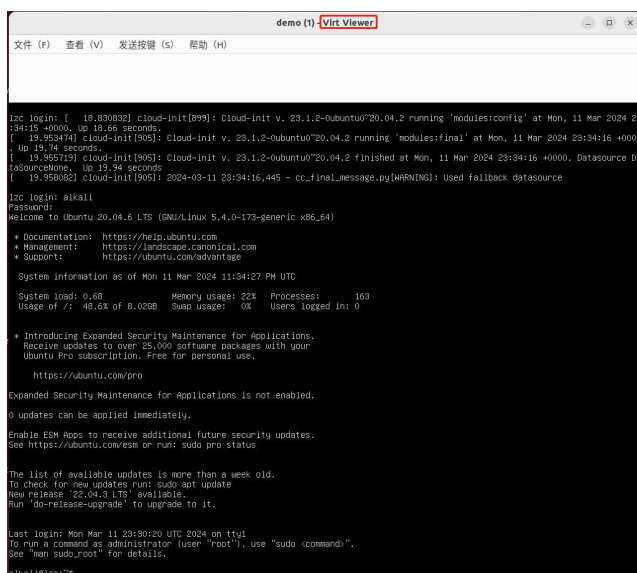
```
virt-viewer -c qemu:///system 1
```

使用 virt-viewer 工具来连接并查看虚拟机的图形界面。virt-viewer 是一个用于显示虚拟机的图形化界面的程序。它允许用户与虚拟机的图形化桌面或安装界面进行交互，类似于在物理机上使用显示器。**‘-c qemu:///system’** 这个选项指定 virt-viewer 连接到由 libvirt 管理的、运行于 QEMU 上的虚拟机。**‘qemu:///system’** 是用于访问由系统级 libvirt 守护进

程管理的所有虚拟机的URI。这意味着 virt-viewer 将连接到守护进程，该守护进程拥有管理整个系统上所有虚拟机的权限。这里的‘1’是虚拟机的 Id。

综上所述，‘virt-viewer -c qemu:///system 1’这个命令是用来打开 Id 为 1 的虚拟机的图形界面，使得用户可以像在物理机上操作一样，通过图形界面与该虚拟机进行交互。

效果：

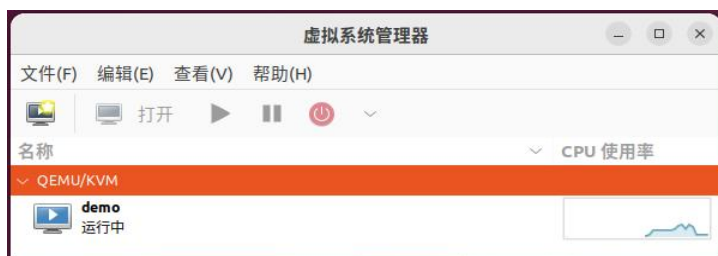


成功以 virt-viewer 工具的方式打开了虚拟机 demo。因为前面下载的 ubuntu 是 server 版本的，不是 desktop 版本的，所以本身不存在图形界面，因而无法被 virt-viewer 工具图形化。

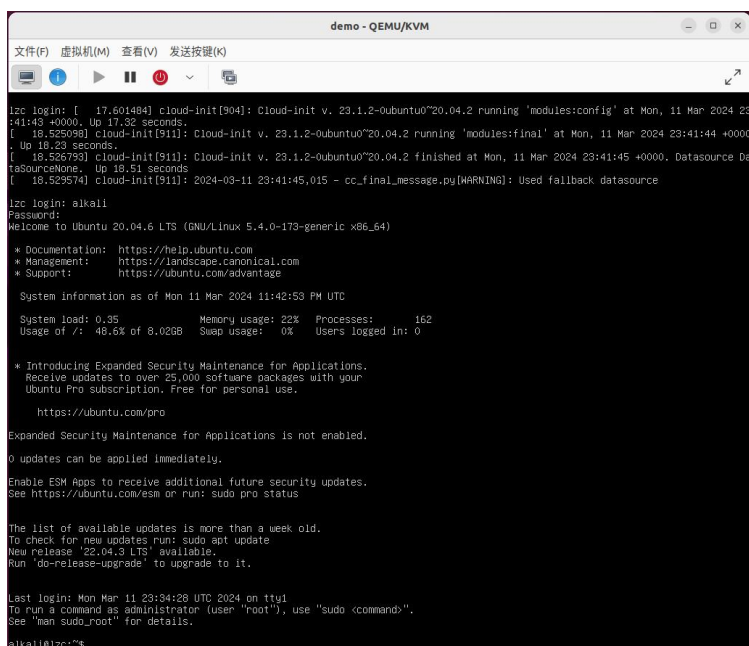
使用 virt-manager 启动图形化界面：

```
sudo virt-manager
```

效果：



双击下方 demo 虚拟机，即可以图形界面方式打开它。



成功以 virt-manager 工具的方式打开了虚拟机 demo。因为前面下载的 ubuntu 是 server 版本的，不是 desktop 版本的，所以本身不存在图形界面，因而无法被 virt-manager 工具图形化。

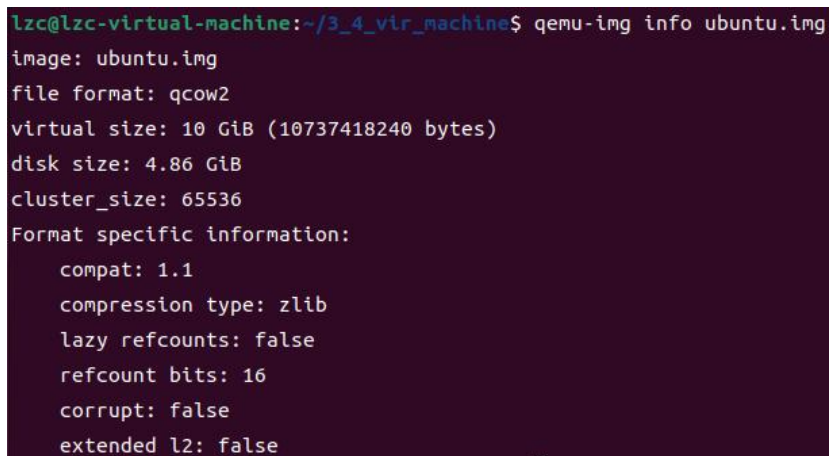
3.6 比较虚拟机镜像格式 raw 和 qcow2 的区别

输入命令：

```
qemu-img info ubuntu.img
```

获取 ubuntu.img 的磁盘镜像文件的信息。

终端显示：



解释：

image: 查询的镜像文件名 ubuntu.img

file format: 镜像的文件格式。这里是 qcow2，这是一种常用的虚拟机磁盘文件格式，支持诸如写时复制、快照等特性。

virtual size: 镜像文件在虚拟机中表现出的大小，这里为 10 GiB (10 吉字节)，即 10737418240 字节。这是虚拟机看到的磁盘大小，不必然是文件在物理硬盘上的实际大小。

disk size: 实际占用的磁盘空间大小。这里是 4.86 GiB，意味着虽然虚拟磁盘大小为 10 GiB，但实际只有 4.86 GiB 的数据存储在磁盘上。这体现了 qcow2 格式的空间优化特性。

cluster_size: 镜像文件中的群集大小，这里为 65536 字节。群集是 qcow2 文件存储数据的基本单元。

Format specific information: 这部分提供了 qcow2 格式特有的信息：

compat: 兼容性版本，1.1 表明此镜像兼容于 qcow2 版本 1.1 的特性。

compression type: 镜像使用的压缩类型，这里是 zlib。

lazy refcounts: 是否启用了延迟引用计数，这里为 false，意味着引用计数更新是即时的。

refcount bits: 引用计数位的数量，这里为 16 位，影响文件的引用计数方式。

corrupt: 指示镜像文件是否损坏，这里的 false 表明文件正常。

extended l2: 是否启用了扩展的 L2 缓存，这里为 false。

输入命令：

```
qemu-img convert -f qcow2 -O raw ubuntu.img Ubuntu-raw.img
```

该命令将一个名为 ubuntu.img 的 qcow2 格式的虚拟机磁盘镜像文件转换成一个名为 Ubuntu-raw.img 的 raw 格式的文件。raw 格式是一种更简单、直接映射到磁盘的存储方式，而 qcow2 是一种更高级的格式，支持诸如动态分配空间、快照等特性。

使用命令‘qemu-img info Ubuntu-raw.img’查看 raw 格式的虚拟机磁盘镜像文件，终端显示：

```
lzc@lzc-virtual-machine:~/3_4_vir_machine$ qemu-img info Ubuntu-raw.img
image: Ubuntu-raw.img
file format: raw
virtual size: 10 GiB (10737418240 bytes)
disk size: 4.45 GiB
```

虚拟机镜像格式 raw 和 qcow2 的区别：

qcow2 格式：

(1) 动态分配: qcow2 文件是动态分配的。这意味着文件初始大小很小, 并随着向虚拟磁盘写入数据而增长, 但不会超过其设定的最大大小。

(2) 快照支持: qcow2 支持快照。可以创建虚拟机的多个状态的快照而无需复制整个磁盘的内容。

(3) 压缩: qcow2 文件可以使用压缩来减少存储空间的使用。

(4) 加密: qcow2 格式支持加密。

(5) 性能: 相较于 raw 格式, qcow2 在某些情况下可能有略微的性能开销, 因为它需要处理额外的功能, 如快照管理和文件的动态扩展。

raw 格式:

(1) 简单性和性能: raw 格式是最简单的磁盘映像格式, 它是一个未经处理的磁盘映像。由于它不包含任何额外的元数据或特性, 通常提供最好的性能。

(2) 固定大小: raw 文件通常有固定的大小, 它等于虚拟磁盘的完整大小, 不论实际使用的空间有多少。

(3) 没有额外特性: raw 格式不支持动态分配、快照、压缩或加密。

(4) 兼容性: 由于其简单性, raw 格式通常与更多的虚拟化平台兼容。

两种虚拟机镜像格式的取舍:

qcow2 是一个更加灵活的选择, 特别是当你需要快照、动态分配或压缩功能时。它适用于需要频繁修改虚拟机状态的场景, 比如开发和测试环境。

raw 格式更适合性能敏感的应用, 如生产环境中的数据库服务器。如果你不需要额外的特性, 比如快照, 那么 raw 格式可能是一个更好的选择。

4 容器镜像制作

4.1 安装 docker

在终端输入以下命令, 安装 docker:

```
# 更新软件包索引
sudo apt update
# 在 ubuntu 中安装 docker 以及 Docker Compose
sudo apt install docker.io docker-compose
# 验证 Docker 安装
sudo docker --version
# 测试 Docker 是否工作正常
sudo docker run hello-world
```

对于命令‘sudo docker --version’，终端显示：

```
lzc@lzc-virtual-machine:~$ sudo docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
```

对于命令‘sudo docker run hello-world’，终端显示：

```
lzc@lzc-virtual-machine:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

说明，docker 可以正常工作了。

4.2 从 docker-hub 上下载 ubuntu20.04 镜像

输入以下命令：

```
docker pull ubuntu:focal
```

终端显示：

```
lzc@lzc-virtual-machine:~$ docker pull ubuntu:focal
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock:
Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/create?fromImage=ubuntu&tag=focal": dial unix /va
r/run/docker.sock: connect: permission denied
```

因为当前用户没有权限访问 docker 守护进程。docker 守护进程通常需要 root 用户权限或属于 docker 用户组的用户才能访问。要解决这个问题，可以通过将当前用户添加到 docker 用户组来获取所需的权限。这样做之后，用户就能够运行 docker 命令而不需要‘sudo’。

在终端运行以下命令：

```
sudo usermod -aG docker ${USER}
```

这个命令会将当前用户添加到 docker 组。‘\${USER}’ 是一个环境变量，代表当前登录的用户名。为了使组更改生效，需要注销并重新登录，或者重启系统。

再次在终端输入命令 ‘docker pull ubuntu:focal’，终端显示：

```
lzc@lzc-virtual-machine:/etc/docker$ docker pull ubuntu:focal
focal: Pulling from library/ubuntu
17d0386c2fff: Pull complete
Digest: sha256:80ef4a44043dec4490506e6cc4289eeda2d106a70148b74b5ae91ee670e9c35d
Status: Downloaded newer image for ubuntu:focal
docker.io/library/ubuntu:focal
```

这段输出信息表明我成功地从 Docker Hub 下载了 ubuntu:focal 镜像（下载过慢，需要更换国内镜像源）。

17d0386c2fff: Pull complete: 这一行显示 Docker 成功下载了镜像的一个层。Docker 镜像是由多个层组成的，每个层都包含了文件系统的一部分。完成所有层的下载后，Docker 将这些层组合起来以形成完整的镜像。

Digest: sha256:80ef4a44043dec4490506e6cc4289eeda2d106a70148b74b5ae91ee670e9c35d: 这是镜像的 SHA256 摘要，它是镜像内容的一个唯一指纹。可以使用这个摘要来验证下载的镜像是否正确和完整。

Status: Downloaded newer image for ubuntu:focal: 这一行说明已经成功下载了最新的 ubuntu:focal 镜像。

docker.io/library/ubuntu:focal: 这是我刚刚下载的镜像的完整名称，包括 Docker Hub 的地址 (docker.io)、仓库名称 (library/ubuntu) 以及镜像的标签 (focal)。

4.3 使用ubuntu20.04 镜像运行一个容器

在终端输入以下命令：

```
docker run -it --name my-container ubuntu:focal bash
```

这个命令在 Docker 中启动了一个新的 Ubuntu 20.04 (focal) 容器，并提供了一个可交互的 Bash shell 环境。具体来说：

‘docker run’：启动一个新容器的命令。

‘-it’：组合选项，使容器在交互模式下运行，并分配一个虚拟终端，允许用户直接与容器内的 shell 交互。

‘--name my-container’：为新容器指定名字“my-container”，方便日后管理（如启动、停止）。

‘ubuntu:focal’：指定使用 Ubuntu 20.04 LTS 的 Docker 镜像。

‘bash’：容器启动后执行的命令，此处启动了 Bash shell。

简而言之，这个命令创建了一个名为“my-container”的新容器，运行 Ubuntu 20.04，并打开了一个 Bash shell，让用户可以在其中执行命令。

终端显示：

```
lzc@lzc-virtual-machine:/etc/docker$ docker run -it --name my-container ubuntu:focal bash
root@e7d581864459:/#
```

可以看到出现了一个Bash shell。

可以在容器中测试任意的 linux 命令：

```
root@e7d581864459:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr
```

```
root@e7d581864459:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3364 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3452 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1191 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [29.7 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [3927 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1486 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.4 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3517 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [28.6 kB]
Get:18 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [55.2 kB]
Fetched 30.5 MB in 8s (3857 kB/s)
Reading package lists... Done
```

```
root@e7d581864459:/# exit
exit
```


4.4 使用docker commit命令从容器创建一个新镜像

在终端输入以下命令：

```
docker commit -m "Just installed a new software" -a "lizhicheng" my-container my-image1
```

该命令从一个已存在的 Docker 容器创建一个新的镜像。命令中的各个部分具有以下含义：

‘docker commit’：这是 Docker 中用于从容器创建一个新镜像的命令。

‘-m "Just installed a new software"’：‘-m’ 或 ‘-message’ 参数允许用户为这次提交添加一条注释信息。

‘-a "lizhicheng"’：‘-a’ 或 ‘-author’ 参数用于指定镜像的作者的名字。

‘my-container’：这指的是要从中创建镜像的容器的名称或 ID。在这个例子中，“my-container” 是之前通过 docker run 创建的容器的名字。

‘my-image1’：这是新创建镜像的名字。使用这个名字，用户以后可以通过 ‘docker run’ 命令启动基于这个新镜像的容器。

总结来说，这个命令创建了一个名为 “my-image1” 的新 Docker 镜像，这个镜像是基于 “my-container” 容器的当前状态。

终端显示：

```
lzc@lzc-virtual-machine:/etc/docker$ docker commit -m "Just installed a new software" -a "lizhicheng" my-container my-image1
sha256:8179e0a4bd59ff10bf5f0770681de91f7248b6908a1160c01239b57e925bdb7d
```

这条消息sha256:8179e0a4bd59ff10bf5f0770681de91f7248b6908a1160c01239b57e925bdb7d 表示我成功使用 ‘docker commit’ 命令从一个现有容器创建了一个新的镜像。

4.5 使用 dockerfile 编译镜像

在终端输入以下命令，可以查看所有的镜像信息：

```
docker images
```

终端显示：

```
lzc@lzc-virtual-machine:/etc/docker$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-image1            latest             8179e0a4bd59       3 minutes ago      248MB
ubuntu               focal              3cff1c6ff37e       3 weeks ago        72.8MB
hello-world          latest             d2c94e258dcb       10 months ago      13.3kB
```

在 Docker 镜像中，“TAG”是用来指定特定镜像版本的标签。每个镜像可以有多个标签，这些标签可以帮助用户区分不同的镜像版本或构建。

latest: latest 标签通常用于指示某个镜像仓库中最新的稳定版本。当用户拉取一个镜像但没有指定特定的标签时（例如，仅使用 `docker pull ubuntu`），Docker 默认使用 latest 标签。但需要注意，latest 并不总是指最新版本的镜像，它仅代表镜像仓库中标记为 latest 的那个版本。

focal: focal 是 Ubuntu 20.04 LTS 版本的代号，这个标签在 Docker 镜像中用于指示基于 Ubuntu 20.04 LTS 版本构建的镜像。

使用 `ubuntu:focal` 时，用户可以确保拉取的是基于 Ubuntu 20.04 LTS 的 Docker 镜像。每个标签都代表了镜像的一个不同版本，可以根据需要选择不同的标签来使用特定版本的镜像。例如，如果你需要特定版本的 Ubuntu，你可以指定 `ubuntu:18.04`（对应 Ubuntu 18.04 LTS），而不是使用 `ubuntu:latest`，后者可能会给你最新但不一定是 LTS 版本的 Ubuntu 镜像。

使用dockerfile创建虚拟机:

```
touch Dockerfile
```

Dockerfile的文件内容如下:

```
FROM ubuntu:focal
RUN apt-get update
RUN apt-get install -y openssh-server
EXPOSE 22
RUN echo "OpenSSH server installed"
```

解释:

‘**FROM ubuntu:focal**’: 这行指定了基础镜像，即构建新镜像的起点。在这里，它使用了标记为 focal 的 Ubuntu 镜像，这代表 Ubuntu 20.04 LTS 版本。

‘**RUN apt-get install -y openssh-server**’: 这行使用 ‘**apt-get install**’ 命令安装了 openssh-server 软件包，即 SSH 服务器。-y 参数意味着在安装过程中自动回答 “yes” 于任何提示，确保无需人工干预即可完成安装。

‘**EXPOSE 22**’: 这行指示 Docker 在运行该镜像时打开端口22。SSH 服务默认监听在端口 22，所以这个设置允许通过 Docker 容器的 22 端口来访问 SSH 服务。

‘`RUN echo "OpenSSH server installed"`’：这行执行了 `echo` 命令，打印出 “OpenSSH server installed”。这行主要起到信息提示的作用，但实际上在构建镜像时这个输出不会显示给最终用户。

总的来说，这个 `Dockerfile` 的目的是构建一个包含 `OpenSSH` 服务的 `Ubuntu 20.04` 容器，暴露 `22` 端口以便能够通过 `SSH` 访问该容器。这样的容器可以用于远程管理、`SSH` 隧道等用途。

基于 `Dockerfile` 创建镜像：

```
docker build -t my-image2 .
```

该命令用于从一个 `Dockerfile` 构建一个新的 `Docker` 镜像。下面是命令各部分的解释：

‘`docker build`’：这是 `Docker` 中用于从 `Dockerfile` 创建镜像的命令。

‘`-t my-image2`’：`-t` 参数用于指定新建镜像的名字和（可选的）标签。在这里，镜像被命名为 `my-image2`。如果没有指定标签，则默认标签为 `latest`。

‘`.`’：最后的点 `.` 是这个命令的一个重要部分，它告诉 `Docker` 命令去当前目录（即点 `.` 所表示的目录）查找 `Dockerfile`。

这个命令的作用是，根据当前目录下的 `Dockerfile`，构建一个名为 `my-image2` 的新 `Docker` 镜像。这个过程包括执行 `Dockerfile` 中的所有指令，如安装软件包、设置环境变量等，最终创建出一个可用的镜像。

终端显示：

```
lzc@lzc-virtual-machine:~/docker$ docker build -t my-image2 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM ubuntu:focal
--> 3cfff1c6ff37e
Step 2/5 : RUN apt-get update
--> Running in 7a739f3fc0fa
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3452 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1191 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3364 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [29.7 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3517 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [3928 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1487 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [55.2 kB]
Get:18 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [28.6 kB]
Fetched 30.5 MB in 7s (4438 kB/s)
Reading package lists...
```

```
Removing intermediate container 4d3393b345d6
--> 2c984aceddb8
Successfully built 2c984aceddb8
Successfully tagged my-image2:latest
```

Docker 镜像 my-image2 构建成功！

4.6 比较 docker build 和 docker commit 创建镜像的区别

通过命令‘docker images’查看刚刚创建的镜像信息：

```
lzc@lzc-virtual-machine:~/docker$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-image2	latest	2c984aceddb8	8 minutes ago	248MB
my-image1	latest	8179e0a4bd59	13 hours ago	248MB
ubuntu	focal	3cfff1c6ff37e	3 weeks ago	72.8MB
hello-world	latest	d2c94e258dcb	10 months ago	13.3kB

其中镜像 my-image2 是刚刚我们使用 docker build 命令创建的镜像，镜像 my-image1 是我们使用 docker commit 命令创建的镜像。

当使用 docker build 和 docker commit 命令创建 Docker 镜像时，尽管它们可能基于同一个基础镜像，但这两个命令在创建镜像的方式和用途上有显著的区别：

1. 使用 docker build 创建镜像：

基于 Dockerfile：docker build 命令基于 Dockerfile 来创建镜像。Dockerfile 是一个文本文件，其中包含了一系列指令和命令，用于定义如何构建镜像。

可重复性：通过 Dockerfile，构建过程是自动化和可重复的。这意味着可以在不同环境中重复构建相同的镜像，而且每次都会得到一样的结果。

透明性和文档化：Dockerfile 提供了对镜像内容和构建过程的完全控制，并且作为镜像构建过程的文档。这对于了解镜像的构成和后续的维护非常有帮助。

适用场景：适用于创建预定义配置的镜像，如在开发过程中构建应用环境、自动化部署等。

2. 使用 docker commit 创建镜像：

从现有容器创建：docker commit 创建的镜像是基于一个已经运行并可能被修改过的容器。这个命令将容器的当前状态“快照”成一个新的镜像。

手动过程：与 docker build 相比，docker commit 是一个更加手动的过程。用户需要先对容器进行所需的更改，然后使用 commit 命令来创建一个新的镜像。

缺乏透明性：使用 `docker commit` 创建的镜像缺乏构建过程的透明性。除非有详细的记录，否则很难知道容器在 `commit` 之前进行了哪些更改。

适用场景：适用于快速捕捉容器状态的情况，如测试某些软件更改的结果，或者从现有容器状态创建一个基准镜像。

5 虚拟机和容器的启动速度比较

虚拟机（VMs）和容器（如 Docker 容器）在启动速度上有显著差异，主要原因在于它们各自的架构和工作方式的不同。

虚拟机（VM）启动速度：

虚拟机模拟整个硬件系统，包括CPU、内存、网络接口以及磁盘等。因此，启动虚拟机通常涉及启动完整的操作系统（OS）。这个过程包括加载和初始化内核、启动服务、以及执行各种系统级检查。由于这些因素，虚拟机通常需要更长的时间来启动，可能是几秒到几分钟不等，具体取决于VM的配置和宿主机的性能。

容器启动速度：

容器共享宿主机的操作系统内核，而不需要模拟硬件或启动自己的操作系统。容器通常仅封装应用程序和其依赖，这使得它们非常轻量级。因此，容器可以在几秒钟内快速启动，有时甚至在毫秒级别。

综上所述，容器在启动速度上通常比虚拟机快得多。这一特性使得容器非常适合于需要快速扩展、高弹性和频繁部署的应用场景。而虚拟机由于提供了更完整的隔离和模拟整个硬件系统的能力，更适用于需要运行完整操作系统或提供更强隔离的场景。

6 虚拟机和容器的镜像大小比较

虚拟机（VM）和容器的镜像大小通常有显著差异，这主要由它们各自的架构和包含内容决定。

虚拟机镜像大小：

虚拟机镜像包括完整的操作系统（OS）以及该 OS 上运行的所有应用程序和服务。因为它包含了一个完整的操作系统，所以虚拟机镜像通常较大，可能有几 GB 大小。VM 镜像的大小也受到安装在操作系统上的应用程序和服务数量的影响。


```
lzc@lzc-virtual-machine:~/3_4_vir_machine$ qemu-img info ubuntu.img
image: ubuntu.img
file format: qcow2
virtual size: 10 GiB (10737418240 bytes)
disk size: 4.86 GiB
cluster_size: 65536
Format specific information:
  compat: 1.1
  compression type: zlib
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
  extended l2: false

lzc@lzc-virtual-machine:~/3_4_vir_machine$ qemu-img info Ubuntu-raw.img
image: Ubuntu-raw.img
file format: raw
virtual size: 10 GiB (10737418240 bytes)
disk size: 4.45 GiB
```

可以看到，在上述实验中，我们创建的虚拟机磁盘镜像文件有几 GB 大小。

容器镜像大小：

容器镜像通常只包含应用程序及其直接依赖，不包含完整的操作系统。这些镜像共享宿主机的操作系统内核，因此不需要像虚拟机那样包含整个操作系统。容器镜像因此通常较小，可能只有几 MB 到几百 MB 大小，这取决于应用程序本身和其依赖的复杂性。

```
lzc@lzc-virtual-machine:~/docker$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-image2	latest	2c984aceddb8	8 minutes ago	248MB
my-image1	latest	8179e0a4bd59	13 hours ago	248MB
ubuntu	focal	3cff1c6ff37e	3 weeks ago	72.8MB
hello-world	latest	d2c94e258dcb	10 months ago	13.3kB

我们可以看到，通过上述命令操作创建的容器镜像的大小都很小，只有几 MB 到几百 MB 大小。

总体来说，容器镜像比虚拟机镜像小得多，因为它们只包含应用程序和必要的运行时环境，而不是整个操作系统。这种大小的差异是容器在资源利用率、启动速度和可伸缩性方面优于虚拟机的重要原因之一。

7 总结

通过本次实验，我们成功完成了虚拟化技术的深入实践，包括虚拟机的创建、管理，以及容器技术的应用。通过实验，我们不仅掌握了虚拟机和容器的创建与管理技能，还比较了它们在不同方面的性能差异。

首先，在虚拟机方面，通过使用QEMU和相关工具，我们详细了解了虚拟机的创建过程，包括磁盘镜像准备、操作系统安装、虚拟机配置与管理。我们还学习了如何使用XML文件管理虚拟机，以及如何通过图形化界面运行虚拟机，这些技能对于理解虚拟化技术的深层次应用非常重要。

在容器技术的学习中，通过安装Docker并使用Docker Hub上的ubuntu20.04镜像，我们实践了从镜像运行容器、使用`docker commit`命令创建新镜像，以及使用Dockerfile编译镜像的全过程。这一部分的实践加深了我们对容器技术的理解，特别是在容器镜像制作和管理方面的知识。

通过比较虚拟机与容器的启动速度，我们发现容器在启动速度上有明显优势，这主要得益于容器轻量级和共享宿主操作系统的特性。此外，我们也对比了虚拟机镜像格式（raw与qcow2）以及容器镜像创建方法（`docker build`与`docker commit`），深入理解了它们的优缺点和适用场景。

总的来说，本次实验不仅让我们掌握了虚拟机和容器的操作技能，而且通过实践比较，更加深刻地理解了两者在云计算应用中的性能差异和适用场景。这对于我们未来在云计算、虚拟化领域的学习和研究具有重要意义。