

2022/3/24 杨澜

注：1.5 Dockerfile 和 2 VS Code 或 VS 中配置 Docker 以及 3 一种github文件夹构造介绍 比较有用，别的就是一些理解方面的，可以不看。建议最好看完这三个部分再从 github 拉取文件夹构造本地环境。

Docker

1.1 Docker是什么

1.1.1 容器是什么

有效的将单个操作系统的资源划分到孤立的组中，以便更好的在孤立的组之间平衡有冲突的资源使用需求。

1.1.2 Dockers的原理

Docker 可以在一台物理机上隔离出多个类似虚拟机的环境。相比于虚拟机，Docker 并没有虚拟化CPU，内存等资源，而是基于内核的机制来做进程隔离。

1.1.3 Docker和虚拟机实现原理的区别

虚拟机的 Guest OS 层（虚拟机中安装的操作系统）和 Hypervisor 层在 Docker 中被 Docker Engine 层所替代。虚拟机实现资源隔离的方法是用 Hypervisor 虚拟化 CPU、内存、IO 设备等实现的，类似于在实际物理机上模拟一个物理机。

1.1.4 Docker和虚拟机相比的优点

简单来讲就是虚拟机隔离级别更高所以没有 Docker 轻量级。

1. Docker 有着更少的抽象层，不需要虚拟实现硬件资源、使用的是实际的物理机资源，在资源利用率上更高。
2. Docker 直接使用 Host OS (宿主操作系统，一般是 Linux)的内核然后分配相应的资源，新建容器时不需要重新加载一个 Guest OS，加载速度更快，镜像更小。

1.1.5 Docker需要 Linux 作为宿主操作系统为什么能在别的系统下使用

Windows 提供2种类型的容器运行时模型 Windows Server 和 Hyper-V

1. Hyper-V (Windows)

Hyper-V 中的容器仅由 Docker 管理，Hyper-V 中的虚拟机则用 Hyper-V Manager 等传统工具管理。启动 Hyper-V 容器需要比 Windows Server Containers 更长的时间，但两者都比具有完整操作系统的 VM 快。

Hyper-V 原理相当于在底层系统上运行一个 Linux 虚拟机，然后 docker 的所有容器都运行在这个虚拟机上。虽然也建了虚拟机，但性能还是比 VM 好，而且更方便，可以和 Windows Terminal、VS Code 等软件联动使用。

2. Windows Server (Windows)

（了解一下就行，这个一般用不到）Windows Server Container 与 Linux Container 容器模型一致，容器和底层操作系统共享宿主系统内核，但是 Windows Container 只能运行 Windows 应用程序。

Mac（待补充）

1.2 什么是镜像 (Image)

1.2.1 镜像的作用

Docker 镜像类似于创建虚拟机时的 *ISO* 镜像。*Docker* 的镜像一般比虚拟机的更小。

1.2.2 如何获取镜像

可以通过以下格式的指令获取 *Docker* 镜像

```
# 指令格式
docker pull [选项] [Docker Registry 地址[:端口号]/]仓库名[:标签]

# 指令使用示例，直接从 DockerHub 获取版本为18.04的ubuntu
docker pull ubuntu:18.04
```

DockerHub是 *Docker* 官方维护的一个公共仓库，可以在里面上传和拉取镜像。就像 *GitHub* 里 `pull` 和 `push` 代码那样。

1.2.3 如何订制镜像

可以通过以下格式指令保存对一个镜像作出的修改

```
# 指令格式
docker commit [选项] <容器ID或容器名> [<仓库名>[:<标签>]]

# 指令使用示例，保存 ID 为 f19fe4eb216f 的容器为镜像
docker commit f19fe4eb216f
```

一般不建议使用 `commit` 而建议使用 *Dockerfile* 文件订制一个镜像。因为通过 `commit` 指令生成的镜像，除了镜像制作者其他人无从得知执行过什么命令、怎么生成的镜像，不利于镜像的维护工作。在1.5中介绍了 *Dockerfile* 的使用方法。

1.2.4 如何删除镜像

可以通过以下格式指令删除镜像

```
# 指令格式
docker image rm [选项] <镜像1> [<镜像2> ...]

# 指令使用示例，删除 ID 为 aaeee51e0c8a 镜像
docker image rm aaeee51e0c8a
```

1.2.5 镜像的更多具体操作

详见附录 1 中的参考资料“1. *Docker* 从入门到实践”中“使用镜像”一栏

1.3 什么是容器 (Container)

1.3.1 容器的作用

Docker 容器类似于虚拟机里创建的系统。如果把镜像比作类，容器就是这个类的实例化。

1.3.2 的具体操作

```
# 以下指令中的容器id都可以换为容器name

# 启动一个容器
docker start container_id

# 进入一个容器
docker attach container_id

# 停止一个容器
docker stop container_id

#停止所有容器
docker stop $(docker ps -a)

# 重启一个容器
docker restart container_id

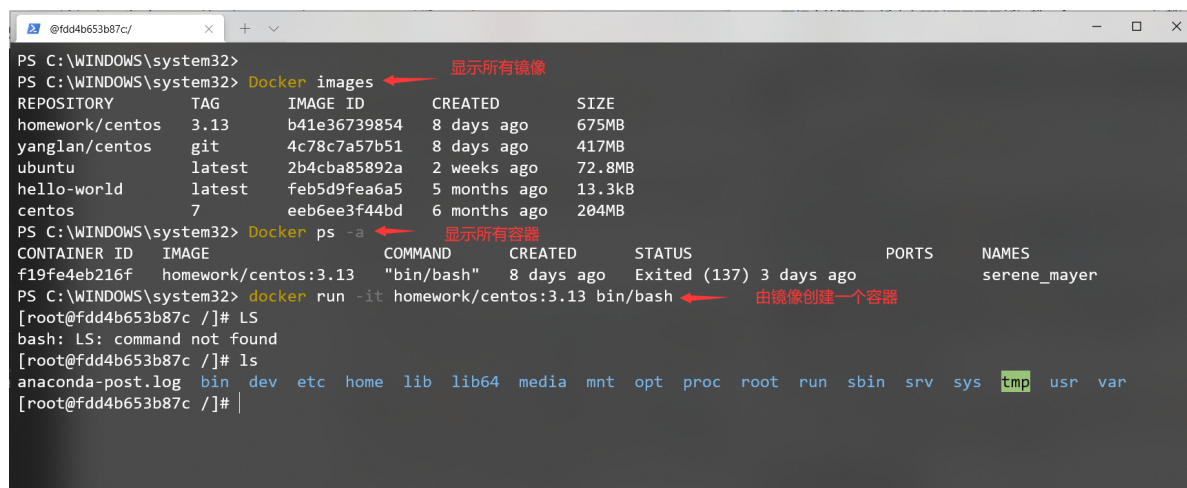
# 删除一个容器，删除前需要先停止
docker rm container_id
```

详见附录 1 中的参考资料“1. Docker 从入门到实践”中使用“操作容器”一栏。

1.4 Docker有什么用

Docker 可以理解为轻量级的虚拟机，可以直接通过指令快速进入容器运行匹配不同环境的程序。

下图使用了 Windows Terminal（用 CMD 也可以）。可以看出通过 Docker 可以直接从 Windows 的终端进入一个 CentOS 系统的终端。



```
PS C:\WINDOWS\system32>
PS C:\WINDOWS\system32> Docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
homework/centos      3.13            b41e36739854   8 days ago     675MB
yanglan/centos       git             4c78c7a57b51   8 days ago     417MB
ubuntu               latest          2b4cba85892a   2 weeks ago    72.8MB
hello-world          latest          feb5d9fea6a5   5 months ago   13.3kB
centos                7              eeb6ee3f44bd   6 months ago   204MB

PS C:\WINDOWS\system32> Docker ps -a
CONTAINER ID        IMAGE               COMMAND          CREATED        STATUS          PORTS          NAMES
f19fe4eb216f        homework/centos:3.13 "bin/bash"      8 days ago    Exited (137)    3 days ago    serene_mayer

PS C:\WINDOWS\system32> docker run -it homework/centos:3.13 bin/bash
[root@fdd4b653b87c /]# LS
bash: LS: command not found
[root@fdd4b653b87c /]# ls
anaconda-post.log  bin  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@fdd4b653b87c /]#
```

注：

1. `docker ps -a` 显示所有容器，`docker ps` 显示所有正在运行的容器。
2. 创建一个容器并不是进入一个已有的容器，而是由镜像（image）创建一个新容器运行。与创建容器指令相对的，`docker attach 容器ID` 命令可以进入一个已有的容器，但是attach进入容器后，如果从容器退出会导致容器停止。如果使用 `docker exec -it 容器ID /bin/bash` 命令行进入一个已有的容器，从容器退出后，容器不会停止。

1.5 Dockerfile**

1.5.1 Dockerfile的作用

Dockerfile 是一个用来构建镜像的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

1.5.2 Dockerfile实例

Dockerfile 的一个例子如下：

```
# 选择一个镜像作为基础,centos8停止更新了,所以用centos7。
# 之前试过ubuntu,但是ubuntu需要更新apt-get,如果不换源就更新速度奇慢,而且换源比较麻烦,懒得搞了。
FROM centos:7

# 维护者的信息
LABEL maintainer="YangLan 1356298972@qq.com"

# 启动镜像后进行的操作,这里是更新yum,并依次安装软件。
RUN yum update -y
# 下面两行是更新版本,centos:7 install默认下载的gcc版本是 4.8.5 只支持到c++11,但是应该够用了,因此这个镜像不更新gcc版本
# RUN yum install centos-release-scl -y
# RUN yum install devtoolset-9-gcc devtoolset-9-gcc-c++
# 下载gcc、g++
RUN yum install -y gcc
RUN yum install -y gcc-c++
# 下载clang
RUN yum install -y epel-release
RUN yum install -y clang
#安装Make
RUN yum -y install gcc automake autoconf libtool make
# 安装gdb
RUN yum install -y gdb
# 安装flex bison
RUN yum install -y flex
RUN yum install -y bison
# 安装 LLVM/Clang 二进制文件
RUN yum install llvm clang

# 安装系统管理指令,为CMake做准备
# RUN yum install sudo -y
# RUN yum install wget -y

# 安装CMake,centos7因为版本比较老所以要下载源码并编译。
# 用了docker大家的环境都一样所以应该不需要Cmake,make就足够了
# RUN yum -y install gcc gcc-c++ openssl openssl-devel tar make
# RUN wget https://github.com/Kitware/CMake/releases/download/v3.17.0/cmake-3.17.0.tar.gz
# RUN tar -zxf cmake-3.17.0.tar.gz
# RUN cd cmake-3.17.0
# RUN ./bootstrap --prefix=/usr --datadir=share/cmake --docdir=doc/cmake && make
# RUN sudo make install

# centos默认下载的git版本是1.8,古早版本现在应该已经没法用了。
# 我感觉在容器里只调试代码就可以了,用容器里更新到github也不太必要,可以在容器里调试完了出容器再更新到github
# RUN yum install -y git
```

```
# 拷贝本地文件到镜像中，前面是容器中的路径，后面是本地路径。
# 感觉不太必要，vscode可以直接选在文件夹在docker中打开
# COPY ./* D:\学习\大三下学期\编译原理与技术课程设计\for_git
```

在系统终端进入文件所在文件夹并运行如下命令行，即创建出一个名字为 `xxx/ubuntu:someversion` 的镜像。该命令行最后的点表示在当前目录寻找 *Dockerfile*

建议生成镜像时关闭 VPN 或者将 VPN 切换为直连模式，因为 VPN 有可能会和 WSL2 SSL 接口冲突。如果镜像生成速度很慢或者失败，可以 Ctrl+C 退出生成然后关闭 VPN 再试一次。

```
docker build -t="xxx/ubuntu:someversion" .
```

接下来一条指令表示根据该镜像生成一个容器。-it 使用交互模式运行容器并且分配一个伪输入终端，并绑定到容器的标准输入上；/bin/bash 是启动容器后运行的指令，这里是启动一个 bash 终端允许用户交互。

```
docker run -it xxx/ubuntu:someversion bin/bash
```

1.6 Docker Compose

Docker Compose 用于配置多个 container 并且将其同时运行。允许用户通过一个单独的 `docker-compose.yml` 模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目。**本次作业应该用不到这个功能，因为只需要一个容器，不需要多个。**

2 VS Code 或 VS 中配置 Docker

2.1 如何在 VS Code 中配置 Docker

2.1.1 初次打开容器

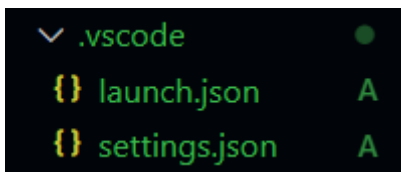
1. 在 VS Code 中安装 Docker 和 Remote - Containers 扩展
2. 将 Dockerfile 放在准备用来存放代码的文件夹中（最好不要在github的本地储存库存放调试的代码建立容器，避免污染 github 仓库的时间线）。
3. 点击**左下角**蓝色或者绿色的箭头部分。
4. 在 VS Code 顶端中选取 Remote - Containers : Open Folder in Containers 选项，选取在容器中打开的本地文件夹，然后选取 From 'Dockerfile' 选项。
5. 等待建立镜像和容器，建立完毕后该文件夹即在容器中打开。成功标志是左下角双箭头处显示“Dev Container:Existing Dockerfile”。
6. **请注意第五步时最好把梯子换成直连模式或者关掉**，梯子的接口冲突会导致没法拉取centOS镜像和安装软件。

Dockerfile 的具体内容请参考 1.5.2 Dockerfile实例

想要再次打开容器时 可以 Open Folder in Containers 然后选文件夹，也可以在打开文件夹的前提下选择 Reopen In Containers 选项。选择的文件夹最好是自己之前建立容器的文件夹，因为建立容器后文件夹中已经有了将文件夹和容器连接在一起的配置文件 `.devcontainer`。

2.1.3 调试代码

1. 调试的文件夹中需要一个 .vscode 文件夹，其中存放 launch.json 和 settings.json 文件



2. 配置 launch.json 文件: <https://geek-docs.com/vscode/vscode-tutorials/vscode-code-debugger.html#launchjson> 这个文件的作用是配置调试环境。
3. 运行代码时使用makefile对不同的程序进行连接。

2.1.3 github的文件同步

每次进入容器前用 git 指令拉取代码。

更新代码后可以按右下角的箭头选择 Open Folder locally 退出容器，然后在 VSCode 下方的终端栏中利用 git 指令将文件夹传入 github 。

2.2 如何在 VS 中配置 Docker

3 一种github文件夹构造介绍

3.1 如何从 github 拉取仓库、上传文件到仓库

1. 在本地选一个作为本地仓库的文件夹，在文件夹里打开git bash（**因为只有一个分支，建议不要在这个文件夹里调试代码避免污染时间线**）
2. 拉取远程代码（这一步之前可能需要配公钥私钥，网上能找到怎么配）

```
git clone https://github.com/.../xxx.git
```

3. 将远程分支和本地分支关联并拉取到本地（我们这个仓库就只有一个分支，所以直接拉master分支就可以）

```
git pull origin master
```

4. 关联好之后以后再拉取只需要 `git pull`
5. 修改完本地仓库中的代码后上传到远程分支

1. 添加该目录下所有文件至本地仓库，点表示所有文件 `git add .`
2. 提交并添加注释 `git commit -m "这里是注释"`
3. 上传到 github `git push`

3.3 一种 github 文件夹构造介绍

名称	修改日期	类型
.git	2022/3/24 16:18	文件夹
Code	2022/3/24 17:41	文件夹
周报	2022/3/24 16:18	文件夹
README.md	2022/3/8 12:38	Markdown File
编译原理课程设计技术栈.md	2022/3/24 16:18	Markdown File
开发记录.md	2022/3/24 16:18	Markdown File

希望我们的仓库按照如上图的格式放置文件。

Code 文件夹中放置代码，Code下有 n 个文件夹，分别代表编译器的 n 个步骤，第 i 个步骤文件夹中放置第 i 步的代码。除这些步骤文件夹之外还有一个Dockerfile文件用来建立容器。

1LexicalAnalysis	2022/3/24 19:14	文件夹
2SyntaxAnalysis	2022/3/24 19:14	文件夹
3SemanticAnalysis	2022/3/24 19:15	文件夹
4Optimization	2022/3/24 19:16	文件夹
5CodeGeneration	2022/3/24 19:17	文件夹
Dockerfile	2022/3/24 18:56	文件

这些步骤文件夹中放置.cpp文件和makefile文件，以及一个README.md文件对代码和更新进行介绍。

因为合并分支很麻烦，所以建议仓库只有master一个分支。

建议大家不要在拉取 github 仓库的文件夹中直接建立容器调试代码，而是拉取代码后将代码复制到准备好的调试代码的文件夹中，调试完毕后复制回仓库文件夹按照 3.1 的步骤上传代码，避免把不必要的文件上传到master分支里污染时间线。

每次大家在容器里写完代码，退出容器后在windows上把代码更新到github上。每个人写自己部分的代码之前把相应的上个步骤的文件夹的代码复制下来，这样makefile不需要经常改，然后也能实现互相功能的同步。

4 Windows Terminal

Windows 上比较好用的命令行终端。

CMD 和 PowerShell 的区别: PowerShell 是 CMD 的升级版或者说超集。

Windows Terminal 继承了PowerShell，可以看作画面更易读美观的 CMD。

这个东西不是项目必须的。如果想用可以直接找相关的安装资料，不需要太复杂的设置。

附录：参考资料

1. Docker 从入门到实践 https://yeasy.gitbook.io/docker_practice/
2. 求求你了，用 Docker 吧 https://juejin.cn/post/7021006271818137630?share_token=878fa440-84d9-4d8a-980e-aaba73440daf
3. 菜鸟教程：Docker 教程 <https://www.runoob.com/docker/>

4. VSCode 调试 *Docker* 中的程序(C++) 和离线安装 VSCode 插件的方法 <https://blog.csdn.net/xianxjm/article/details/112250699>
5. VSCode: 将代码提交到 *github* https://blog.csdn.net/weixin_38750084/article/details/89604906
- 6.