Day13:Docker

Docker的基本组成

三个组成部分

我们认为docker的主要组成是镜像(image), 容器(container), 仓库(repository)这三部分

image就像是大家在C++里学到的类模板，container就是大家通过写好的类模板 new 出来的对象，repository则是大家写好上传镜像的地方，就像是github，其中有一些对我们而言比较重要的镜像如ubuntu等。

镜像(image)

image就是一个只读的模板，他可以用来创建docker容器(container)，而且一个镜像可以创建很多容器。有时他也是一个root的文件系统，如官方的ubuntu镜像就包含了相关版本的最小系统的root文件系统。有点像于C++中的类模板。

容器(container)

Docker利用容器独立的运行应用，每一个容器都是用镜像创建的运行示例，有点像我们new出来的C++对象。他是一个虚拟的环境，也算是一个简易版的linux系统（包括root权限，进程空间，用户空间和网络空间）为我们需要运行的东西提供支持，他们可以被启动，开始运行，终止运行，删除等。容器与容器之间相互独立，互不干扰，保证安全。

仓库(repository)

用以集中存放image文件的地方。

仓库分为公开仓库(public)和私有仓库(private)，我们这里只学习公开仓库.

最大的公开仓库是docker hub:

```
https://hub.docker.com/
```

(演示)

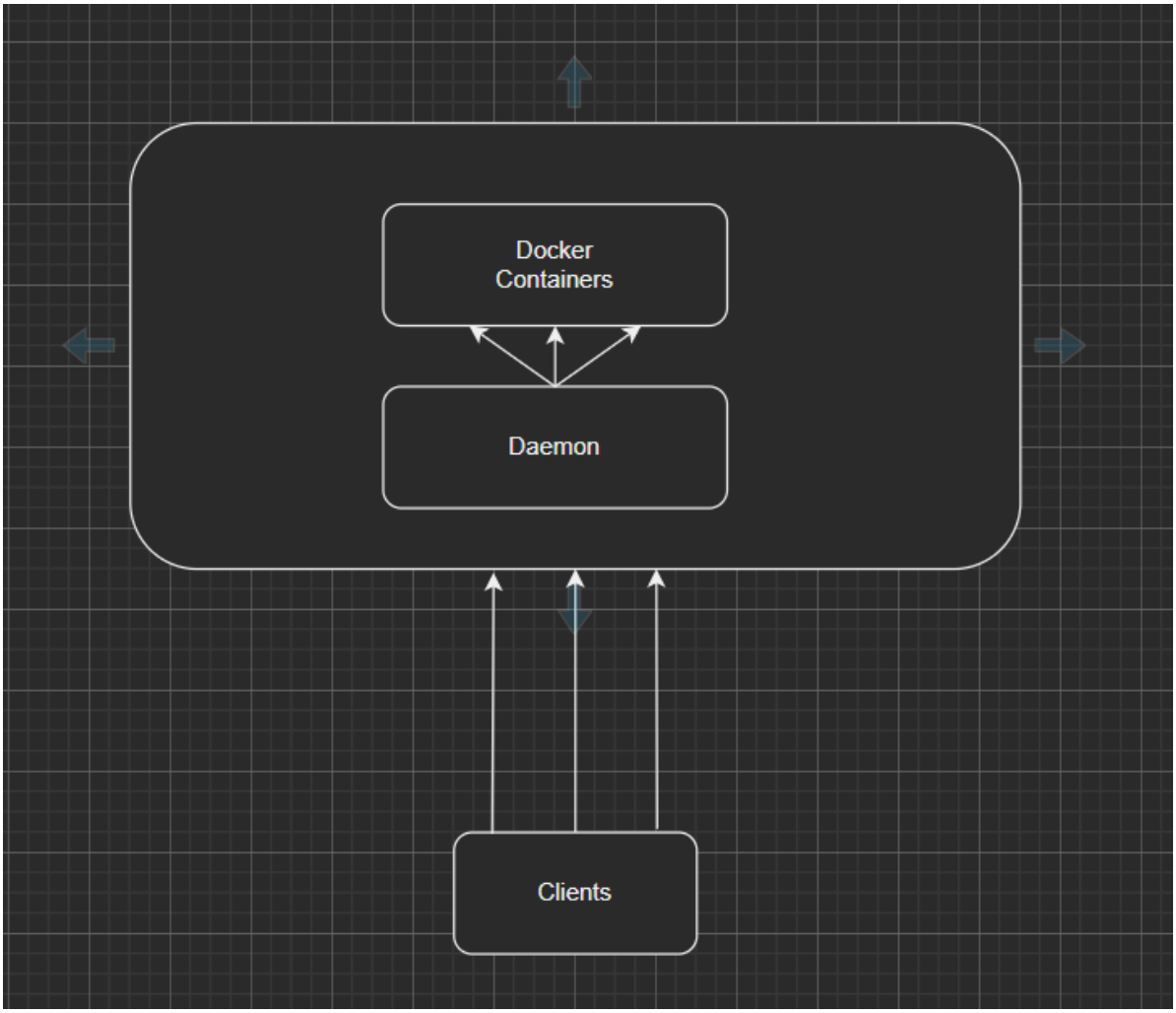
需要的时候我们直接pull就可以了。

Docker的结构

Docker的运行结构

Docker是一个Client-Server结构的系统，他的守护进程

运行在主机上，通过socket链接从client访问，Daemon从client接受命令并管理主机上的容器。



Docker的基本命令

docker build, docker pull, docker run

我们的client将这三种命令发至daemon时，docker的处理方式如下：

docker build 是将我们的环境源码依赖等打包成一个镜像并存放在本地，docker pull 是指从docker hub 上拉取镜像并存在本地，docker run 则是如果在本地找到了相关镜像，那么就直接启动，如果本地没有，就到docker hub上载image并启动。这是docker最基本的命令。

一个例子Helloworld

```
li-yr20@DESKTOP-BOCQUL9:~$ sudo docker run hello-world
[sudo] password for li-yr20:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:2498fce14358aa50ead0cc6c19990fc6ff866ce72aeb5546e1d59caac3d0d60f
Status: Downloaded newer image for hello-world:latest

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/

li-yr20@DESKTOP-BOCQUL9:~$
```

可以看到首先是Unable to find image 'hello-world:latest' locally

说明本地没有helloworld image，故docker会先从docker hub上拉取改image，在此基础上以改image为基础启动容器，开始运行。

Docker 常用命令

启动类命令

（对于win以及相应的wsl系统，docker会随着开机自动开启)

docker info:查看docker的信息如资源占用，版本等

docker --help(直接输入docker也可以)：获得docker的所以命令以及命令的参数与功能。

docker 具体命令 --help(可以获得更详细的命令信息)

镜像命令（image)!

docker images

基本：docker images

```
docker images
```

可以给出所有的本地镜像（包括仓库源，版本 ,ID, 创建时间，大小)

参数:

-a: 列出所有本地镜像，含历史映像层(后面会讲)

-q: 只显示镜像ID

docker search

```
docker search ubuntu
```

基本：搜索我们的远程仓库是否有相应的image（演示）

参数	说明
NAME	镜像名
DESCRIPTION	镜像说明
STARS	点赞数量
OFFICIAL	是否官方
AUTOMATED	是否是自动构建的

（如果有official，我们一般选择official)

参数：

--limit ： 只列出N个镜像，默认25个(按照stars数排序)

```
docker search --limit 5 ubuntu
```

docker pull

基本：拉取镜像

(不加TAG默认latest)

docker pull 镜像名[:TAG]

```
docker pull ubuntu:latest
docker pull ubuntu
```

docker system df

查看docker中images/containers/local volumes（数据卷）占用的空间

```
docker system df
```

docker rmi (image ID or name)

```
docker rmi hello-world
docker rmi feb5d9fea6a5
```

基本：删除镜像。

参数：

-f: 有时image正在被停止的容器使用，无法删除，那么此时-f就可以强制删除。

```
docker rmi -f feb5d9fea6a5
```

可以同时删除多个

```
docker rmi -f feb5d9fea6a5 ba6acccedd29
```

删除全部(有点像sql)

```
docker rmi -f $(docker images -qa)
```

容器命令(采用ubuntu演示)

docker run

```
docker run [options] IMAGE [command] [age]
```

options	功能
--name="the name"	指定容器名字
-d	后台运行容器并返回ID，守护式容器
-i	交互式运行容器（常与t结合使用）
-t	为容器分配一个伪输入终端（常与i结合使用）
-P	随机端口映射
-p	指定端口映射

端口映射，ubuntu的端口与主机的端口进行映射

启动交互式容器：(必须演示)

```
docker run ubuntu
docker run -it ubuntu /bin/bash
exit
```

以交互式启动

docker ps

```
docker ps [OPTIONS]
```

显示正在运行的容器。

参数：

-a: 列出所有当前正在运行的+历史上运行过的

-l: 显示最近创建的容器

-n: 显示最近n个创建的容器

-q: 仅显示容器号

docker exit

进入容器后

```
exit
退出并停止容器
ctrl + p + q
退出但不停止
```

docker 其他控制命令

```
docker start (ID or name)
docker restart (ID or name)
docker stop (ID or name)
docker kill (ID or name)
```

docker rm

```
docker rm [options](ID or name)
```

只能删除已停止的

```
docker rm -f (ID or name)
```

进出操作

我们的docker通过images创造的containers不能用一次就删一次，我们希望容器可以长时间的存在以供我们长期使用，那样就需要我们进行进出容器的操作。

```
docker run -d
```

对于一些容器，我们不需要和他进行交互，就可以使用-d参数让容器在后台运行。

但是，在这种情况下，docker容器在后台运行时必须有一个前台进程，如果我们的容器运行的不是那种一直挂起的命令是会自动退出的。所以一般情况下我们还是采用交互式容器，防止自动退出。

查看容器日志

```
docker logs ID
```

查看容器内运行的进程

```
docker top ID
```

查看容器内部

```
docker inspect ID
```

进入容器

```
docker exec -it ID(在容器中打开一个新的终端，exit不会导致容器停止)
docker attach ID(直接进入容器的启动命令的终端，exit会导致容器停止)
```

推荐exec

容器与主机

docker cp

```
docker cp 容器ID:容器路径 主机路径
```

复制容器中的文件到主机上

docker export

```
docker export 容器ID >文件名.tar
```

导出容器内部的内容作为tar归档文件（相当于导出了整个容器，备份了整个容器)

docker import

```
cat 文件名.tar | docker import - 镜像用户(可选，类似java的包)/镜像名:镜像版本号
```

将export的容器恢复

Docker镜像的分层

docker的镜像是可以由很多层构成的

UnionFS(联合文件系统):Union文件系统（UnionFS)是一种分层、轻量级且高性能的文件系统，他支持对文件系统的修改作为一次提交来一层层叠加，同时可以将不同目录挂载到同一个虚拟文件系统下。UnionFS是Docker镜像的基础。镜像可以通过分层来进行继承，基于基础镜像可以制作各种具体的应用场景。

特性：一次加载多个文件系统，但从外面看来只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录。

Docker镜像的加载原理

docker的镜像实际上就是由UnionFS构成的，刚开始时会加载botfs文件系统，实现linux内核与boot加载器，boot加载完后整个内核都在内存中了，此时内存的使用权就会由bootfs转交给linux内核，而后卸载bootfs。之后再bootfs的基础上加载rootfs，包含了linux系统中的/dev /proc / etc等标准目录。rootfs是各种不同版本操作系统的发行版，如Ubuntu，Centos等。

（为什么这么小。因为只有bootfs和rootfs，只有最基本的命令)

为什么要这样

镜像分层后可以共享资源，且方便复用

Docker中分为镜像层与容器层，镜像层都是可读的，容器层才是可写的，当容器启动时，一个新的可写层被加载到镜像的顶部，这就是容器层，“容器层”之下都叫“镜像层”。

Docker commit

作用

加强镜像(演示)

以ubuntu为例，如果我们直接从docker hub中下载了官方的ubuntu，会发现它少了很多命令如vim等。如果我们想加强功能，那么我们就需要commit加强镜像

我们直接在容器中下载vim

```
apt-get update
apt-get -y install vim
```

此时我们的容器拥有了vim功能，现在我们的容器变强了。那么我们现在就要提交我们的容器把容器变成镜像

```
docker commit -m="add vim" -a=(“作者”) (容器ID!!) 镜像用户(可选，类似java的包)/镜像名:镜像版本号
docker commit -m="add vim" -a="li-yr20" ef647f03ca34 eesast/myubuntu:1.0
```

现在它成为了一个新的镜像，且该镜像自动具有vim功能，相当于在原ubuntu中增加了一层。

我们可以通过继承原有的ubuntu镜像，为他增加新的镜像层从而实现更高级的功能。有点类似父类与继承。

Dockerfile

简介

Dockerfile是用来构建Docker镜像的文本文件，是由一条条构建镜像所需的指令和参数构成的脚本。

我们一次次commit来增强我们的image比较麻烦，Dockerfile可以轻松的解决这一问题，在一个file中快速增加镜像的层数，增加镜像的功能。

官网查询：

（打开并演示之后得内容，结合THUAI5 Dockerfile）

```
https://docs.docker.com/engine/reference/builder/
```

大致流程：

编写Dockerfile---docker build命令构建镜像---docker run以镜像为基础构建容器实例

基础知识

每条保留字指令都必须是大写字母且后面要跟随至少一个参数

每条指令从上到下，顺序执行

#表示注释

每条指令都会创建一个新的镜像层并对镜像进行提交。

Docker执行Dockerfile

docker从我们得基础镜像运行一个容器

执行一条指令并对容器做出修改

执行类似docker commit得操作。提交生成一个新的镜像层

docker再基于刚commit得镜像运行一个新容器

执行下一条指令，重复操作知道所有指令都执行完成。

小总结

Dockerfile -->docker image -->docker container

保留字介绍

From

基本是第一行，说明我们的基础镜像是什么，eesast的基础镜像一般是ubuntu，不过版本可能有所不同

MAINTAINER

说明镜像的作者及维护者，以及他们的联系方式

RUN

容器构建后执行相关的命令，可以是命令行形式，也可以是exec形式，命令行如如我们之前的例子安装vim

```
RUN apt-get -y install vim
```

exec:

```
RUN ["可执行文件","参数1","参数2"]
RUN "可执行文件" "参数1" "参数2"
```

EXPOSE

当前容器所暴露的端口

WORKDIR

指定容器创建后，终端默认登录进来的工作目录，一个落脚点

也就是说这条指令指定我们进入容器打开终端，终端所在的目录。如果默认则与USER相对应

USER

指定该镜像以什么样的用户去执行，若不指定，默认是root

ENV

运行时环境，用来在构建镜像的过程中设置我们的环境变量

```
ENV RUNNING_PATH /usr/eesast
```

那么此时RUNNING_PATH就代表此目录，我们可以直接

```
WORKDIR $RUNNING_PATH
```

默认位置就是RUNNIG_PATH - /usr/eesast

COPY

从宿主机中copy文件到镜像中（类似docker cp)

```
COPY src dest
COPY ["src","dest"]
```

ADD

将宿主机下的文件copy进镜像中，且会自动处理URL和解压TAR压缩包（相对于COPY增加了解压功能）

CMD

指定容器启动后要干的事（格式类似RUN)，一般是最后一行

```
CMD shell
CMD ["可执行文件","参数1","参数2"]
CMD "可执行文件" "参数1" "参数2"
```

可以写多个CMD，但只有最后一个会生效。

但如果我们在启动container的时候增加了命令

```
docker run 参数
```

可能会覆盖CMD的内容，导致CMD不执行

ENTRYPOINT

也是用来指定一个容器启动时要运行的命令，类似CMD指令，但是ENTRYPOINT不会被docker run后的命令覆盖，而且那些命令会被当做参数送给ENTRYPOINT指令指定的程序。

```
ENTRYPOINT ["<EX指令>","参数1","参数2"....]
```

ENTRYPOINT可以和CMD一起用，ENTRYPOINT ""，此时的CMD不在直接运行将CMD的内容作为参数传递给ENTRYPOINT，让ENTRYPOINT执行。