Docker概述

类比开发安卓：

Java -- apk -- 发布（应用商店） -- 张三使用apk -- 下载安卓应用！

Java -- jar（环境） --- 打包项目带上环境（镜像） --- （Docker仓库：商店） --- 下载我们发布的镜像

（根本无需考虑环境问题，因为镜像中集成了环境）

Docker的思想来自于集装箱！

Docker通过隔离机制，可以将服务器利用到极致。

传统：JRE -- 多个应用（端口冲突）-- 原来都是交叉的

隔离：Docker核心思想！打包装箱！每个箱子都是互相隔离的

Docker历史

在容器技术出来之前，我们都是使用虚拟机技术！

虚拟机：在window中装一个Vmware，通过这个软件我们可以虚拟出来一台或者多台电脑！笨重！

容器：

虚拟机属于虚拟化技术，Docker容器技术，也是一种虚拟化技术！

```
vm: linux centos 原生镜像（相当于一台电脑）如果想要隔离，需要开启多个虚拟机！
docker: 如果要隔离，镜像（最核心的环境只有4mb，放一些命令和开机启动等）
按需求制作自己的镜像：核心环境 4MB + jdk + mysql 按需求打包，十分轻巧
虚拟机动辄几个G，非常笨重；启动需要几分钟
容器最小有KB级别，一般几M或者几十MB，几百MB已经非常大了；容器秒级启动
```

Docker是基于GO语言开发的

官网：www.docker.com

文档地址：https://docs.docker.com

仓库地址：https://hub.docker.com 类似github，可以将自己开发的镜像发布上去，也可以下载

Docker能干嘛

容器化技术不是模拟的一个完整的操作系统

容器之间是互相隔离的，每个容器都有运行环境和APP，容器直接运行在操作系统之上，可以充分的利用操作系统的资源

比较Docker和虚拟机技术的不同：

传统虚拟机，虚拟出一条硬件，运行一个完整的操作系统，然后在这个系统上安装和运行软件。

容器内的应用直接运行在宿主机的内核上，容器是没有自己的内核的，也没有虚拟硬件，所以就轻便了

每个容器间是互相隔离的，每个容器内都有一个属于自己的文件系统，互不影响

> DevOps（开发、运维）

应用更快速的交付和部署

传统：一堆帮助文档，安装程序

Docker：打包镜像发布测试，一键运行

更便捷的升级和扩容

使用了Docker之后，我们部署应用就和搭积木一样

项目打包为一个镜像，扩展服务器

更简单的系统运维

在容器化之后，我们的开发，测试环境都是高度一致的

更高效的计算资源利用

Docker是内核级别的虚拟化，可以在一个物理机上运行很多的容器实例！服务器的性能可以体现到极致

Docker安装

基本组成

镜像（image）

docker镜像就好比是一个模板，可以通过这个模板来创建容器服务

tomcat镜像 ==> run ==> tomcat01容器（提供服务）通过这个镜像可以创建多个容器（最终服务运行或者项目运行就是在容器中的）

容器（container）

Docker利用容器技术，独立运行一个或者一个组应用，通过镜像来创建的

启动，停止，删除，基本命令！

目前就可以把这个容器理解为就是一个简易的linux系统

仓库（repository）

仓库就是存放镜像的地方！

仓库分为共有仓库和私有仓库！

官方：Docker Hub（默认是国外的）

阿里云：都有容器服务（配置镜像加速）

环境准备

CentOS 7

Xshell

```
# 系统内核是 3.10 以上的
[root@localhost ~]# uname -r
3.10.0-1127.el7.x86_64
```

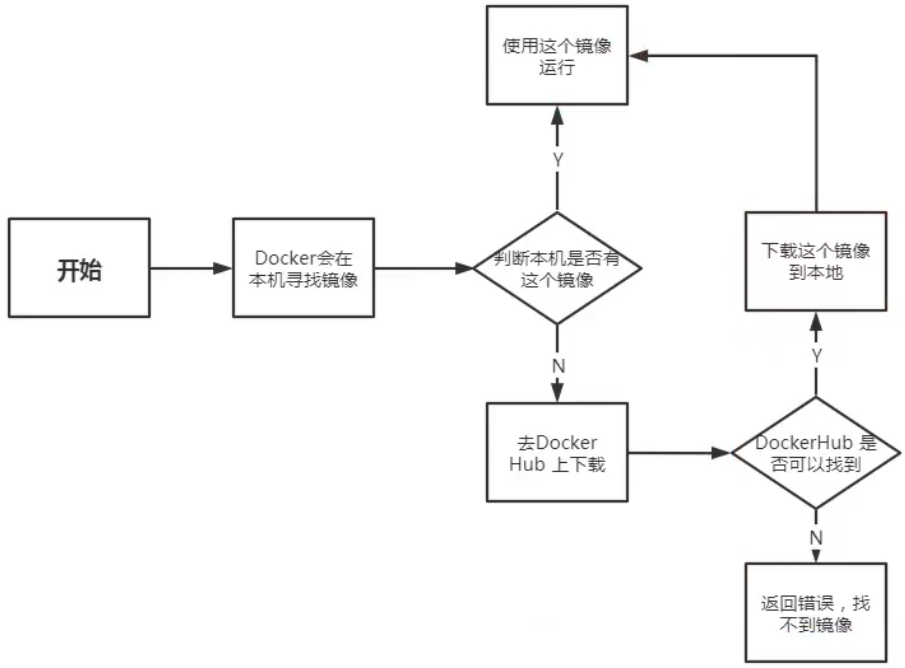
```
# 系统版本
[root@localhost ~]# cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

安装

```
#1、卸载旧版本
yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
#2、需要的安装包
yum install -y yum-utils

#3、设置镜像的仓库
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo # 默认是国外的

yum-config-manager \
    --add-repo \
    http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo # 建议安装阿里
云的

# 安装之前更新yum软件包索引
yum makecache fast

#4、安装docker相关的内容
# 安装：核心-客户端-容器
# docker-ce 社区
# docker-ee 企业版
yum install docker-ce docker-ce-cli containerd.io
```

```
#5、启动docker
systemctl start docker

#6、测试是否安装成功
docker version

#7、运行HelloWorld
docker run hello-world

#8、查看一下下载的 hello-world 镜像
docker images

#9、卸载docker
# 卸载依赖
yum remove docker-ce docker-ce-cli containerd.io
# 删除资源
rm -rf /var/lib/docker
rm -rf /var/lib/containerd
```

run流程

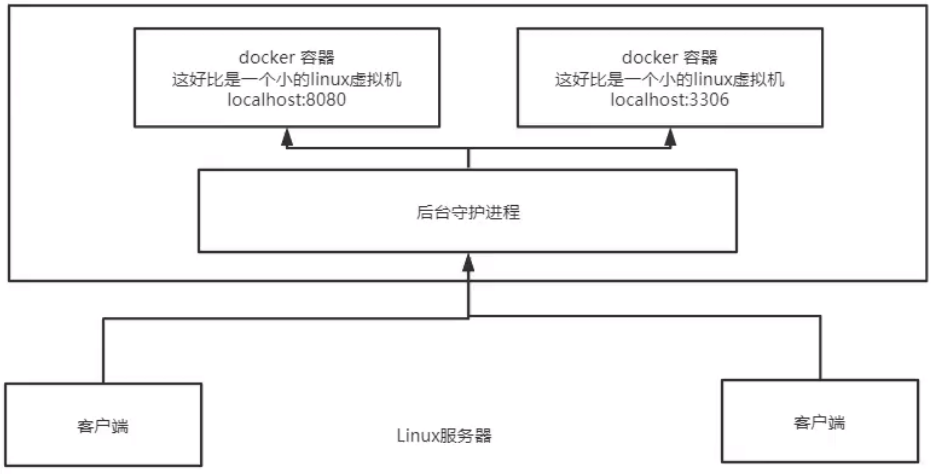


底层原理

docker是怎么工作的

Docker是一个Client - Server结构的系统，Docker的守护进程运行在主机上。通过Socket从客户端访问!

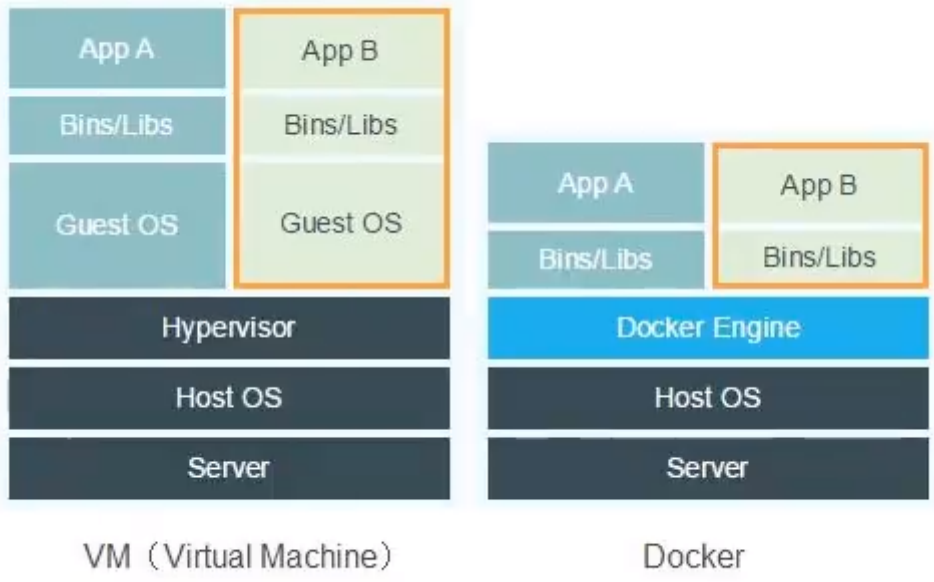
DockerServer接收到 Docker-Client的指令，就会执行这个命令!



docker为什么比VM快

docker有这比虚拟机更少的抽象层

docker利用的是宿主机的内核，VM需要是GuestOS



所以说，新建一个容器的时候，docker不需要想虚拟机一样重新加载一个操作系统内核，避免引导。虚拟机是加载Guest OS，分钟级别的

而docker是利用宿主机的操作系统，省略了这个复杂的过程，秒级！

Docker常用命令

帮助命令

```
# 显示docker的版本信息
docker version
# 显示docker的系统信息，包括镜像和容器数量
docker info
# 帮助命令
docker 命令 --helo
```

镜像命令

查看镜像

`docker images`：查看所有本地主机上的镜像

```
[root@localhost /]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    d1165f221234   3 months ago   13.3kB

# 解释
REPOSITORY  镜像的仓库源
TAG         镜像的标签
IMAGE ID    镜像的id
CREATED     镜像的创建时间
SIZE        镜像的大小
```

搜索镜像

`docker search`：搜索镜像

```
[root@localhost /]# docker search mysql
NAME            DESCRIPTION                                     STARS
OFFICIAL   AUTOMATED
mysql           MySQL is a widely used, open-source relation…   11003     [OK]

mariadb         MariaDB Server is a high performing open sou…   4169      [OK]

# 解释
# 可选项
# 通过星数量来过滤，不小于3000
--filter=STARS=3000
```

下载镜像

`docker pull`：镜像拉取

```
# 下载镜像 docker pull 镜像名[:tag]，默认用最新版
[root@localhost /]# docker pull mysql
Using default tag: latest # 如果不写 tag 默认就是最新版
latest: Pulling from library/mysql
69692152171a: Pull complete # 分层下载， docker image的核心，联合文件系统
1651b0be3df3: Pull complete
951da7386bc8: Pull complete
0f86c95aa242: Pull complete
37ba2d8bd4fe: Pull complete
6d278bb05e94: Pull complete
497efbd93a3e: Pull complete
f7fddf10c2c2: Pull complete
16415d159dfb: Pull complete
0e530ffc6b73: Pull complete
b0a4a1a77178: Pull complete
cd90f92aa9ef: Pull complete
Digest: sha256:d50098d7fcb25b1fcb24e2d3247cae3fc55815d64fec640dc395840f8fa80969
# 签名
Status: Downloaded newer image for mysql:latest
```

```
docker.io/library/mysql:latest # 真实地址

# 相互等价
docker pull mysql
docker.io/library/mysql:latest

# 指定版本下载
[root@localhost /]# docker pull mysql:5.7
5.7: Pulling from library/mysql
69692152171a: Already exists  # 已存在的不再下载
1651b0be3df3: Already exists
951da7386bc8: Already exists
0f86c95aa242: Already exists
37ba2d8bd4fe: Already exists
6d278bb05e94: Already exists
497efbd93a3e: Already exists
a023ae82eef5: Pull complete
e76c35f20ee7: Pull complete
e887524d2ef9: Pull complete
ccb65627e1c3: Pull complete
Digest: sha256:a682e3c78fc5bd941e9db080b4796c75f69a28a8cad65677c23f7a9f18ba21fa
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

删除镜像

`docker rmi`：删除镜像

```
# 通过 id 来删除镜像
[root@localhost /]# docker rmi -f c0cdc95609f1
Untagged: mysql:latest

# 删除指定镜像
docker rmi -f 镜像ID
# 删除多个镜像
docker rmi -f 镜像ID 镜像ID 镜像ID
# 删除全部镜像
docker rmi -f $(docker images -aq)
```

容器命令

说明：有了镜像才可以创建容器，linux，下载一个centos镜像来学习

```
docker pull centos
```

新建容器并启动

```
docker run [可选参数] image

# 参数说明
--name="Name"    容器名字 tomcat01  tomcat02 用来区分容器
-d               后台方式运行
-it              使用交互方式运行，进入容器查看内容
-P（大写）        指定容器的端口 -p 8080:8080 -p 8080
    -P ip:主机端口:容器端口
```

```
    -P 主机端口:容器端口（常用）
    -P 容器端口
-p（小写）        随机指定端口

# 测试
# 启动并进入容器
[root@localhost /]# docker run -it centos /bin/bash
# 查看容器内部的centos，基础版本，很多命令都是不完善的
[root@fb1bdb0d2cac /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
# 退出容器，回到主机
[root@fb1bdb0d2cac /]# exit
exit
```

守护容器

创建一个守护式容器：如果对于一个需要长期运行的容器来说，我们可以创建一个守护式容器

命令如下（容器名称不能重复）：

```
docker run -di --name=mycentos2 centos
```

登录守护式容器：

docker exec -it container_name (或者container_id)/bin/bash（exit退出时，容器不会停止）

命令如下：

```
docker exec -it mycentos2 /bin/bash
```

列出运行的容器

```
docker ps
        #列出当前正在运行的容器
-a      #列出当前正在运行的容器+带出历史运行过的容器
-n=1    #显示最近创建的容器
-q      #只显示容器的编号

# 查看运行的容器
docker ps
# 查看运行过的容器
docker ps -a
```

退出容器

```
exit    #直接容器停止并退出
CTRL+P+Q    # 容器不停止退出
```

删除容器

```
# 删除指定的容器，不能删除正在运行的容器，如果要强制删除，那就 rm -f
docker rm 容器ID
# 删除所有容器
docker rm -f $(docker ps -aq)
# 删除所有容器
docker ps -a -q|xargs docker rm
```

启动和停止容器

```
# 启动容器
docker start 容器ID
# 重启容器
docker restart 容器ID
# 停止容器
docker stop 容器ID
# 强制停止当前容器
docker kill 容器ID
```

常用其他命令

后台启动容器

```
# 通过 docker run -d 镜像名
docker run -d centos

# 问题 docker ps，发现 centos 停止了

# 常见的坑：docker 容器使用后台运行，就必须要有要一个前台进程，docker发现没有应用，就会自动停止
# nginx，容器启动后，发现自己没有提供服务，就会立刻停止
```

查看日志

```
# 显示日志
-tf
--tail
docker logs -tf -t --tail 10 容器ID
```

查看容器中的进程信息

```
docker top 容器ID
```

查看镜像元数据

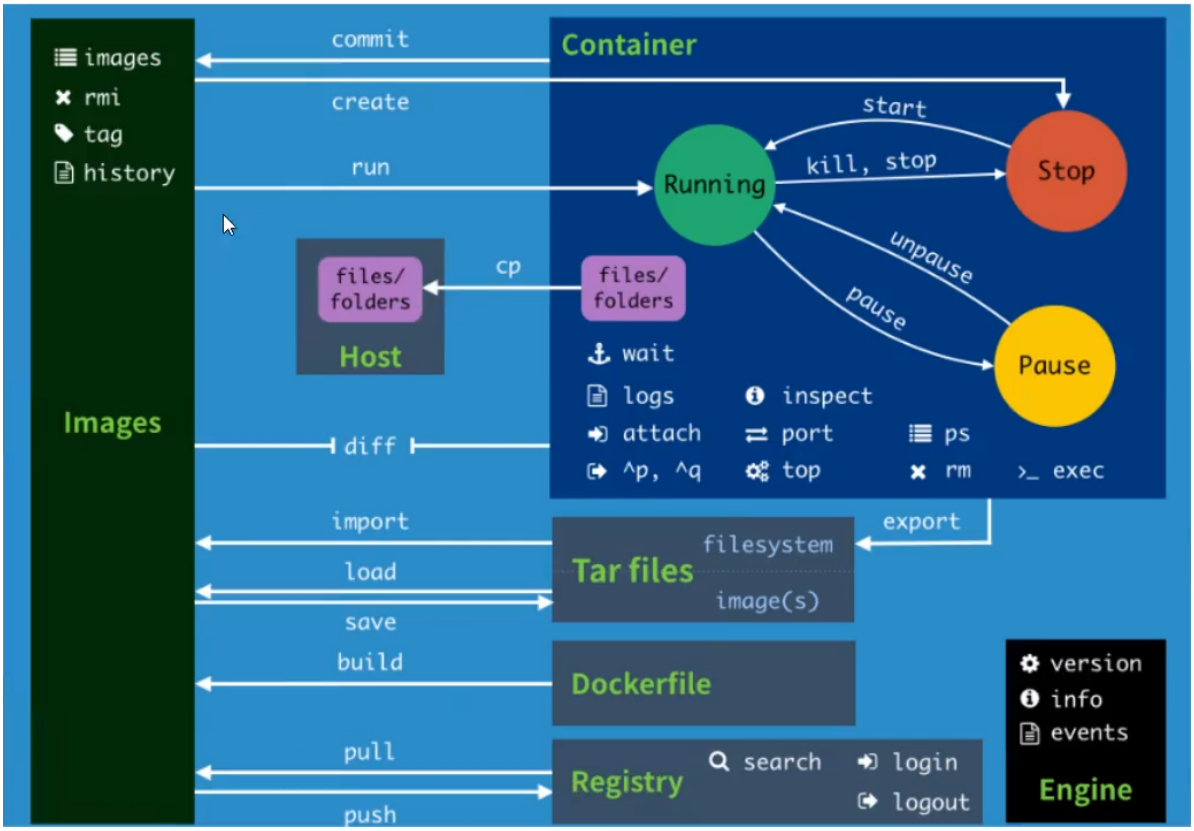
```
docker inspect 容器ID
```

进入当前正在运行的容器

```
# 我们通常容器都是使用后台方式运行的，需要进入容器，修改一些配置

# 方式一命令
docker exec -it 容器id bash

# 方式二命令
docker attach 容器id bash

# 区别
docker exec      进入容器后开启一个新的终端，可以在里面操作（常用）
docker attach    进入容器正在执行的终端
```

从容器内拷贝文件到主机上

```
# 将容器内的文件拷贝到指定目录
docker cp 容器ID:/home/test.java /home
```



Docker镜像讲解

镜像是什么

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

所有的应用，直接打包docker镜像，就可以直接跑起来！

如何得到镜像：

- 从远程仓库下载
- 朋友拷贝给你
- 自己制作一个镜像DockerFile

Docker镜像加载原理

> UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 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（没有父镜像），可以制作各种具体的应用镜像。

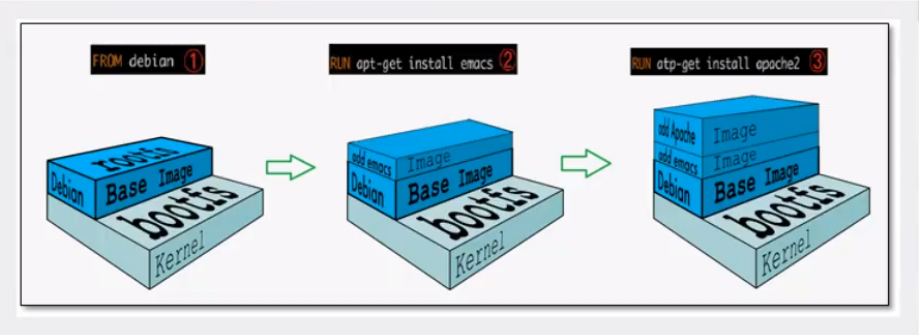
特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

> Docker镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader和kernel, bootloader主要是引导加载kernel, Linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu，Centos等等。



平时我们安装进虚拟机的CentOS都是好几个G，为什么Docker这里才200M？

```
[root@kuangshen home]# docker images centos
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              470671670cac        3 months ago        237MB
```

对于一个精简的OS，rootfs 可以很小，只需要包含最基本的命令，工具和程序库就可以了，因为底层直接用Host的kernel，自己只需要提供rootfs就可以了。由此可见对于不同的linux发行版, bootfs基本是一致的, rootfs会有差别, 因此不同的发行版可以公用bootfs。

分层理解

> 分层的镜像

我们可以去下载一个镜像，注意观察下载的日志输出，可以看到是一层一层的在下载！

```
[root@kuangshen home]# docker pull redis
Using default tag: latest
latest: Pulling from library/redis
54fec2fa59d0: Pull complete
9c94e11103d9: Pull complete
04ab1bfc453f: Pull complete
a22fde870392: Pull complete
def16cac9f02: Pull complete
1604f5999542: Pull complete
Digest: sha256:f7ee67d8d9050357a6ea362e2a7e8b65a6823d9b612bc430d057416788ef6df9
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

思考：为什么Docker镜像要采用这种分层的结构呢？

最大的好处，我觉得莫过于是资源共享了！比如有多个镜像都从相同的Base镜像构建而来，那么宿主机只需在磁盘上保留一份base镜像，同时内存中也只需要加载一份base镜像，这样就可以为所有的容器服务了，而且镜像的每一层都可以被共享。

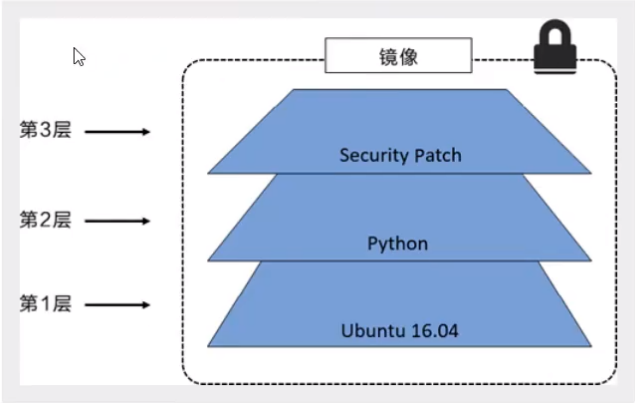
shell

理解：

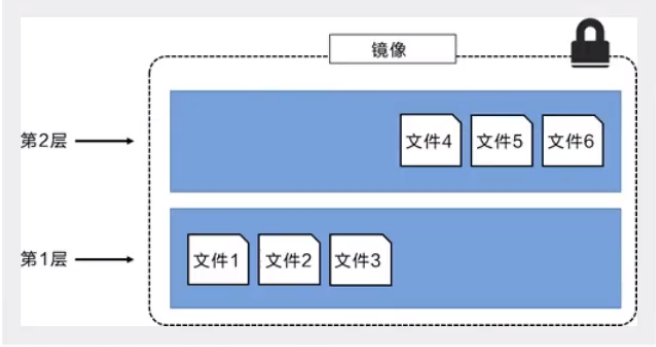
所有的 Docker 镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。

举一个简单的例子，假如基于 Ubuntu Linux 16.04 创建一个新的镜像，这就是新镜像的第一层；如果在该镜像中添加 Python包，就会在基础镜像层之上创建第二个镜像层；如果继续添加一个安全补丁，就会创建第三个镜像层。

该镜像当前已经包含 3 个镜像层，如下图所示（这只是一个用于演示的很简单的例子）。

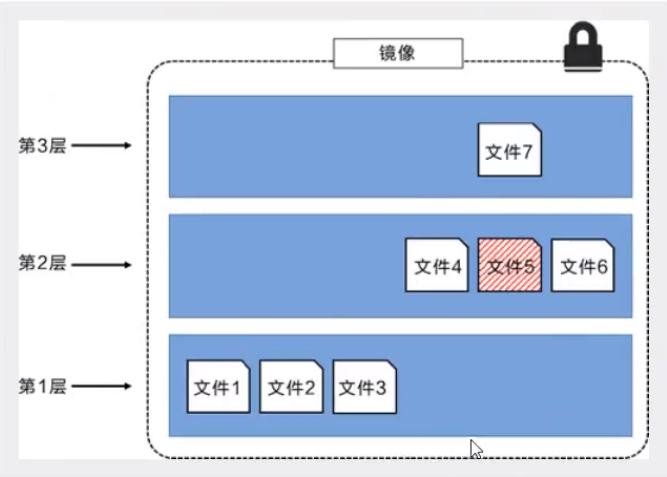


在添加额外的镜像层的同时，镜像始终保持是当前所有镜像的组合，理解这一点非常重要。下图中举了一个简单的例子，每个镜像层包含 3 个文件，而镜像包含了来自两个镜像层的 6 个文件。



上图中的镜像层跟之前图中的略有区别，主要目的是便于展示文件。

下图中展示了一个稍微复杂的三层镜像，在外部看来整个镜像只有 6 个文件，这是因为最上层中的文件 7 是文件 5 的一个更新版本。



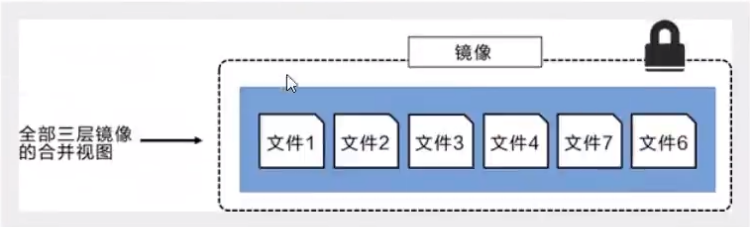
这种情况下，上层镜像层中的文件覆盖了底层镜像层中的文件。这样就使得文件的更新版本作为一个新镜像层添加到镜像当中。

Docker 通过存储引擎（新版本采用快照机制）的方式来实现镜像层堆栈，并保证多镜像层对外展示为统一的文件系统。

Linux 上可用的存储引擎有 AUFS、Overlay2、Device Mapper、Btrfs 以及 ZFS。顾名思义，每种存储引擎都基于 Linux 中对应的文件系统或者块设备技术，并且每种存储引擎都有其独有的性能特点。

Docker 在 Windows 上仅支持 windowsfilter 一种存储引擎，该引擎基于 NTFS 文件系统之上实现了分层和 CoW[1]。

下图展示了与系统显示相同的三层镜像。所有镜像层堆叠并合并，对外提供统一的视图。



> 特点

Docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部！

这一层就是我们通常说的容器层，容器之下的都叫镜像层！

Commit镜像

```
docker commit 提交容器成为一个新的副本

# 命令和git原理类似
docker commit -m="提交的描述信息" -a="作者" 容器id  目标镜像名：[TAG]
```

实战测试

```
# 1、启动一个默认的tomcat

# 2、发现这个默认的tomcat 是没有webapps应用， 镜像的原因，官方的镜像默认 webapps下面是没有文件的！

# 3、我自己拷贝进去了基本的文件

# 4、将我们操作过的容器通过commit提交为一个镜像！我们以后就使用我们修改过的镜像即可，这就是我们自己的一个修改的
```

容器数据卷

DockerFile

DockeFile介绍

dockerfile是用来构建dokcer镜像的文件!命令参数脚本

构建步骤:

1、编写一个dockerfile文件

2、docker build构建成为一个镜像

3、docker run运行镜像

4、docker push发布镜像(DockerHub、阿里云镜像仓库)

```
FROM scratch
ADD centos-7-x86_64-docker.tar.xz /

LABEL \
    org.label-schema.schema-version="1.0" \
    org.label-schema.name="CentOS Base Image" \
    org.label-schema.vendor="CentOS" \
    org.label-schema.license="GPLv2" \
    org.label-schema.build-date="20201113" \
    org.opencontainers.image.title="CentOS Base Image" \
    org.opencontainers.image.vendor="CentOS" \
    org.opencontainers.image.licenses="GPL-2.0-only" \
    org.opencontainers.image.created="2020-11-13 00:00:00+00:00"

CMD ["/bin/bash"]
```

很多官方的镜像都是基础包，很多功能都没有，我们通常会自己搭建自己的镜像!

官方既然可以制作镜像，我们也可以!

DockerFile构建过程

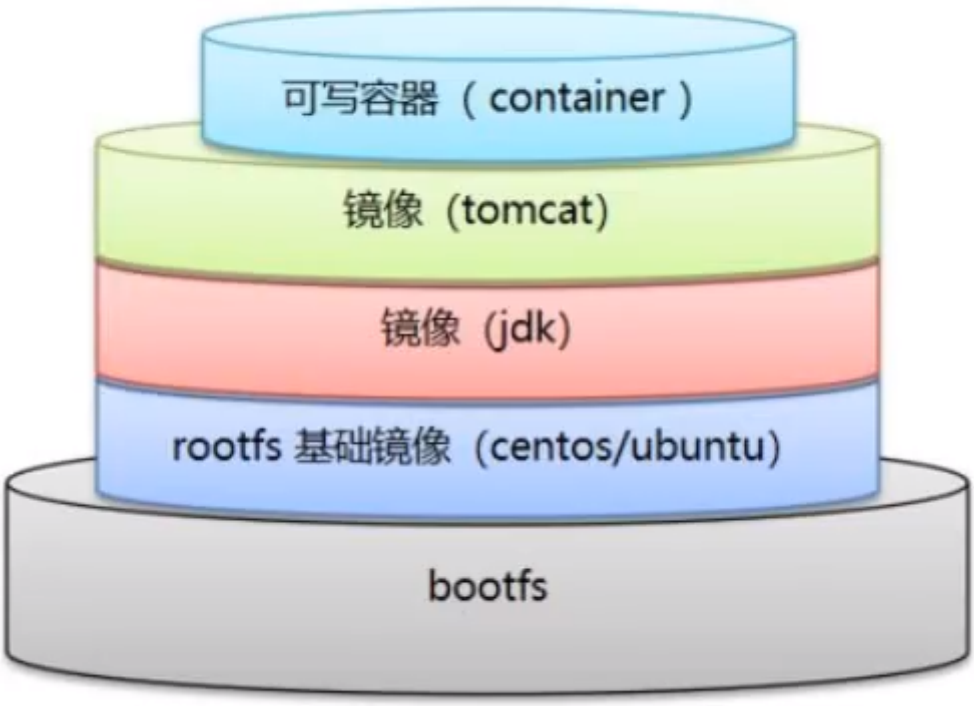
基础知识:

1、每个保留关键字（指令）都必须是大写字母

2、执行从上到下顺序执行

3、# 表示注释

4、每一个指令都会创建提交一个新的镜像层，并提交



dockerfile是面向开发的，我们以后要发布项目，做镜像，就需要编写dockerfile文件，这个文件是十分简单的！

Docker镜像逐渐成为了企业交付的标准，必须要掌握！

步骤：开发、部署、运维。。。缺一不可

DockerFile：构建文件，定义了一切的步骤，源代码

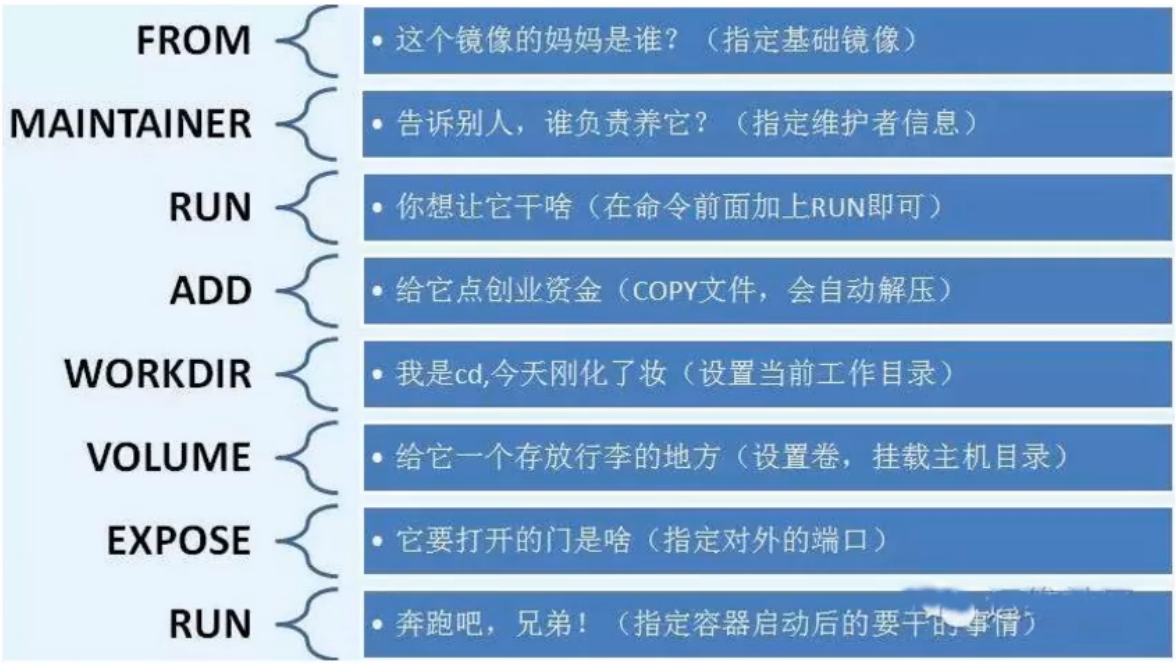
DockerImages：通过DockerFile构建生成的镜像，最终发布和运行的产品！

（原来是jar war交付给客户，现在直接生成镜像给用户）

Docker容器：容器就是镜像运行起来提供服务的

DockerFile的指令

```
FROM        # 基础镜像，一切从这里开始构建
MAINTAINER  # 镜像是谁写的：姓名+邮箱
RUN         # 镜像构建的时候需要运行的命令
ADD         # 步骤：tomcat镜像，这个tomcat压缩包就是添加内容
WORKDIR     # 镜像的工作目录
VOLUME      # 挂在的目录
EXPOSE      # 指定暴露端口
CMD         # 指定容器启动的时候要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT  # 指定容器启动的时候要运行的命令，可以追加命令
ONBUILD     # 当构建一个被继承 DockerFile 这个时候就会运行 ONBUILD 的指令。触发指令
COPY        # 类似ADD，将我们的文件拷贝到镜像中
ENV         # 构建的时候设置环境变量
```



实战测试

Docker Hub中99%镜像都是从这个基础镜像过来的`FROM scratch`，然后配置需要的软件和配置来进行的构建

> 创建一个自己的 centos

1、编写DockerFile

```
[root@kuangshen dockerfile]# cat mydockerfile-centos
FROM centos
MAINTAINER kuangshen<24736743@qq.com>

ENV MYPATH /usr/local
WORKDIR $MYPATH

RUN yum -y install vim
RUN yum -y install net-tools

EXPOSE 80

CMD echo $MYPATH
CMD echo "----end----"
CMD /bin/bash
```

2、通过这个文件构建镜像

```
docker build -f mydockerfile-centos -t mycentos:0.1 .
```

查看镜像构建历史

```
docker history 镜像ID
```

实战tomcat镜像

1、准备镜像文件：tomcat压缩包，jdk压缩包！

2、编写 dockerfile 文件

官方命名：`Dockerfile`，build会自动寻找这个文件

```
FROM centos
MAINTAINET kuangshen<24736743@qq.com>

COPY readme.txt /usr/local/readme.txt

ADD jdk-8u11-linux-x64.tar.gz /usr/local/
ADD apache-tomcat-9.0.22.tar.gz /usr/local/

RUN yum -y install vim

ENV MYPATH /usr/local
WORKDIR $MYPATH

ENV JAVA_HOME /usr/local/jdk1.8.0_11
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
ENV CATALINA_HOME /usr/local/apache-tomcat-9.0.22
ENV CATALINA_BASH /usr/local/apache-tomcat-9.0.22
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin

EXPOSE 8080

CMD /usr/local/apache-tomcat-9.0.22/bin/startup.sh && tail -F /url/local/apache-tomcat-
9.0.22/bin/logs/catalina.out
```

3、构建镜像

```
docker build -t diytomcat .
```