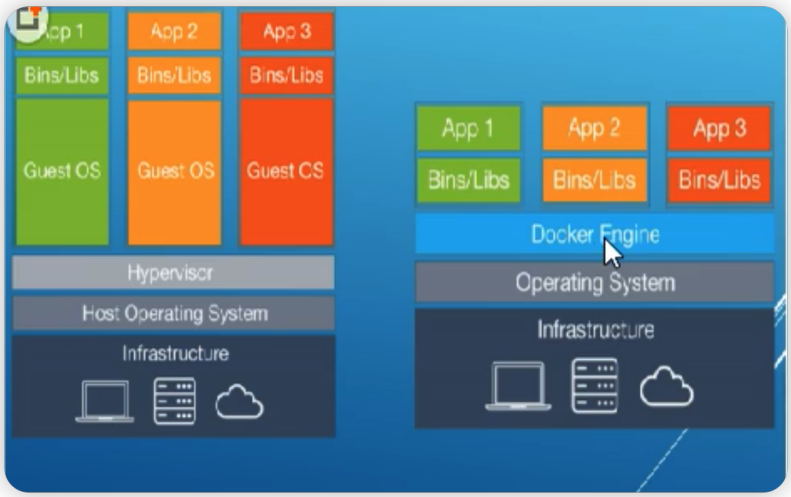
【3-05】docker快速入门

笔记本：rh_环境部署
创建时间：2022/5/10 23:01　　更新时间：2022/9/13 9:19
作者：兰鸣人花道

【Docker】容器化部署技术

docker三大核心组件

- docker的概述
  - docker产生的背景？
    - 开发自测完成后，交给运维部署
      - 但是，运维部署的环境部署出来有问题
      - 开发说我自己的环境是对的，这个时候就有冲突了
      - 同时，可能不止是一个环境部署有问题，可能还有其他环境，比如TEST环境、UAT环境、PRO环境，集群就更老火了。。。
        - 开发、运维、测试都崩溃。。。
      - 这就成为了一个痛点
      - 为什么有问题呢？
        - 1）环境差异
        - 2）应用配置文件有差异
        - 。。。
    - 于是，软件带环境安装，就产生了docker
      - 一套包含了开发人员所有的原始环境(代码、运行环境、依赖库、配置文件、数据文件等)
      - 那么我就可以在任何环境上直接运行
      - 以前部署项目的代码叫搬家，现在直接就是搬整栋楼；以前只买鱼，现在把鱼缸那些一套全买了；
  - docker的理念？
    - 一次编译，到处运行，不依赖于环境了
      - Docker是基于Go语言实现的云开源项目
      - Docker的主要目标是“Build，Ship and Run Any App，Anywhere”，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管理，使用户的App(可以是一个web应用或者数据库应用等等)及其运行环境能够做到"一次封装，到处运行"
        - build，构建
        - ship，传输
        - run，运行
    - 总结，什么是docker？
      - docker就是解决了运行环境和配置问题的软件容器，方便持续集成并有助于整体发布的容器化虚拟技术。
  - docker的优势



	Docker容器	虚拟机(VM)
操作系统	与宿主机共享OS	宿主机OS上运行虚拟机OS
存储大小	镜像小，便于存储与传输	镜像庞大(vmdk、vdi等)
运行性能	几乎无额外的性能损失	操作系统额外的CPU、内存消耗
移植性	轻便、灵活，适应于Linux	笨重，与虚拟化技术耦合度高
硬件亲和性	面向软件开发者	面向硬件运维者

- 传统的虚拟机
  - vmware workstation，virtual box，virtual pc
    - 可以在一个操作系统中运行另一种操作系统
    - 模拟的是包括一整套操作系统
  - 启动慢，分钟级的
  - 资源占用多
  - 冗余步骤多
  - 有Hypervisor硬件资源虚拟化
- 容器化虚拟技术
  - docker就去掉了Hypervisor硬件资源虚拟化，换成了Docker Engine
    - 那么运行在docker容器上的程序直接使用的都是实际物理机的硬件资源
    - 因此在cpu、内存利用率上docker将会有明显上的效率优势
  - LXC， Linux Containers
    - 模拟的不是一个完整的操作系统
    - 只需要精华版、缩小版、浓缩版的的小型的操作系统
      - centos/ubuntu基础镜像仅200M不到，iso文件多大(4个多G吧)
      - 容器与虚拟机不同，不需要捆绑一整套操作系统
      - 只需要软件所需的库资源和设置
      - 因此变得轻量级，并且能保证在任何环境中的软件都能始终如一地运行
  - docker启动是秒级的
    - docker利用的是宿主机的内核，而不需要Guest OS
    - 因此，当新建一个容器时，docker不需要和虚拟机一样重新加载一个操作系统内核
    - 从而避免了引导、加载操作系统内核这个比较费时费资源的过程
    - 当新建一个虚拟机时,虚拟机软件需要加载Guest OS,这个新建过程是分钟级别的
    - 而docker由于直接利用宿主机的操作系统，则省略了这个过程，因此新建一个docker容器只需要几秒钟

        - 容器内的应用进程直接运行于宿主机的内核
            - 容器没有自己的内核，而且也没有进行硬件虚拟
            - 因此容器要比传统的虚拟机更为轻便
        - 每个容器之间互相隔离
            - 每个容器有自己的文件系统
            - 容器之间进程不会相互影响
            - 能区分计算资源
        - 宿主机可以部署100~1000个容器，你这个传统的虚拟化能行？
        - 性能，尤其是IO和内存的消耗低
        - 更轻量
            - 基于容器的虚拟化，仅包含业务运行的所需的Runtime环境
        - 更高效
            - 无操作系统虚拟化的开销
                - 计算：轻量，无额外开销
                - 存储：系统盘aufs/dm/overlayfs；数据盘volume
                - 网络：宿主机网络，NS隔离
        - 更敏捷、更灵活
            - 分层的存储和包管理，devops理念
            - 支持多种网络配置
- 镜像
    - 镜像就是软件 + 运行环境的一整套
    - 镜像可能包括什么？
        - 代码
        - 运行的操作系统发行版
        - 各种配置
        - 数据文件
        - 运行文档
        - 运行依赖包
        - 。。。
    - 镜像就是模板
- 容器
    - 容器就是镜像的运行实例
        - 容器可以有多个，均来自于同一个镜像，镜像是一个只读的模板
    - 容器就是集装箱
        - 容器就是运行着的一个个独立的环境，独立的软件服务
        - 容器都是相互隔离、保证安全的平台
    - 容器也可以看做一个简易版的linux环境
        - 包括root用户权限、进程空间、用户空间和网络空间等
        - 和运行在其中的应用程序
- 仓库
    - 存放镜像的地方
    - 2仓库【Repository】是集中存放镜像文件的场所
    - 仓库注册服务器【Registry】，仓库注册服务器上存放着多个仓库
        - 每个仓库中又包含了多个镜像
        - 每个镜像有不同的标签Tag
    - 仓库有公开仓库【public】和私有仓库【private】两种形式
        - 最大的公开仓库：Docker Hub
            - 国内的公开仓库有：阿里云、网易云等
        - 私有仓库，一般自己公司搭建的
            - 一般也是运维搭建，测试不用管

- 鲸鱼背上有集装箱
  - 鲸鱼：**docker**
  - 集装箱：容器实例 ---> 来自镜像
  - 鲸鱼在蓝色的大海里：宿主机操作系统

1、image镜像

- docker image ls， 列出本机的镜像
- docker rmi 镜像id， 删除镜像
  - 可以删除多个 docker rmi 镜像id1 镜像id2 镜像id3 ...
- docker rmi 仓库名:TAG，删除镜像，同上一个命令
  - 可以删除多个 docker rmi 仓库名1:TAG1 仓库名2:TAG2 ...
- docker search 仓库名，搜索docker hub上的仓库
- docker image inspect 镜像id， 查看镜像详情
- docker image inspect 仓库名:TAG， 查看镜像详情，同上一个命令
- docker save -o XXX.tar 镜像:tag， 导出镜像
- docker [image] load -i XXX.tar， 导入镜像
- docker tag 要起名的那个镜像id 自己要起的什么名
  - 对镜像起别名

- 下面命令不需要掌握：
  - docker image ls -qa，只查询出镜像的id
  - docker rmi -f $(docker image ls -qa)，强制删除所有的镜像
  - docker rmi -f `docker images -qa`，也是强制删除所有的镜像
  - docker image ls --digests， 附带查看digests信息
  - docker image ls --no-trunc，镜像id全部展示，不截断

1）镜像操作

```
[root@lanhai ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
busybox             latest              cabb9f684f8b        10 days ago         1.24MB
httpd               latest              1132a4fc88fa        2 weeks ago         143MB
nginx               1.14-alpine         8a2fb25a19f5        2 years ago         16MB
[root@lanhai ~]# docker search python

[root@lanhai ~]# docker pull python
```

```
Using default tag: latest
latest: Pulling from library/python
bb7d5a84853b: Pull complete
f02b617c6a8c: Pull complete
d32e17419b7e: Pull complete
c9d2d81226a4: Pull complete
3c24ae8b6604: Pull complete
8a4322d1621d: Pull complete
b777982287b6: Pull complete
2c5fb32d4bef: Pull complete
4f3be23cccd3: Pull complete
Digest: sha256:a487658b37559c499868dd4bdcdc6b18ed25cbfb5a02d054c9eaefaf713d5aca
Status: Downloaded newer image for python:latest
[root@lanhai ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
busybox             latest              cabb9f684f8b        10 days ago         1.24MB
python              latest              4246fb19839f        10 days ago         917MB
httpd               latest              1132a4fc88fa        2 weeks ago         143MB
nginx               1.14-alpine         8a2fb25a19f5        2 years ago         16MB
[root@lanhai ~]# docker rmi 4246fb19839f
Untagged: python:latest
Untagged: python@sha256:a487658b37559c499868dd4bdcdc6b18ed25cbfb5a02d054c9eaefaf713d5aca
Deleted: sha256:4246fb19839fd033a0dd925c1f89cd1ad482c6b703d56f34bf0d2808b076e132
Deleted: sha256:e70202aaab7d43f3a2dd807ca15450a7e9fd67a5ba04334dbabe16cc88a391d4
Deleted: sha256:ee867c53632845b4431b1152625f44a4b0f93ca5aec0b30a8fa6c3cc7d34775b
Deleted: sha256:aa62ac1c3ab9cd1685bdc0ea13733876a8cf41fa7f29ebe1d6477e22248797b3
Deleted: sha256:7271700556cd5b8701ca5e39766d0e45b6fa64e5f94636e61098d31358bdfb5e
Deleted: sha256:e67b6800e9e8882c7060611038966ea29afe619bce27d0ea01528979a0f5c0fd
Deleted: sha256:995950c940fdede4906e13ddb5a13691b727b942a9b67afc23cc0172d80897a8
Deleted: sha256:b7a4a299f0c4a0e9d6f4156cd61b3a00c0595d9ee3db2dd7888f3a855b541fd6
Deleted: sha256:a9e0e6b8fdcd469c8785099c4559093696ad2c7da957d355557a17ed1bb8d23f
Deleted: sha256:62a747bf1719d2d37fff5670ed40de6900a95743172de1b4434cb019b56f30b4
[root@lanhai ~]#
```

```
#    启动容器 busybox
[root@lanhai ~]# docker run -it --name busybox busybox /bin/sh
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ # mkdir -pv /data/html/
created directory: '/data/'
created directory: '/data/html/'
/ # touch /data/html/index.html
/ # echo "busybox start httpd index.html..." >> /data/html/index.html
/ # httpd -f -h /data/html/


#    访问
[root@lanhai ~]# docker ps
CONTAINER
ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
549fd8740e9b        busybox             "/bin/sh"           3 minutes ago       Up 3
minutes                             busybox
[root@lanhai ~]# docker inspect busybox

。。。
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"11567d63623e8f56dd832506c266c13ff11b8a14cbda999cd844995aeeb33150",
                    "EndpointID":
"b8a26942d087f6719fa4cb1b7e98887040dff008d692045ecfcacc5b5898e376",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
[root@lanhai ~]#
```

```
[root@lanhai ~]# curl 172.17.0.2
busybox start httpd index.html...
[root@lanhai ~]#
```

2）container操作

```
#    启动一个最简单的nginx
[root@lanhai ~]# docker run -d --name nginx-container-name nginx:1.14-alpine
065ae3d2c91f1a75d6e70468d75de25a09ca3f493364cd63266e36753a003849
[root@lanhai ~]#
[root@lanhai ~]# docker inspect 065ae3d2c91f1
[
    {
        "Id": "065ae3d2c91f1a75d6e70468d75de25a09ca3f493364cd63266e36753a003849",
        "Created": "2021-11-06T18:51:51.057347192Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
。。。


                    "EndpointID":
"f86e4abebaa5616463b820c07a9118aa39a988071eab3380f87616a912424d39",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
[root@lanhai ~]# curl 172.17.0.2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>


<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>


<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@lanhai ~]#
```

3）镜像的打包及导入

```
[root@lanhai ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
lanhai/busybox          v0.0.2              a45fa6626b14        2 hours ago         1.24MB
busybox                 v0.0.1              a45fa6626b14        2 hours ago         1.24MB
busybox                 latest              cabb9f684f8b        10 days ago         1.24MB
```

```
httpd                   latest              1132a4fc88fa        2 weeks ago         143MB
nginx                   1.14-alpine         8a2fb25a19f5        2 years ago         16MB
quay.io/coreos/flannel  v0.9.1              2b736d06ca4c        3 years ago         51.3MB
[root@lanhai ~]# docker save -o my_image.tar busybox:v0.0.1 lanhai/busybox:v0.0.2
[root@lanhai ~]# ls
busybox_lanhai.tar  datamanage-v1.1.2-20211019  my_image.tar  test  更新说明1.1.2.txt
[root@lanhai ~]#


[root@lanhai ~]# docker load -i my_image.tar
Loaded image: busybox:v0.0.1
Loaded image: lanhai/busybox:v0.0.2
[root@lanhai ~]#
[root@lanhai ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
lanhai/busybox          v0.0.2              a45fa6626b14        2 hours ago         1.24MB
busybox                 v0.0.1              a45fa6626b14        2 hours ago         1.24MB
busybox                 latest              cabb9f684f8b        10 days ago         1.24MB
httpd                   latest              1132a4fc88fa        2 weeks ago         143MB
nginx                   1.14-alpine         8a2fb25a19f5        2 years ago         16MB
quay.io/coreos/flannel  v0.9.1              2b736d06ca4c        3 years ago         51.3MB
[root@lanhai ~]#
```

2、container容器

- docker run xxx
  - -d 后台运行
  - --name 自己起的容器名称，如果不指定，系统自动给你随机分配一个名字
  - -p 本宿主机端口:容器端口
    - 比如 -p 3307:3306 是把容器的3306端口映射到本机的3307端口
  - -P 随机端口号
    - 访问的时候，用docker ps 查看端口号，然后通过ip:端口号访问
  - -i 交互式
    - -i, --interactive Keep STDIN open even if not attached
  - -t 终端
    - -t, --tty Allocate a pseudo-TTY
  -
  - -v 宿主机目录:容器内部目录
    - -v /tomcat/data:/usr/local/tomcat/webapps 挂载数据卷，将tomcat的部署的目录(/usr/local/tomcat/webapps)挂载到自定义的挂载卷(/tomcat/data)上面去
    - -v /tomcat/conf:/usr/local/tomcat/conf 挂载数据卷，将tomcat的配置文件路径(/usr/local/tomcat/conf)挂载到本机定义的挂载卷(/tomcat/conf)上面去，以后启动别的容器，就可以重用这个conf配置
    - tomcat:8.0-jre8 启动的tomcat的镜像名称
    - 数据卷的作用？
      - 1）持久化容器的数据
      - 2）容器间共享数据
- docker run hello-world 启动一个镜像为hello-world的容器
- docker ps 查看正在运行的容器
  - docker run hello-world
  - 运行容器，就是要用run命令，
  - hello-world 没有接TAG，就表示要使用latest版本
  - 会提示你找不到这个镜像
  - 会自动给你下载这个hello-world的最新版本latest的镜像
- docker ps -a 查看所有容器，包括运行的和没有运行的
- docker ps -q 查看容器的id，静默显示
- docker ps -qa 查看所有容器的id，包括没有运行的
- docker run --name mycentos centos 运行容器，--name xxx表示给这个容器起个名字

- docker run -it centos 交互式运行容器，并进入容器，-i交互式的，-t给分配一个伪终端
  - exit 退出容器，但是出来后容器就被关闭了
  - ctrl + P + Q 退出容器，但是不关闭容器
- docker stop 容器名称 # 停止容器，这种是缓慢的停止
  - docker stop 容器ID # 也是可以的
- docker start centos 启动容器
- docker restart centos 重启容器
- docker kill centos 停止容器，粗暴的停止
- docker inspect mycentos 查看容器的详细信息
- docker exec -it c1 /bin/bash也可进入到容器
  - 进入之后，可以进行自定义的修改，比如修改tomcat主页
  - exit
  - ctrl + P+Q 退出终端，而不关闭容器
- docker commit -a '作者' -m '注释信息' 容器名/容器ID 自己要起什么名[仓库名:TAG]
  - 提交新的镜像：docker commit -a 'lanhai' -m 'comment, update tomcat index page' tomcat1 tomcat:test1_v1.0.1
  - 然后就可以使用新的镜像docker run -d -p 8086:8080 --name tomcat3 tomcat:test1_v1.0.1
  - 使用场景，你如果在一个容器内了做了一些修改配置，然后以后也想用这样的配置，就可以这样搞哈
- docker run -d --name c1 centos 后台启动容器
- docker logs c1 查看日志
- docker logs -f c1 滚动查看日志
- docker logs --tail 3 c1 显示最后几行日志
- docker cp 本机文件 容器名:容器的路径目录 #将本机的文件拷贝到容器内部的指定的那个目录
- docker cp c1:/tmp/test.log /root/test.log 容器内拷贝出来
- docker run -d -p 8888:8080 -v /root/volume_test:/container_volume --name t1 tomcat
  挂载宿主机的/root/volume_test到容器的/container_volume目录

- 以下命令不需要掌握：
  - docker ps -l，显示上次运行的容器
  - docker ps -n 3，显示最近三次运行的容器
- 某些说明：
  - docker run -d centos，为什么运行后就退出呢？
    - docker ps -a 查看，会发现容器已经退出
    - 注意：docker容器后台运行，就必须有一个前台进程
    - 容器运行的命令如果不是那些一直挂起的命令(比如top、tail等)，就是会自动退出的
    - 这个是docker的机制问题，比如你的web容器，我们以nginx为例。
      - 正常情况下，我们配置启动服务只需要启动响应的service即可。例如servcie nginx start
      - 但是，这样做，nginx为后台进程模式运行，就导致docker前台没有运行的应用
      - 这样，容器后台启动后，会立即自杀，因为它觉得它没事可做了
      - 所以，最佳的解决方式，是将你要运行的程序以前台进程的形式运行
  - docker run -d centos /bin/sh -c "while true;do echo hello canglaoshi $(date +'%Y-%m-%d %H:%M:%S');sleep 2;done"
    - 写个死循环，占用前台进程
    - docker logs -f 容器名， 可以查看容器的日志
- docker top 容器名称， 查看容器内部的进程
- docker attach 容器名称，直接进入容器
- docker exec -it 容器 bash
  - docker exec -it 容器名称 /bin/bash
  - docker exec -it 容器名称 /bin/sh
  - docker exec -it 容器名称 ls -l /tmp， 可以不进入容器，直接查看容器内命令结果
  - docker exec 容器名称 ls -l /tmp，可以不进入容器，直接查看容器内ls -l /tmp命令的结果
- attach 和 exec 的区别：

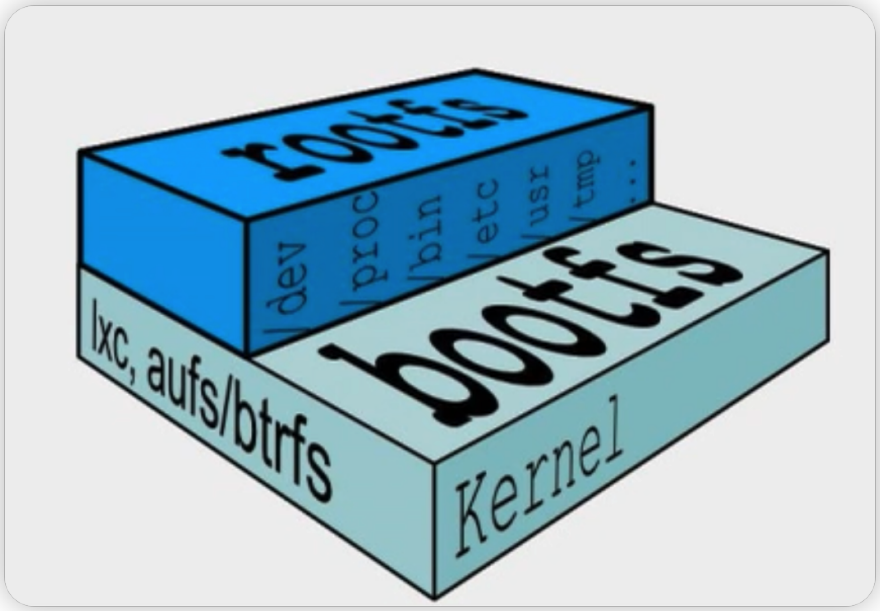
    - 两个命令都可以进入容器
    - attach 是直接进入之后才能干活
    - exec 也可以直接进入后干活，但是exec还能在不进入容器的情况下，运行命令
- docker run -d -v xxx:yyy:ro 镜像， 挂载卷设置只读权限
    - 表示什么呢？
        - 1）宿主机上挂载的那个目录可以正常写入
        - 2）容器内挂载的那个目录只读，【容器只能看】
- --volumes-from
    - docker run -it --name centos1 lanhai:v0.0.1
    - docker run -it --name centos2 --volumes-from centos1 lanhai:v0.0.1
    - docker run -it --name centos3 --volumes-from centos1 lanhai:v0.0.1
        - 分别在centos2里面的挂载目录新建数据
        - 分别在centos3里面的挂载目录新建数据
        - 删除容器centos1，那么centos2和centos3的都在
        - 同时，任何一个挂载卷新增修改数据，都会同步
            - 所以，可以实现容器间数据共享

3、repository仓库

- docker push 镜像名:TAG

4、拓展（不用掌握）：

- 1）镜像的原理
    - 镜像的加载分层采用UnionFS联合文件系统
    - UnionFS：
        - 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
        - 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(没有父镜像)，可以只做各种具体的应用镜像
    - 特点：；
        - 一次同时加载多个文件系统
        - 但是从外面看起来，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
- 2）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就叫UnionFS
        - bootfs(boot file system)主要包含bootloader和kernel，bootloader主要是引导加载kernel,linux刚启动的时候会加载bootfs文件系统，在Docker镜像的最底层就是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加载完成后，整个内核就在内存中了，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，包含的就是典型的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Centos、Ubuntu等。



    - 对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了
    - 因为底层直接用的HOST的kernel，自己只需要提供rootfs就行了
    - 由此可见对于不同的linux发行版，bootfs基本是一致的,rootfs有差别，因此不同的发行版可以公用bootfs
- 3）为什么docker采用这种分层的原理呢？
    - 最大的好处，就是资源共享
    - 比如：
        - 有多个镜像都从相同的base镜像构建而来，那么宿主机只需要在磁盘上保存一份base镜像就可以了
        - 同时内存中也只需加载一份base镜像，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
        - 比如，你同时下载了tomcat、centos、nginx、mysql等的镜像
            - 那么他们这些镜像里面可能有很多的base层镜像是一样的，那么就不用每个都去下载了
            - 只保留一份就可以了
- 4）镜像的特点？
    - 镜像是只读的
    - 当镜像启动为容器后，一个新的可写层被加载到了镜像的顶部
    - 这一层通常被称为容器层，而容器层以下的都叫镜像层

5、一些问题解惑？

- 1）docker run -it -p 8081:8080 --name tomcat1 tomcat:8.0-jre8 bash
    - 像这个命令启动容器后为什么，访问不了？
        - 比如 localhost:8081
        - 或者浏览器访问 http://ip:8081
    - 分析：
        - 启动容器的时候，带了启动命令bash，那么就会tomcat的Dockerfile里面的CMD命令给覆盖了
        - 说白了，就是这个tomcat容器启动的时候，没有启动前台命令
        - 如何解决？
            - 其实现在run -it的方式已经进入了容器，那么可以手动启动tomcat的脚本，就能访问了
            - cd /usr/local/tomcat/bin

- ./startup.sh
- 再次访问,ok