docker

什么是容器

- 容器技术已经成为应用程序封装和交付的核心技术
- 容器技术的核心有以下几个内核技术组成
  - Cgroups(Control Groups)：资源管理
  - NameSpace：进程隔离
  - SELinux：安全
- 由于是在物理机上实施隔离，启动一个容器，可以像启动一个进程一样快速

什么是docker

- Docker是完整的一套容器管理系统
- Docker提供了一组命令，让用户更加方便直接地使用容器技术，而不需要过多关心底层内核技术
- docker优点
  - 相比于传统的虚拟化技术，容器更加简洁高效
  - 传统虚拟机需要给每个vm安装操作系统
  - 容器使用的共享公共库和程序
- docker缺点
  - 容器的隔离性没有虚拟化强
  - 共有Linux内核，安全性有先天缺陷
  - SELinux难以驾驭
  - 监控容器和容器排错是挑战
- 安装docker

```
#yum源在RHEL7-extras.iso中
yum -y install docker
systemctl restart docker
systemctl enable docker
```

什么的镜像

- 在docker中容器是基于镜像启动的
- 镜像是启动容器的核心
- 镜像采用分层设计
- 使用快照的cow技术，确保底层数据不丢失

镜像操作

- docker hub镜像仓库： https://hub.docker.com

```
docker images   #查看所有镜像
docker search busybox   #搜索镜像
docker search centos
docker search nginx
```

```
#下载镜像
docker help pull    #查看帮助
docker pull docker.io/busybox
#上传镜像
docker help push    #查看帮助
docker push docker.io/busybox
#导出镜像
docker save docker.io/busybox:latest -o busybox.tar #-o后加起的名字
#导入镜像
docker load -i nginx.tar
docker load < nginx.tar
#启动镜像
docker images
docker run -it docker.io/centos:latest /bin/bash    #/bin/bash 启动命令
#查看容器信息
docker ps
```

- 信号向量
  - 容器运行的进程，真机可看到，可以用kill杀死，但真机运行的进程，容器中看不到
- docker 命令

```
docker images
docker history  #查看镜像制作历史
docker inspect  #查看镜像底层信息，即查看详细信息
    docker inspect 镜像 / 容器 / docker组件
docker rmi      #删除本地镜像，容器先删，镜像才可以删
docker save     #另存为tar包
docker load     #导入镜像
docker search   #搜索
docker tag      #修改镜像名称和标签
    docker tag docker.io/busybox:latest 新名称：新标签 #相当于硬链接
docker run
    -i: 交互的
    -t: 终端
    -d: 服务，放后台用
docker ps [-a]  #查看容器列表，-q显示id
docker stop / start / restart 容器ID
docker top 容器ID #查看容器进程信息
docker rm 容器ID  #删容器
```

- docker 重录：后面执行结果可以作为前一命令的参数

```
docker rm $(docker ps -aq)
```

- 怎么退出容器，且不杀死上帝进程

```
ctrl + p + q
```

- 怎么进入容器

```
#以上帝进程进，不推荐
docker attach 容器ID
#以容器中的命令进
docker exec -it 容器ID
    docker exec -it /bin/bash
```

- 自定义镜像：docker commit
  - 使用镜像启动容器，在该容器基础上修改
  - 另存为一个新镜像

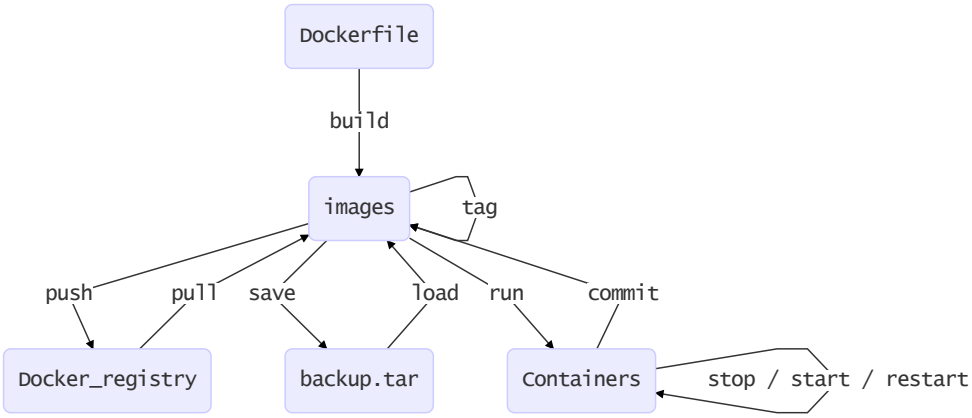
```
docker run -itd docker.io/centos
... #增删改数据，安装软件等。。。
docker ps   #获得容器的ID
dcker commit 容器ID 新容器名
    docker commit 456464 docker.io/myos:latest
```

  - 使用Dockerfile（脚本）自定义一个新镜像
    - Dockerfile格式
      - FROM #基础镜像
      - MAINTAINER #镜像创建者信息
      - EXPOSE #开发的端口
      - ENV #设置变量
      - ADD #复制文件到镜像
      - RUN #制作镜像时执行的命令，可以有多个
      - WORKDIR #定义容器默认工作目录
      - CMD #容器启动时执行的命令，仅可只有一条

```
vim Dockerfile
FROM docker.io/centos:latest
RUN rm -f /etc/yum.repos.d/*.repo
ADD local.repo /etc/yum.repos.d/local.repo
RUN yum -y install net-tools vim psmisc tree lftp iproute
RUN yum clean all
docker built -t 名字 Dockerfile目录
    docker built -t myos:latest /root
```

自定义镜像仓库

- registry：共享镜像的一台服务器（镜像化的一台服务器）



自定义私有仓库

- 安装私有仓库

```
yum -y install docker-distribution
systemctl start docker-distribution
systemctl enable docker-distribution
port : 5000
```

- 仓库配置文件及数据存储路径

  /etc/docker-distribution/registry/config.yml

  /var/lib/registry

- 客户端配置
  - 修改配置文件 /etc/sysconfig/docker

```
13: ADD_REGISTERY='--add-registry 192.168.1.31:5000'
#允许非加密方式访问
24: INSECURE_REGISTRY='--insecure-registry 192.168.1.31:5000'
systemctl restart docker
```

- 上传镜像
  - 为镜像创建标签， IP地址要写宿主机的IP地址和主机名
  - docker tag 镜像：标签 ip:5000/镜像：latest

    ```
    docker tag docker.io/busybox:latest 192.168.1.31:5000/busybox:latest
    ```

  - 上传

  docker push ip:5000/镜像：latest

```
docker push 192.168.1.31:5000/busybox:latest
```

  批量上传

```
for i in latest python httpd sshd
    do
    docker tag myos:${i} 192.168.1.31:5000/myos:${i}
        docker push 192.168.1.31:5000/myos:${i}
    done
```

- 查看私有镜像仓库中的镜像名称
  - curl http://192.168.1.31:5000/v2/_catalog
- 查看某一查看的标签
  - curl http://192.168.1.31:5000/v2/myos/tags/list
- 客户端若使用私有仓库需要在配置文件中指定远程私有仓库地址即可
  - 远程启动镜像

```
docker run -it 192.168.1.31:5000/myos:latest
```

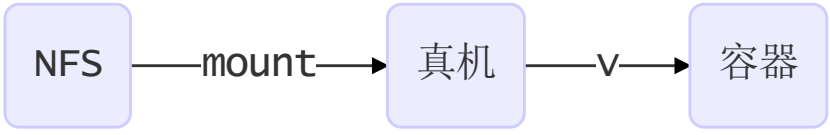
存储卷

- docker容器不保存任何书籍
- 重要数据使用外部卷存储（数据持久化）
- 容器可以挂载真机目录或共享存储为卷

主机卷的映射

- 将真机目录挂载到容器中提供持久化存储
- 目录不存在自动创建
- 目录存在直接覆盖

```
docker run -v 真机目录：容器目录 -itd 容器
docker run -v /data:/data -itd myos:latest
```



docker网络拓补

- 查看默认Docker创建的网络模型

```
docker network list
... bridge  #桥接模型
... host    #主机模型
... none    #无网络
```

- 新建Docker网络模型

```
docker network create --subnet=10.10.10.0/24 docker1
```

- 使用自定义网桥

```
docker run -itd --network=docker1 myos:latest
```

- 默认容器可以访问外网
- 外部网络主机不可以访问容器内的资源
- 容器可以把宿主机变成对应的服务

```
#使用-p把容器端口和宿主机端口绑定
#-p 宿主机端口：容器端口
docker run -itd -p 80:80 docker.io/myos:httpd
docker run -itd -p 80:80 docker:id/msos:nginx
```