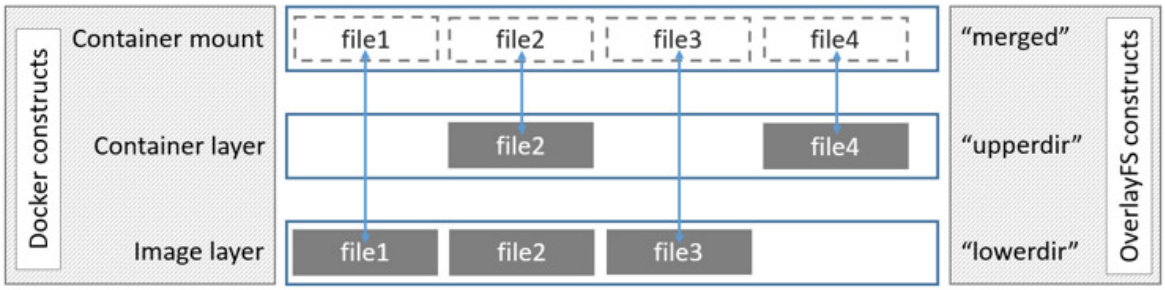
什么是镜像

容器解决应用开发、测试和部署的问题，而镜像解决应用部署环境问题。镜像是一个只读的容器模板，打包了应用程序和应用程序所依赖的文件系统以及启动容器的配置文件，是启动容器的基础。镜像所打包的文件内容就是容器的系统运行环境——rootfs(根文件系统或根目录)。容器与镜像类似对象与类的关系。

Docker镜像原理

1. 分层： Docker镜像采用分层的方式构建，每一个镜像都由一组镜像组合而成。每一个镜像层都可以被需要的镜像所引用，实现了镜像之间共享镜像层的效果。这样的分层设计在镜像的上传与下载过程当中有效的减少了镜像传输的大小，在传输过程当中本地或注册中心只需要存在一份底层的基础镜像层即可，真正被保存和下载的内容是用户构建的镜像层。而在构建过程中镜像层通常会被缓存以缩短构建过程。
2. 写时复制：底层镜像层在多个容器间共享，每个容器启动时不需要复制一份镜像文件，而是将所有需要的镜像层以只读的方式挂载到一个挂载点，在只读层上再覆盖一层读写层。在容器运行过程中产生的新文件将会写入到读写层，被修改过的底层文件会被复制到读写层并且进行修改，而老文件则被隐藏。
3. 联合挂载：docker采用联合挂载技术，在同一个挂载点同时挂载多个文件系统，从而使得容器的根目录看上去包含了各个镜像层的所有文件。如下图：



LowerDir：被引用的镜像层，该层所有内容均为只读
UpperDir：容器启动之后，创建的读写层
Merged：容器启动后，会将LowerDir的所有条目的所有文件连同UpperDir的内容，一起挂载到Merged，从而形成一个完成的根目录

4. 内容寻址：根据镜像层内容计算校验和，生成一个内容哈希值，并使用该值来充当镜像层ID、索引镜像层。内容寻址提高了镜像的安全性，在pull、push和load、save操作后检测数据的完整性。另外基于内容哈希来索引镜像层，对于来自不同构建的镜像层，只要拥有相同的内容哈希值，就能被不同的镜像所引用。

Docker镜像关键概念

1. registry：注册中心，用来保存docker镜像，其中包括镜像的层次结构和关于镜像的元数据。
2. repository：仓库，即由具有某个功能的Docker镜像的所有迭代版本构成的镜像组。
3. manifest：Docker镜像元数据文件，在pull、push、save和load中作为镜像结构和基础信息的描述文件。在镜像被pull或者load到Docker宿主机时，manifest被转化为本地的镜像配置文件config
4. image：镜像，用来存储一组相关的元数据信息，主要包括镜像的架构（如amd64）、镜像默认配置信息、构建镜像的容器配置信息、包含所有镜像层的rootfs。
5. layer：镜像层，是docker用来管理镜像的中间概念，镜像是由镜像层组成的，单个镜像层可以被多个镜像和容器共享。

6. dockerfile：是一个镜像制作过程的定义，文档包含了镜像制作的所有命令和完整操作流程

镜像管理命令

docker build

`docker build` 命令用于使用 `Dockerfile` 创建镜像。

```
docker build [OPTIONS] PATH | URL | -
OPTIONS说明：

--add-host :向hosts文件中添加自定义 host:ip 映射

--build-arg=[] :设置镜像创建时的变量；

--cache-from :指定镜像用作当前构建的缓存镜像

--cgroup-parent :容器的可选父cgroup

--compress : 使用gzip压缩构建上下文

--cpu-period :限制 CPU CFS周期；

--cpu-quota :限制 CPU CFS配额；

-c,--cpu-shares :设置 cpu 使用权重；

--cpuset-cpus :指定使用的CPU id；

--cpuset-mems :指定使用的内存 id；

--disable-content-trust :跳过镜像校验，默认为true；

-f,--file :指定要使用的Dockerfile路径；

--force-rm :不论构建是否成功，总是删除中间容器，默认false。注意：中间容器；

--iidfile :将镜像ID写入到文件

--isolation :使用容器隔离技术；
    linux系统：只支持default，即linux命名空间隔离技术
    windows系统：
        default:表示使用docker守护进程配置的选项，否则默认使用process选项
    process:命名空间隔离
    hyperv:基于Hyper-V 管理程序基于分区的隔离

--label=[] :设置镜像使用的元数据；

-m :设置内存最大值；

--memory-swap :设置Swap的最大值为内存+swap，"-1"表示不限swap；

--network: 在构建期间为RUN指令设置网络模式（默认“default”）与docker run指令无关
```

--no-cache :创建镜像的过程不使用缓存；

--pull :总是尝试去更新镜像的新版本；

--quiet, -q :安静模式，成功后只输出镜像 ID；

--rm :构建成功后，删除中间容器，默认true。注意：中间容器，不是镜像；

--security-opt :安全选项

--shm-size :设置/dev/shm的大小，默认值是64M；/dev/shm 是基于内存的tmpfs文件系统。

--ulimit :Ulimit配置。

--tag, -t : 镜像的名字及标签，通常 name:tag 或者 name 格式；可以在一次构建中为一个镜像设置多个标签。

--target : 设置要生成的目标生成阶段

--ulimit : Ulimit 选项

```
docker build  -t nodetodo:v1.0.0 -f Dockerfile .
```

docker images

```
docker images : 列出本地镜像。
```

```
docker images [OPTIONS] [REPOSITORY[:TAG]]

OPTIONS说明：

-a :列出本地所有的镜像（含中间映像层，默认情况下，过滤掉中间映像层）；

--digests :显示镜像的摘要信息；

-f :显示满足条件的镜像；

--format :指定返回值的模板文件；

--no-trunc :显示完整的镜像信息；

-q :只显示镜像ID
```

```
# 查看所有悬空镜像
docker images --filter dangling=true
```

```
# 清除所有悬空镜像
docker image prune
```

docker rmi

```
docker rmi : 删除本地一个或多个镜像。
```

```
docker rmi [OPTIONS] IMAGE [IMAGE...]

OPTIONS说明：

-f :强制删除；

--no-prune :不移除该镜像的过程镜像，默认移除；
```

docker tag

```
docker tag : 标记本地镜像，将其归入某一仓库。
```

```
docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]
```

```
docker tag mynginx:v1 mynginx:1.0
```

docker save

```
docker save : 将指定镜像保存成 tar 归档文件。
```

```
docker save [OPTIONS] IMAGE [IMAGE...]
OPTIONS 说明：

-o :输出到的文件。
```

```
docker save -o mynginx.tar mynginx:v1
```

```
# 解压文件，查看内容
tar vxf mynginx.tar -C mynginx
```

docker load

```
docker load : 导入使用 docker save 命令导出的镜像。
```

```
docker load [OPTIONS]
OPTIONS 说明：

--input , -i : 指定导入的文件，代替 STDIN。

--quiet , -q : 精简输出信息。
```

```
docker load -i mynginx.tar
```

docker import

```
docker import : 从归档文件中创建镜像。
```

```
docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]
OPTIONS说明：

-c :应用docker 指令创建镜像；

-m :提交时的说明文字
```

```
docker import  mynginx.tar mynginx:v3
```

docker commit

```
docker commit :从容器创建一个新的镜像。
```

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
OPTIONS说明：

-a :提交的镜像作者；

-c :使用Dockerfile指令来创建镜像；

-m :提交时的说明文字；

-p :在commit时，将容器暂停
```

```
docker commit -a "nick" -m "mynginx commit" mynginx mynginxtest:1.0
```

镜像仓库命令

docker login/logout

```
docker login : 登陆到一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

docker logout : 登出一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub
```

```
docker login [OPTIONS] [SERVER]

docker logout [OPTIONS] [SERVER]
OPTIONS说明：

-u :登陆的用户名

-p :登陆的密码
```

```
docker login -u 用户名 -p 密码

docker logout
```

docker pull

```
docker pull : 从镜像仓库中拉取或者更新指定镜像
```

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
OPTIONS说明：

-a :拉取所有 tagged 镜像

--disable-content-trust :忽略镜像的校验,默认开启
```

```
docker pull nginx
```

docker push

```
docker push : 将本地的镜像上传到镜像仓库,要先登陆到镜像仓库
```

```
docker push [OPTIONS] NAME[:TAG]
OPTIONS说明：

--disable-content-trust :忽略镜像的校验,默认开启
```

docker search

```
docker search : 从Docker Hub查找镜像
```

```
docker search [OPTIONS] TERM
OPTIONS说明：

--automated :只列出 automated build类型的镜像；

--no-trunc :显示完整的镜像描述；

-f <过滤条件>:列出收藏数不小于指定值的镜像
```

```
//从 Docker Hub 查找所有镜像名包含 nginx，并且收藏数大于 10 的镜像
docker search -f stars=10 nginx
```

参数说明：
NAME: 镜像仓库源的名称
DESCRIPTION: 镜像的描述
OFFICIAL: 是否 docker 官方发布
stars: 类似 Github 里面的 star，表示点赞、喜欢的意思。
AUTOMATED: 自动构建。

实践

镜像分层和联合挂载

nginx 镜像为例

```
//运行一个nginx容器
docker run -d -p 81:80 --name mynginx1 nginx:latest
//以shell终端访问该容器，目的直观的看到该容器下的文件目录
docker exec -it mynginx1 /bin/bash
//docker save 保存镜像到宿主机归档文件
docker save -o nginx/nginx.tar nginx:latest
//解压归档文件
tar vxf nginx/nginx.tar
//查看manifest
//查看每一层的文件内容，json等

//将运行的容器提交为镜像，重复以上的动作，对比两个镜像的差别，可以看出镜像层被复用的现象
```

commit构建镜像

```
//运行一个nginx容器
docker run -d -p 81:80 nginx
//语法
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
//案例
docker commit b415cdd95dfc mynginx:v1
```

注意：提交一个容器只会保存在提交的那个时刻容器的文件系统的状态，而不是进程状态。docker容器不是虚拟机。如果环境的状态依赖于一些正在运行的进程状态，而这些京城不同通过标准文件恢复的话。docker commit 将无法帮助用户保存所需要的状态。

适用场景：

构建临时的测试镜像

容器被入侵后，使用docker commit，基于被入侵的容器构建镜像，从而保留现场，方便以后追溯。

除了这两种场景，不建议你使用docker commit来构建生产现网环境的镜像。

使用docker commit构建的镜像包含了编译构建、安装软件，以及程序运行产生的大量无用文件，这会导致镜像体积很大，非常臃肿。

镜像分享

tag命令的使用方式

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
docker tag IMAGE_ID TARGET_IMAGE[:TAG]
```

作用：

本地镜像管理，可以用版本号来指定镜像的tag,方便镜像管理

可以用来修改本地镜像名和tag

指定目标远程仓库镜像和tag

公共仓库分享

1. 登录 https://hub.docker.com/ 创建公有的 repository，注意此处的repository具体到了镜像名

例如：本例子使用的repository 为：xlhmzch/mynginx 表示 xlhmzch 这个账户下mynginx镜像

2. 登录远程仓库

```
//语法
//docker login [OPTIONS] [SERVER]
docker login
//打tag
docker tag mynginx:v1 xlhmzch/mynginx:v1
//推送镜像
docker push xlhmzch/mynginx:v1
```

3. 镜像推送成功之后，即可通过docker pull 拉取到镜像

docker save 加 docker load 分享

通过docker save 保存一个或多个镜像包到.tar文件，通过docker load 加载tar文件中的镜像

```
docker save : 将指定镜像保存成 tar 归档文件。
```

```
docker save -o images.tar mynginx:v1 mysql:5.7.30
```

```
docker load : 导入使用 docker save 命令导出的镜像。
```

```
docker load -i images.tar
```

私有注册中心搭建并分享镜像

htpasswd 是开源 http 服务器 apache httpd 的一个命令工具，用于生成 http 基本认证的密码文件。

```
docker run -d -p 5000:5000 --name registry --restart=always registry:2
docker ps -a
# curl http://localhost:5000/v2/_catalog
# {"repositories":""}
```

```
mkdir -p $HOME/registry/
mkdir -p $HOME/registry/auth
mkdir -p $HOME/registry/data

sudo apt install apache2-utils

htpasswd  -Bbn testuser testpwd > /home/nick/registry/auth/htpasswd

# 若启用了user命名空间隔离，需要修改目录owner和组
sudo chown 165536:165536 -R registry

docker run -d -p 5000:5000 --name registry --restart=always -v
$HOME/registry/data:/var/lib/registry -v $HOME/registry/auth:/auth -e
"REGISTRY_AUTH=htpasswd" -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm"  -e
REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd registry:2
```

```
vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--insecure-registry localhost:5000

sudo systemctl daemon-reload
sudo systemctl restart docker
```

注意：--insecure-registry 指定一个受信任的注册中心，IP:Port 或 Domain:Port

```
docker login http://localhost:5000

docker tag mynginxtest:1.1 localhost:5000/mynginx:1.0.0
docker push localhost:5000/mynginx:1.0.0
```