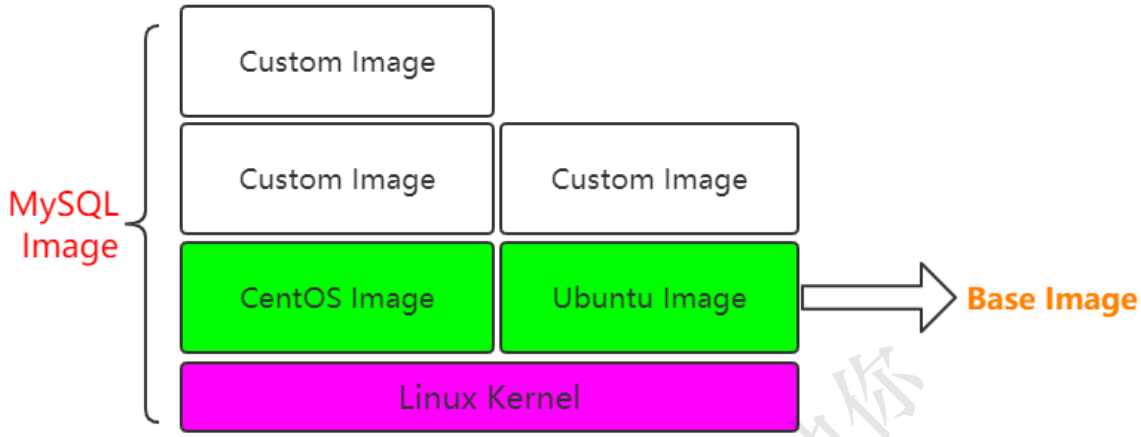
02 Image and Container

2.1 深入探讨Image



> 说白了，image就是由一层一层的layer组成的。

2.1.1 官方image

> https://github.com/docker-library

mysql

> https://github.com/docker-library/tomcat/blob/master/8.5/jdk8/openjdk/Dockerfile

2.1.2 Dockerfile

> 不妨我们也来制作一个自己的image镜像，顺便学习一下Dockerfile文件中常见语法

2.1.2.1 FROM

指定基础镜像，比如FROM ubuntu:14.04

```
FROM ubuntu:14.04
```

2.1.2.2 RUN

在镜像内部执行一些命令，比如安装软件，配置环境等，换行可以使用""

```
RUN groupadd -r mysql && useradd -r -g mysql mysql
```

2.1.2.3 ENV

设置变量的值，ENV MYSQL_MAJOR 5.7，可以通过docker run --e key=value修改，后面可以直接使用${MYSQL_MAJOR}

```
ENV MYSQL_MAJOR 5.7
```

2.1.2.4 LABEL

设置镜像标签

```
LABEL email="itcrazy2016@163.com"
LABEL name="itcrazy2016"
```

2.1.2.5 VOLUME

指定数据的挂在目录

```
VOLUME /var/lib/mysql
```

2.1.2.5 COPY

将主机的文件复制到镜像内，如果目录不存在，会自动创建所需要的目录，注意只是复制，不会提取和解压

```
COPY docker-entrypoint.sh /usr/local/bin/
```

2.1.2.6 ADD

将主机的文件复制到镜像内，和COPY类似，只是ADD会对压缩文件提取和解压

```
ADD application.yml /etc/itcrazy2016/
```

2.1.2.7 WORKDIR

指定镜像的工作目录，之后的命令都是基于此目录工作，若不存在则创建

```
WORKDIR /usr/local
WORKDIR tomcat
RUN touch test.txt
```

> 会在/usr/local/tomcat下创建test.txt文件

```
WORKDIR /root
ADD app.yml test/
```

> 会在/root/test下多出一个app.yml文件

2.1.2.8 CMD

容器启动的时候默认会执行的命令，若有多个CMD命令，则最后一个生效

```
CMD ["mysqld"]
或
CMD mysqld
```

2.1.2.9 ENTRYPOINT

和CMD的使用类似

```
ENTRYPOINT ["docker-entrypoint.sh"]
```

和CMD的不同

docker run执行时，会覆盖CMD的命令，而ENTRYPOINT不会

2.1.2.10 EXPOSE

指定镜像要暴露的端口，启动镜像时，可以使用-p将该端口映射给宿主机

```
EXPOSE 3306
```

2.1.3 Dockerfile实战Spring Boot项目

```
(1)创建一个Spring Boot项目

(2)写一个controller
    @RestController
    public class DockerController {
        @GetMapping("/dockerfile")
        @ResponseBody
        String dockerfile() {
            return "hello docker" ;
        }
    }

(3)mvn clean package打成一个jar包
    在target下找到"dockerfile-demo-0.0.1-SNAPSHOT.jar"

(4)在docker环境中新建一个目录"first-dockerfile"

(5)上传"dockerfile-demo-0.0.1-SNAPSHOT.jar"到该目录下，并且在此目录创建Dockerfile

(6)创建Dockerfile文件，编写内容
    FROM openjdk:8
    MAINTAINER itcrazy2016
    LABEL name="dockerfile-demo" version="1.0" author="itcrazy2016"
    COPY dockerfile-demo-0.0.1-SNAPSHOT.jar dockerfile-image.jar
    CMD ["java","-jar","dockerfile-image.jar"]

(7)基于Dockerfile构建镜像
    docker build -t test-docker-image .

(8)基于image创建container
    docker run -d --name user01 -p 6666:8080 test-docker-image

(9)查看启动日志docker logs user01

(10)宿主机上访问curl localhost:6666/dockerfile
    hello docker

(11)还可以再次启动一个
    docker run -d --name user02 -p 8081:8080 test-docker-image
```

2.1.4 镜像仓库

2.1.4.1 docker hub

> hub.docker.com
>
> itcrazy2018登录

```
(1)在docker机器上登录
    docker login

(2)输入用户名和密码

(3)docker push itcrazy2018/test-docker-image
[注意镜像名称要和docker id一致，不然push不成功]

(4)给image重命名，并删除掉原来的
    docker tag test-docker-image itcrazy2018/test-docker-image
    docker rmi -f test-docker-image

(5)再次推送，刷新hub.docker.com后台，发现成功

(6)别人下载，并且运行
    docker pull itcrazy2018/test-docker-image
    docker run -d --name user01 -p 6661:8080 itcrazy2018/test-docker-image
```

2.1.4.2 阿里云docker hub

> 阿里云docker仓库
>
> https://cr.console.aliyun.com/cn-hangzhou/instances/repositories
>
> 参考手册
>
> https://cr.console.aliyun.com/repository/cn-hangzhou/dreamit/image-repo/details

```
(1)登录到阿里云docker仓库
sudo docker login --username=itcrazy2016@163.com registry.cn-
hangzhou.aliyuncs.com

(2)输入密码

(3)创建命名空间，比如itcrazy2016

(4)给image打tag
sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/itcrazy2016/test-
docker-image:v1.0

(5)推送镜像到docker阿里云仓库
sudo docker push registry.cn-hangzhou.aliyuncs.com/itcrazy2016/test-docker-
image:v1.0

(6)别人下载，并且运行
    docker pull registry.cn-hangzhou.aliyuncs.com/itcrazy2016/test-docker-
image:v1.0
    docker run -d --name user01 -p 6661:8080 registry.cn-
hangzhou.aliyuncs.com/itcrazy2016/test-docker-image:v1.0
```

2.1.4.3 搭建自己的Docker Harbor

```
(1)访问github上的harbor项目
```

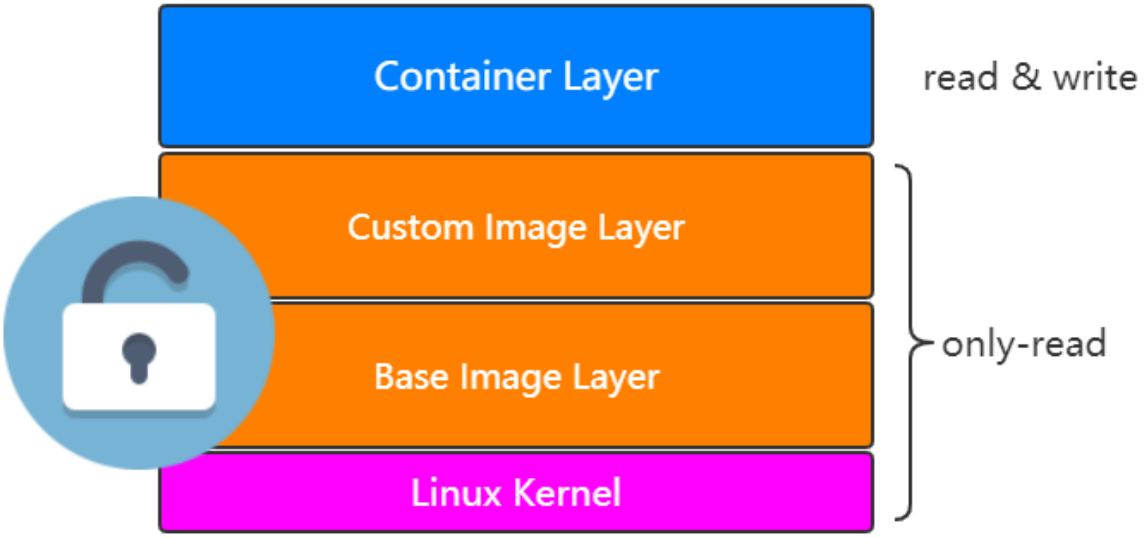
```
    https://github.com/goharbor/harbor

(2)下载版本，比如1.7.1
    https://github.com/goharbor/harbor/releases

(3)找一台安装了docker-compose[这个后面的课程会讲解]，上传并解压
    tar -zxvf xxx.tar.gz

(4)进入到harbor目录
    修改harbor.cfg文件，主要是ip地址的修改成当前机器的ip地址
    同时也可以看到Harbor的密码，默认是Harbor12345

(5)安装harbor，需要一些时间
    sh install.sh

(6)浏览器访问，比如39.100.39.63，输入用户名和密码即可
```

2.1.5 Image常见操作

```
(1)查看本地image列表
    docker images
    docker image ls

(2)获取远端镜像
    docker pull

(3)删除镜像[注意此镜像如果正在使用，或者有关联的镜像，则需要先处理完]
    docker image rm imageid
    docker rmi -f imageid
    docker rmi -f $(docker image ls)      删除所有镜像

(4)运行镜像
    docker run image

(5)发布镜像
    docker push
```


2.2 深入探讨Container

> 既然container是由image运行起来的，那么是否可以理解为container和image有某种关系?



理解：其实可以理解为container只是基于image之后的layer而已，也就是可以通过docker run image创建出一个container出来。

2.2.1 container到image

> 既然container是基于image之上的，想想是否能够由一个container反推出image呢?
>
> 肯定是可以的，比如通过docker run运行起一个container出来，这时候对container对一些修改，然后再生成一个新的image，这时候image的由来就不仅仅只能通过Dockerfile咯。

实验

```
(1)拉取一个centos image
    docker pull centos

(2)根据centos镜像创建出一个container
    docker run -d -it --name my-centos centos

(3)进入my-centos容器中
    docker exec -it my-centos bash

(4)输入vim命令
    bash: vim: command not found

(5)我们要做的是
    对该container进行修改，也就是安装一下vim命令，然后将其生成一个新的centos

(6)在centos的container中安装vim
    yum install -y vim

(7)退出容器，将其生成一个新的centos，名称为"vim-centos-image"
    docker commit my-centos vim-centos-image

(8)查看镜像列表，并且基于"vim-centos-image"创建新的容器
    docker run -d -it --name my-vim-centos vim-centos-image

(9)进入到my-vim-centos容器中，检查vim命令是否存在
    docker exec -it my-vim-centos bash
    vim
```

结论：可以通过docker commit命令基于一个container重新生成一个image，但是一般得到image的方式不建议这么做，不然image怎么来的就全然不知咯。

2.2.2 container资源限制

> 如果不对container的资源做限制，它就会无限制地使用物理机的资源，这样显然是不合适的。
>
> 查看资源情况：docker stats

2.2.2.1 内存限制

```
--memory    Memory limit
如果不设置 --memory-swap，其大小和memory一样
docker run -d --memory 100M --name tomcat1 tomcat
```

2.2.2.2 CPU限制

```
--cpu-shares    权重
docker run -d --cpu-shares 10 --name tomcat2 tomcat
```

2.2.2.3 图形化资源监控

> https://github.com/weaveworks/scope

```
sudo curl -L git.io/scope -o /usr/local/bin/scope
sudo chmod a+x /usr/local/bin/scope
scope launch 39.100.39.63
```

```
# 停止scope
scope stop

# 同时监控两台机器，在两台机器中分别执行如下命令
scope launch ip1 ip2
```

2.2.3 container常见操作

```
(1)根据镜像创建容器
    docker run -d --name -p 9090:8080 my-tomcat tomcat

(2)查看运行中的container
    docker ps

(3)查看所有的container[包含退出的]
    docker ps -a

(4)删除container
    docker rm containerid
    docker rm -f $(docker ps -a)    删除所有container

(5)进入到一个container中
    docker exec -it container bash

(6)根据container生成image
    docker
```

```
(7)查看某个container的日志
    docker logs container

(8)查看容器资源使用情况
    docker stats

(9)查看容器详情信息
    docker inspect container

(10)停止/启动容器
    docker stop/start container
```

2.3 底层技术支持

> Container是一种轻量级的虚拟化技术，不用模拟硬件创建虚拟机。
>
> Docker是基于Linux Kernel的Namespace、CGroups、UnionFileSystem等技术封装成的一种自定义容器格式，从而提供一套虚拟运行环境。

```
Namespace：用来做隔离的，比如pid[进程]、net[网络]、mnt[挂载点]等

CGroups: Controller Groups用来做资源限制，比如内存和CPU等

Union file systems：用来做image和container分层
```