容器镜像

前言

- 本章主要介绍容器镜像分层结构及其关键特性，如copy-on-write；讲解容器镜像构建的具体方法：docker commit和dockerfile；最后介绍Registry。

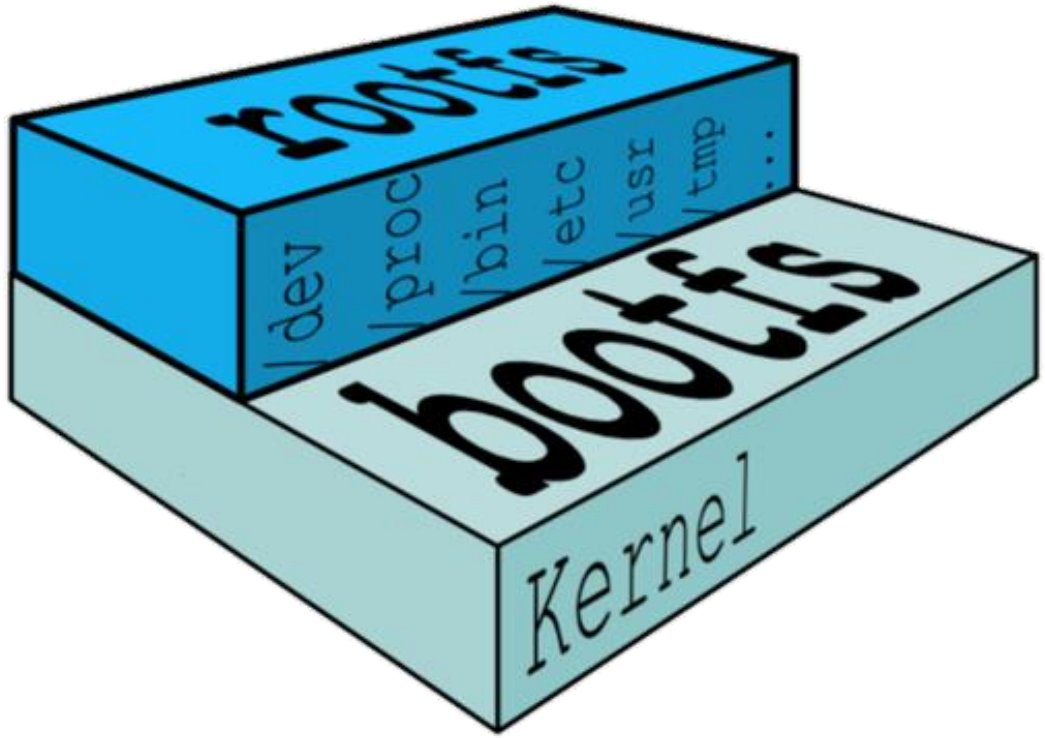
目标

- 学完本课程后，您将能够：
  - 描述容器镜像分层结构
  - 了解容器镜像关键特性
  - 掌握容器镜像构建方法和关键指令
  - 了解Registry

目录

Linux操作系统结构

- Linux操作系统由内核空间和用户空间构成：
  - kernel：Linux系统内核
  - rootfs：Linux系统中的用户空间文件系统。rootfs是一个操作系统所包含的文件、配置和目录，但并不包括操作系统kernel。



容器镜像

- 容器镜像是容器的模板，容器是镜像的运行实例，runtime根据容器镜像创建容器。

Build and Ship any Application Anywhere

- 容器镜像挂载在容器根目录下，是为容器中的应用提供隔离后执行环境的文件系统。
  - 容器镜像打包了整个操作系统的文件和目录（rootfs），当然也包括应用本身。即，应用及其运行所需的所有依赖，都在被封装在容器镜像中。保证了本地环境和云端环境的高度一致。
- 容器镜像采用分层结构：
  - 所有容器共享宿主机Kernel，并且不能修改宿主机Kernel。即，容器运行过程中使用容器镜像里的文件，使用宿主机OS上的Kernel。

base镜像

- base镜像
  - 从scratch构建，不依赖其他镜像
    - scratch本身是个空镜像
  - 其他镜像可在base镜像上进行扩展，创建新的镜像。

scratch ☆

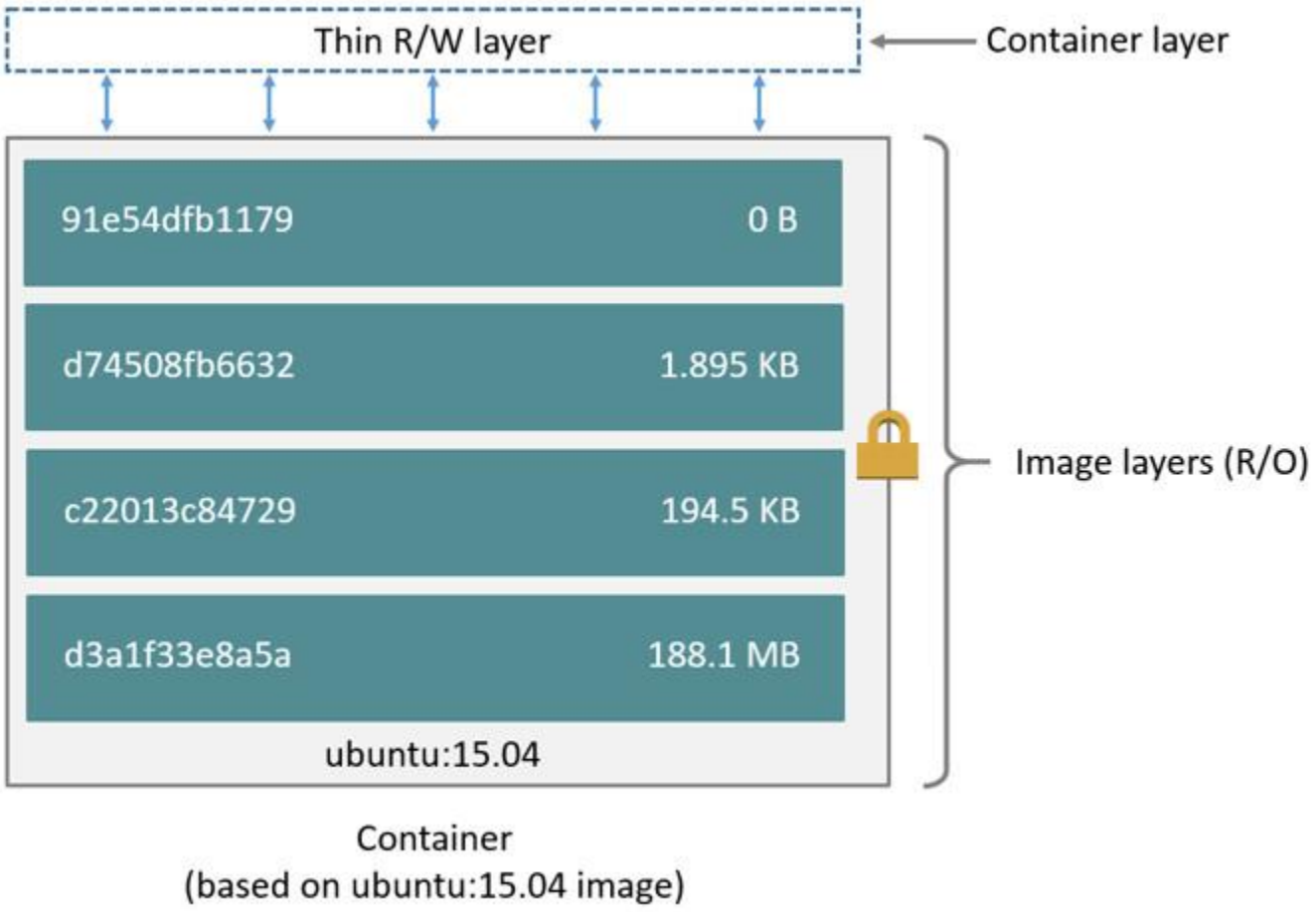
Docker Official Images

an explicitly empty image, especially for building images "FROM scratch"

- 最常用的base镜像是各Linux发行版的Docker镜像，如ubuntu、centos等。

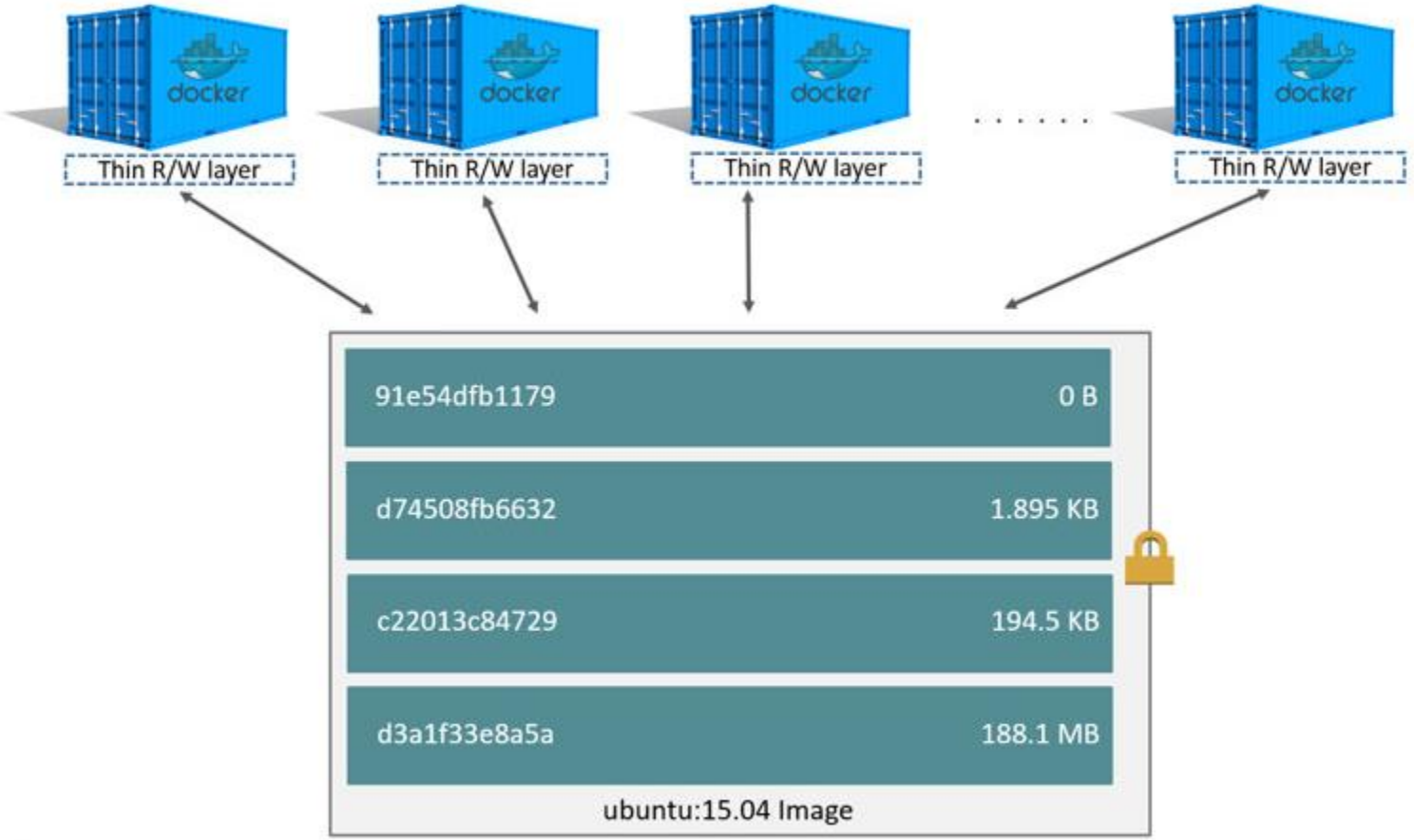
容器镜像分层结构 (1)

- Docker镜像中引入层layer的概念。镜像制作过程中的每一步操作，都会生成一个新的镜像层。



容器镜像分层结构 (2)

- 容器由若干只读镜像层和最上面的一个可写容器层构成。
  - 分层结构使镜像共享、容器创建、分发非常高效。



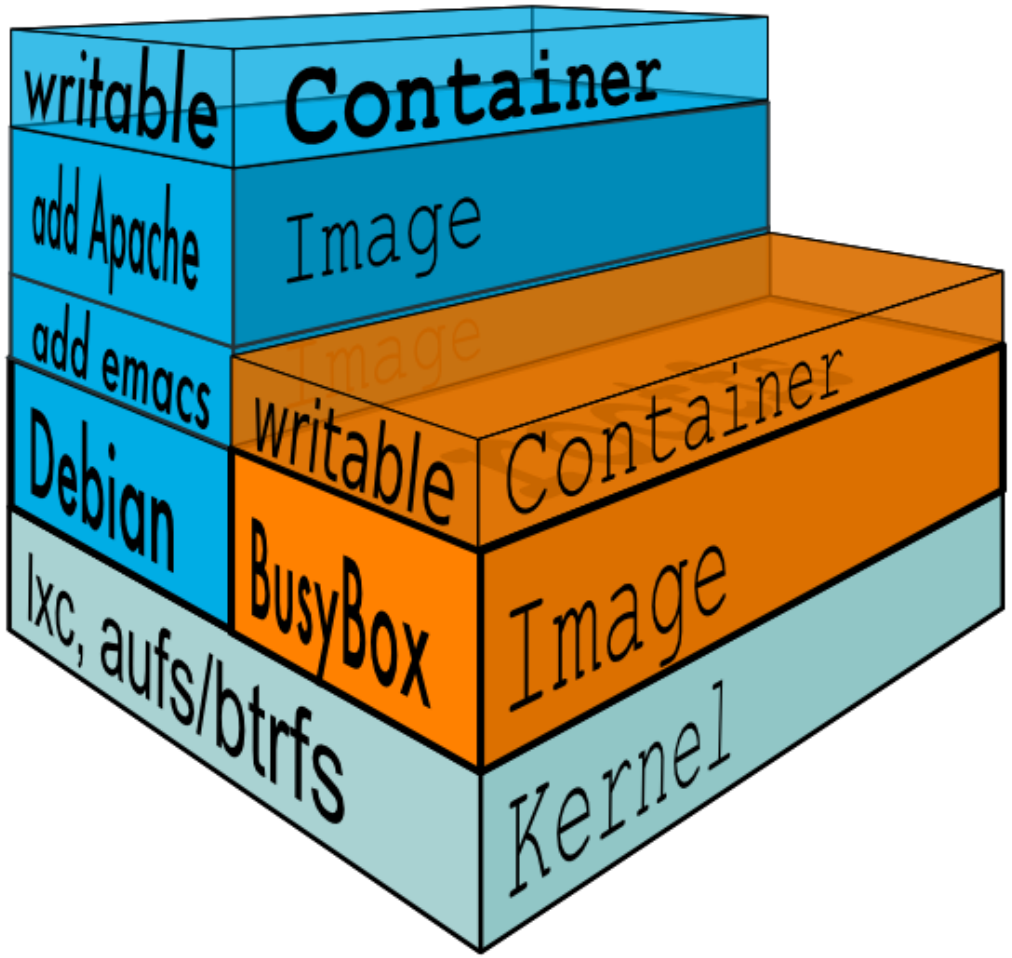
容器镜像分层结构 (3)

- 使用docker image inspect ubuntu命令查看ubuntu镜像分层结构，回显如下。


```
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/b58ead1ad37d8f65289950a10a0f7695f139cdf3a4f2cde9b7b2cc760f3bd994/diff:/var/lib/docker/overlay2/2ba641f16bc40c
2f93bcbd9e683e4610100562dc17cc3798346ef3c54b188293/diff:/var/lib/docker/overlay2/9ae72f65b20f3be310476c8f229f319667503ea3c854db925ded20cd4be99026/diff",
                "MergedDir": "/var/lib/docker/overlay2/672cc7f0b5a10a43796a688d06abe43d807177a266bfb45147bfd74876095706/merged",
                "UpperDir": "/var/lib/docker/overlay2/672cc7f0b5a10a43796a688d06abe43d807177a266bfb45147bfd74876095706/diff",
                "WorkDir": "/var/lib/docker/overlay2/672cc7f0b5a10a43796a688d06abe43d807177a266bfb45147bfd74876095706/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:543791078bdb84740cb5457abbea10d96dac3dea8c07d6dc173f734c20c144fe",
                "sha256:c56e09e1bd18e5e41afb1fd16f5a211f533277bdae6d5d8ae96a248214d66baf",
                "sha256:a31dbd3063d77def5b2562dc8e14ed3f578f1f90a89670ae620fd62ae7cd6ee7",
                "sha256:b079b3fa8d1b4b30a71a6e81763ed3da1327abaf0680ed3ed9f00ad1d5de5e7c"
            ]
        },
```

UnionFS联合文件系统

- UnionFS主要的功能是将多个不同位置的目录联合挂载（union mount）到同一个目录下。
  - 每一个镜像层都是Linux操作系统文件与目录的一部分。在使用镜像时，docker会将所有的镜像层联合挂载到一个统一的挂载点上，表现为一个完整的Linux操作系统供容器使用。



容器copy-on-write特性

- 对容器的增删改查操作：

操作	具体执行
创建文件	新文件只能被添加在容器层中。
删除文件	依据容器分层结构由上往下依次查找。找到后，在容器层中记录该删除操作。 具体实现是，UnionFS会在容器层创建一个”whiteout”文件，将被删除的文件“遮挡”起来。
修改文件	依据容器分层结构由上往下依次查找。找到后，将镜像层中的数据复制到容器层进行修改，修改后的数据保存在容器层中。（copy-on-write）
读取文件	依据容器分层结构由上往下依次查找。

目录

HUAWEI

docker commit构建镜像

- docker commit命令：可将一个运行中的容器保存为镜像。其运行过程可总结如下：
  - 运行一个容器
  - 修改容器内容
  - 将容器保存为镜像

docker commit示例 (1)

- 运行一个centos容器中运行/bin/bash。

```
[root@localhost ~]# docker run -d centos /bin/bash
c0df165d496baf1e19c5582509bac1627ab427ecb77ffc0bd71d9cf4146a402b
```

- 使用docker commit命令将该容器保存为镜像“test1”。

```
[root@localhost /]# docker commit c0df165d496b test1
sha256:ed67331067d33d0f901f9365cb28e51c9a86ed0fe32f0ae8003a9ae748b1df0d
```

- 使用docker images查看新的镜像。

```
[root@localhost ~]# docker images
REPOSITORY      TAG             IMAGE ID       CREATED         SIZE
test1           latest          ed67331067d3   4 minutes ago   202MB
centos          latest          9f38484d220    4 months ago    202MB
```

- 以test1镜像运行一个容器。

```
[root@localhost ~]# docker run -d test1
9053281824cedd88869caead4d531f208838f49b0b68937ba300ea5832438510
```

docker commit示例 (2)

- 使用docker history命令查看镜像构建历史。对比镜像test1与centos，test1比centos多了一个镜像层。

```
[root@localhost ~]# docker history test1
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
ed67331067d3        4 minutes ago       /bin/bash                                       0B
9f38484d220f        4 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           4 months ago        /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>           4 months ago        /bin/sh -c #(nop) ADD file:074f2c974463ab38c…   202MB
[root@localhost ~]# docker history centos
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
9f38484d220f        4 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           4 months ago        /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>           4 months ago        /bin/sh -c #(nop) ADD file:074f2c974463ab38c…   202MB
```

Dockerfile

- DockerFile：文件指令集，描述如何自动创建Docker镜像。
  - Dockerfile是包含若干指令的文本文件，可以通过这些指令创建出dokcer image。
  - Dockerfile文件中的指令执行后，会创建一个个新的镜像层。
  - Dockerfile文件中的注释以“#”开始。
  - Dockerfile一般由4部分组成：
    - 基础镜像信息
    - 维护者信息
    - 镜像操作指令
    - 容器启动指令
- build context：为镜像构建提供所需的文件或目录。

Dockerfile常用指令

指令	作用	命令格式
FROM	指定base镜像	FROM <image>:<tag>
MAINTAINER	注明镜像的作者	MAINTAINER <name>
RUN	运行指定的命令	RUN <command>
ADD	将文件、目录或远程URLs从<src>添加到镜像文件系统中的<dest>	ADD [--chown=<user>:<group>] <src>... <dest>
COPY	将文件或目录从<src>添加到镜像文件系统中的<dest>	COPY [--chown=<user>:<group>] <src>... <dest>
ENV	设置环境变量	ENV <key> <value>
EXPOSE	指定容器中的应用监听的端口	EXPOSE <port> [<port>/<protocol>...]
USER	设置启动容器的用户	USER <user>[:<group>]
CMD	设置在容器启动时运行指定的脚本或命令	CMD command param1 param2
ENTRYPOINT	指定的是一个可执行的脚本或者程序的路径	ENTRYPOINT command param1 param2
VOLUME	将文件或目录声明为volume，挂载到容器中	VOLUME ["/data"]
WORKDIR	设置镜像的当前工作目录	WORKDIR /path/to/workdir

Dockerfile示例 (1)

- 编辑一个dockerfile文件，并命名为dockerfile1，存放于/root/df目录下。

```
FROM httpd
COPY index.html /
RUN /bin/bash -c ”echo Huawei”
```

- 使用docker build命令用文件dockerfile1构建一个镜像，并将镜像命名为test2。

```
[root@localhost df]# docker build -t test2 .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM httpd
 ---> ee39f68eb241
Step 2/3 : COPY index.html /
 ---> acc0347202f8
Step 3/3 : RUN /bin/bash -c "echo Huawei"
 ---> Running in 0ffeeffbffa9
Huawei
Removing intermediate container 0ffeeffbffa9
 ---> 4a02266d7689
Successfully built 4a02266d7689
Successfully tagged test2:latest
```

Dockerfile示例 (2)

容器镜像缓存特性 (1)

- Docker会缓存已有镜像的镜像层，构建或下载镜像时，如果某镜像层已存在，则直接使用无须重新创建或下载。
- 编译dockerfile1文件，增加在末尾声明作者的指令。

```
FROM httpd
COPY index.html /
RUN /bin/bash -c ”echo Huawei”
MAINTAINER Michael@Huawei.com
```

- 重新执行docker build构建镜像，新镜像命名为test3。

```
[root@localhost df]# docker build -t test3 .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM httpd
 ---> ee39f68eb241
Step 2/4 : COPY index.html /
 ---> Using cache
 ---> acc0347202f8
Step 3/4 : RUN /bin/bash -c "echo Huawei"
 ---> Using cache
 ---> 4a02266d7689
Step 4/4 : MAINTAINER Michael@Huawei.com
 ---> Running in 868a18545c90
Removing intermediate container 868a18545c90
 ---> ab75aa7e6450
Successfully built ab75aa7e6450
Successfully tagged test3:latest
```

容器镜像缓存特性 (2)

- 若交换dockerfile文件中命令执行顺序，虽然在逻辑上这种改动对镜像的内容没有影响，但由于分层的结构特性，Docker必须重建受影响的镜像层。

```
FROM httpd
MAINTAINER Michael@Huawei.com
COPY index.html /
RUN /bin/bash -c ”echo Huawei”
```

- 重新构建镜像test3，每一步均生成新的镜像层，未使用到cache。

```
[root@localhost df]# docker build -t test3 .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM httpd
 ---> ee39f68eb241
Step 2/4 : MAINTAINER Michael@Huawei.com
 ---> Running in dfc74ef69ed4
Removing intermediate container dfc74ef69ed4
 ---> d2019843b44b
Step 3/4 : COPY index.html /
 ---> 25216f9ba9a0
Step 4/4 : RUN /bin/bash -c "echo Huawei"
 ---> Running in 2395902cf3df
Huawei
Removing intermediate container 2395902cf3df
 ---> d0c020ed268f
Successfully built d0c020ed268f
Successfully tagged test3:latest
```

镜像命名

- 镜像名称格式
  - image name = repository:tag
  - tag一般用于描述镜像版本。若未指定tag，则默认为”latest”

```
[root@localhost ~]# docker tag httpd httpd:v8.6
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              3556258649b2        2 weeks ago         64.2MB
busybox             latest              db8ee88ad75f        2 weeks ago         1.22MB
httpd               v8.6                ee39f68eb241        3 weeks ago         154MB
httpd               latest              ee39f68eb241        3 weeks ago         154MB
```

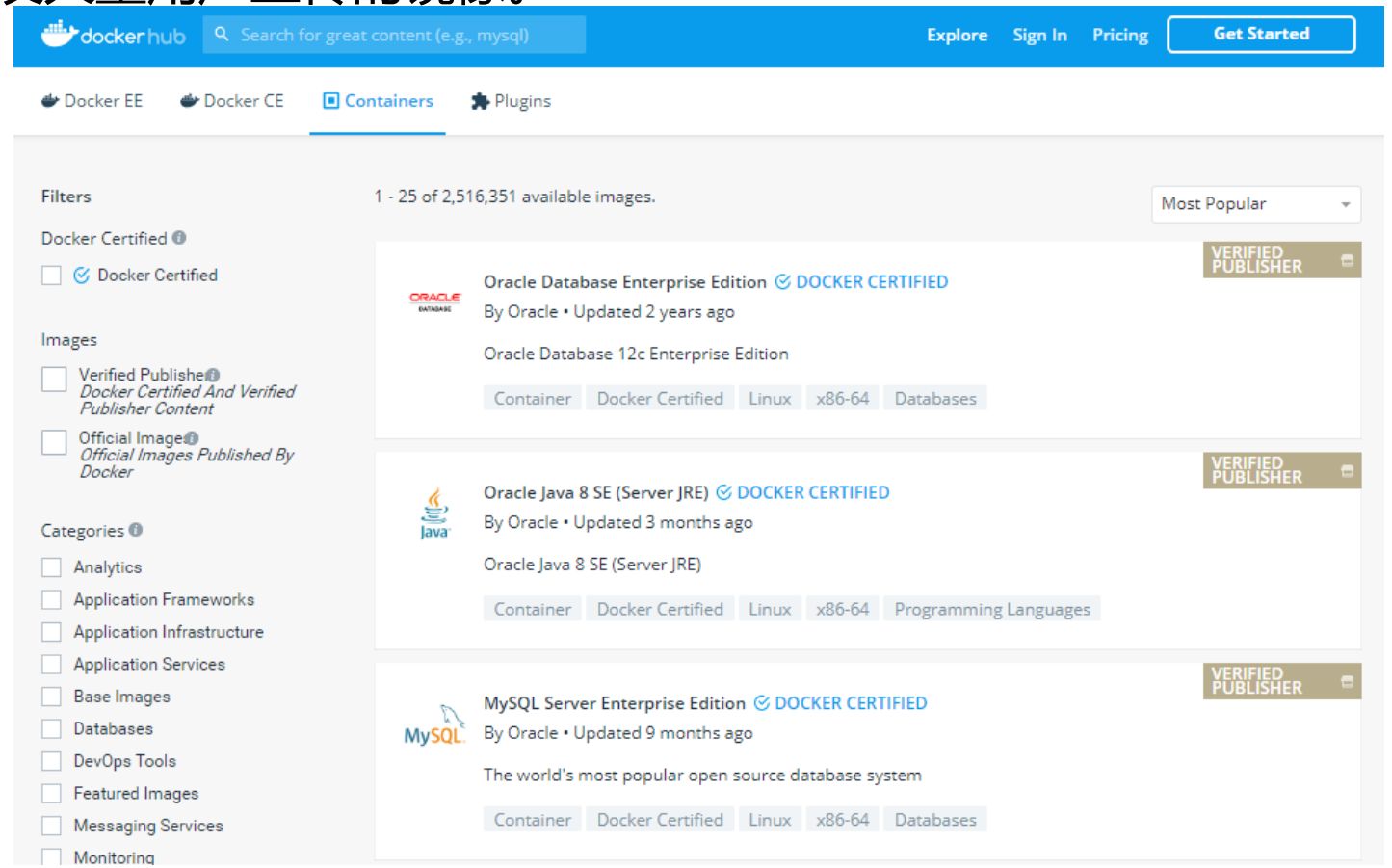
Registry

- Registry是存放容器镜像的仓库，用户可进行镜像下载和访问，分为公有和私有两类Registry。
- 公有镜像仓库：
  - Docker Hub是Docker公司为公众提供的托管Registry。
  - Quay.io现为Red Hat下的公共托管Registry。

- 私有镜像仓库：
  - 企业可以用Docker Registry构建私有的Registry。
    - Registry本身是一个开源项目，可以用于搭建私有Registry。

Docker Hub

- Docker Hub是目前世界上最大的容器镜像仓库，由Docker公司维护。上面有Docker公司提供的镜像及大量用户上传的镜像。



搭建私有Registry示例 (1)

- 使用“registry”镜像构建本地镜像仓库。registry是docker hub上维护的镜像,，其服务端口是5000。-v参数将宿主机的/root/myregistry目录映射到容器的/var/lib/regisgtry目录，用于存放镜像数据。

```
[root@localhost ~]# docker run -d -p 1000:5000 -v /root/myregistry:/var/lib/registry registry
eba841f45ea5f981b7a6757da1d2c4ba07834adebb74575888366f13e89588b5
```

- 使用docker tag命令修改镜像名称，使其符合registry上的格式要求。若要将镜像上传到registry，镜像名称需要符合其命名格式要求：[Registry-host]:[port]/[username]/[repository:tag]

```
[root@localhost ~]# docker tag httpd:v8.6 192.168.137.99:1000/michael/httpd:v8.6
```

- 上传容器镜像到私有Registry。

```
[root@localhost ~]# docker push 192.168.137.99:1000/michael/httpd:v8.6
The push refers to repository [192.168.137.99:1000/michael/httpd]
635721fc6973: Pushed
bea448567d6c: Pushed
bfaa5f9c3b51: Pushed
9d542ac296cc: Pushed
d8a33133e477: Pushed
v8.6: digest: sha256:f2179b693cfb49baa6e7500171deea7bef755338bf165b39aedacf2b4ae28455 size:
1367
```

搭建私有Registry示例 (2)

- 在Linux下，若使用如”192.168.137.99:1000”这样的内网地址作为私有仓库地址，则需要在/etc/docker/daemon.json中写入如下内容（如果文件不存在需新建该文件），然后重启Docker服务。

```
[root@localhost ~]# vi /etc/docker/daemon.json

{"insecure-registries": ["192.168.137.99:1000"]}
```

知识小考

- 镜像打包时的kernel与运行容器的宿主机kernel是否可以不一致？又或镜像的发行版是centos，能运行在host为ubuntu的宿主机上吗?

- 如果容器镜像是在4.x版本的kernel中创建的，而宿主机是3.x版本的kernel，容器是否能正常运行？为什么?

实验&实训任务

- 实验任务
  - 请按照实验手册1.3部分完成容器镜像实验。
- 实训任务
  - 请灵活使用本章节课程及实验手册中学到的知识，按照实验手册1.3.4章节完成容器镜像实训任务。

思考题

1. Dockerfile中的每个RUN指令执行后，都会生成一个对应的镜像层。T or F
2. 容器镜像中的镜像层，实际上是rootfs的一部分，也就是Linux操作系统文件与目录的一部分？T or F
3. 通过docker commit构建镜像时，容器的hostname等信息，会被添加到新的镜像中。T or F

本章总结

- 介绍了docker镜像分层结构，联合文件系统
- 介绍了docker镜像缓存特性
- 构建镜像的两种方法：docker commit命令与Dockerfile
- Docker Hub与本地docker镜像仓库的操作与使用


谢 谢
www.huawei.com