[toc]

课程

三.docker核心技术

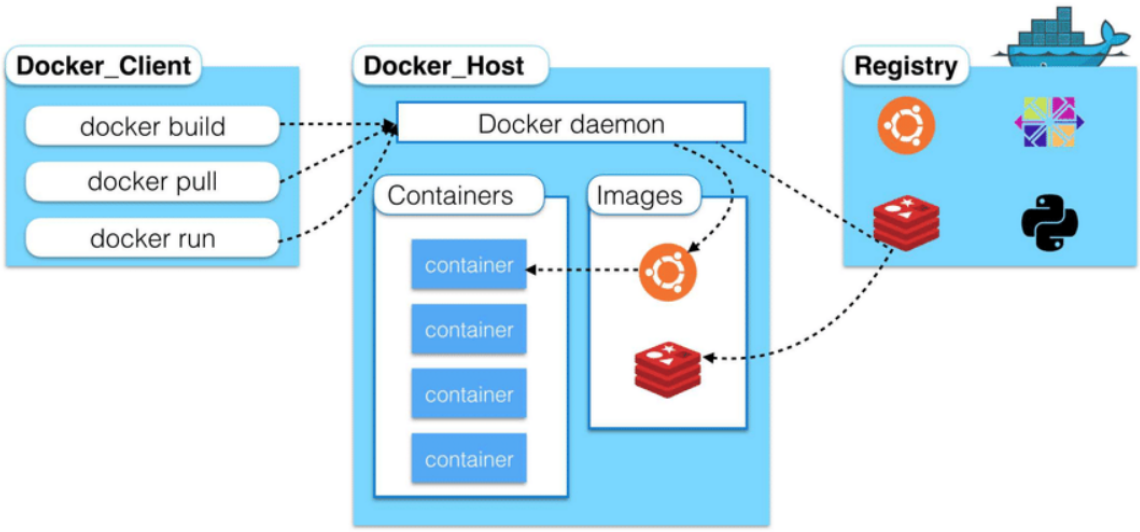
docker和虚拟机的区别

- 虚拟机:需要安装操作系统，更大，较于容器性能弱
- 容器:共享bin lib等文件系统，共享host OS，更小，性能好

docker架构

docker是c/s架构，分以下几点：

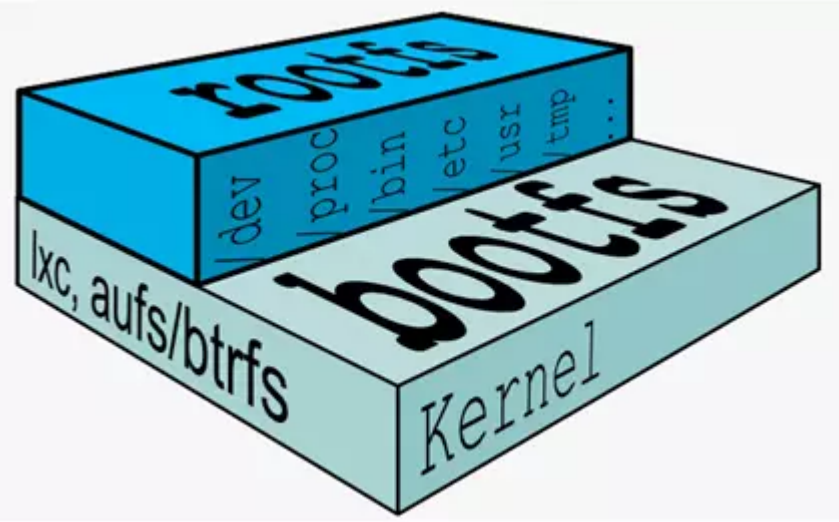
- client，执行 build pull run等命令
- server
  - docker daemon，运行在docker host上
  - image，docker daemon build出来一个image
  - Container，docker daemon利用image run出一个container
- registry，docker daemon push image到registry



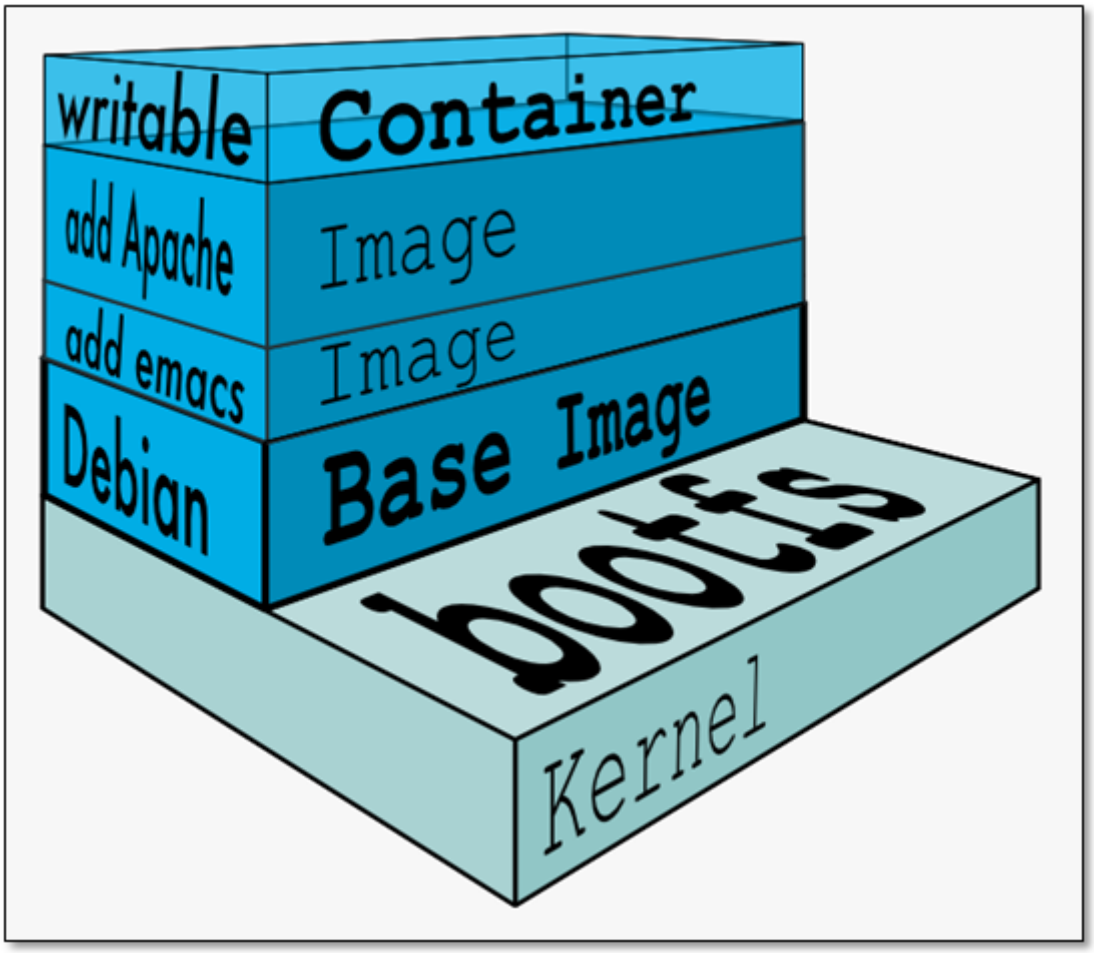
docker image

docker image的结构

- linux操作系统的结构分为两部分
  - boot fs：作为linux的kernel，不同的linux发行版往往kernel相同
  - root fs：包含/etc /bin等文件夹，不同的linux发行版区别就在这里



- docker的镜像结构也分为两层
    - 容器层：这一层writable，主要作用是为增量内容服务，保证容器的隔离性。这是docker的copy on write特性，所有的修改内容在容器层发生。这一层在镜像启动的时候加载的，所以基于一个镜像启动的多个容器可以自由修改里面的内容。
    - 镜像层：read only。docker file加载的不同命令就会往上累加镜像层。



这种分层结构的特点是上层覆盖下层，即union file system。

docker image的构建

两种方式：

- docker commit（不推荐）：即在容器里修改文件，在新窗口docker commit保存为新镜像。
- docker file（推荐）：撰写docker file，然后docker build。build过程中会启动临时容器作出指令后销毁。

docker file中体现的的几个概念

- build context：在docker build的时候，当前工作文件夹为build context，在这个目录下docker会自动查找docker file。所以切记要 -f 目标docker file，防止在目标空间下花费大量时间搜索docker file。

```
docker build –f ./Dockerfile
```

- 缓存特性：在docker这种分层结构中，老镜像层可以被缓存缩短build时间。上层依赖下层的缓存，如果下层缓存失效那么上层缓存必失效。

- Multi-stage build（多阶段构建）：docker老版本（17.05之前）不支持多阶段构建，为了避免dockerfile臃肿/层次深的问题，一般采用写多个docker file+脚本管理的方式。存在多阶段构建后，一个docker file中可以写一阶段的FROM xx AS xx，二阶段COPY --from=...

```
# syntax=docker/dockerfile:1
FROM golang:1.16 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get –d –v golang.org/x/net/html
COPY app.go    ./
RUN CGO_ENABLED=0 GOOS=linux go build –a –installsuffix cgo –o app .

FROM alpine:latest
RUN apk ––no-cache add ca-certificates
WORKDIR /root/
COPY ––from=builder /go/src/github.com/alexellis/href-counter/app ./
CMD ["./app"]
```

docker file常用指令

> from：指定base镜像
>
> maintainer：指定作者
>
> copy：从host copy一个文件到镜像
>
> add：与copy类似，较于copy 支持url的下载
>
> env：设置环境变量
>
> expose：暴露端口
>
> volume：声明某个文件夹挂载到宿主机的硬盘

> workdir：指定工作目录，配合add copy cmd entrypoint指令使用
>
> run：常用于apt-install等安装指令，常配合&&使用。一是为了避免多条run指令新增镜像层，二是为了避免分层后apt-install的工具更新，但老layer缓存了导致apt-install失效
>
> cmd：多条cmd会被覆盖，并且cmd会被docker run -it后接的命令行覆盖。常用于为entrypoint传递参数
>
> entrypoint：多条entrypoint会被覆盖，但entrypoint不会被docker run -it后接的命令行覆盖，常常搭配cmd使用于一个应用程序或服务的启动。
>
> Label:配合docker images -f label=...筛选镜像

docker image相关常用命令

> images：list所有的image
>
> history：显示image的构建历史
>
> commit：将当前容器保存为新镜像
>
> build：配合dockerfile打包镜像
>
> tag：打tag
>
> pull：从registry拉取镜像
>
> push：推送到registry
>
> rmi：在local（即host）删除一个镜像
>
> search：在docker hub查找一个镜像

docker container

容器运行

- 进入容器的两种方法
  - exec：开启一个新终端进程，常用于进去干点什么事
  - attach：沿用他本身的终端进程，常用于查看
- 启动容器的命令
  - ```
    CMD/ENTRYPOINT/docker run
    ```
  - 两个命令行参数
    - -it：交互方式启动
    - -d：后台启动
  - 指定容器的三种途径
    ```

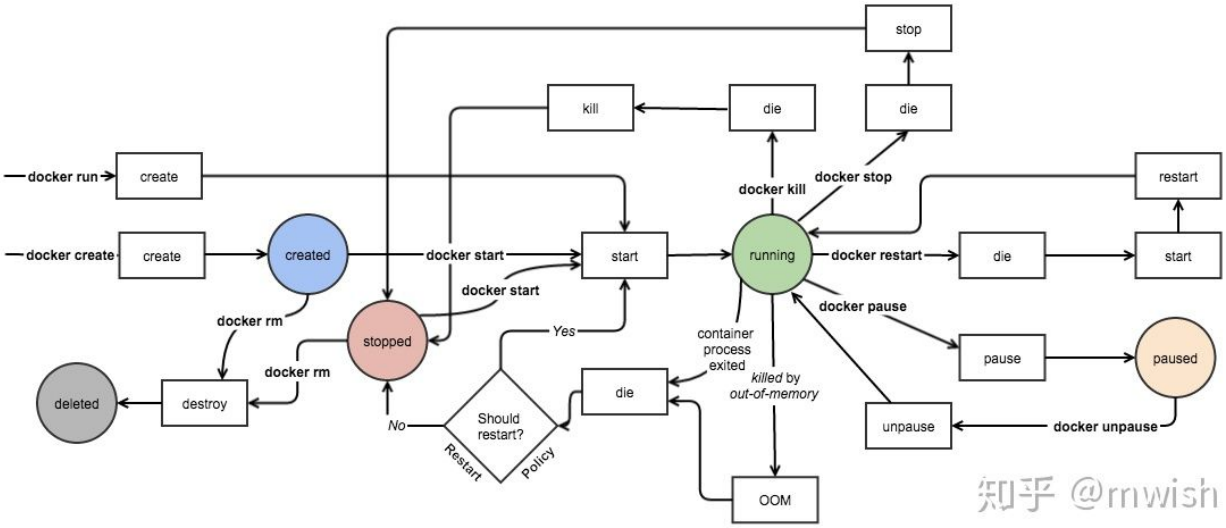
    - 长container id
    - 短container id
    - name（此name在docker run的时候可以指定name）

容器状态与启停

- 启停
    - stop：合法停止，给容器进程发送sigterm信号
    - kill：给容器进程发送sigkill信号
    - start：对于created/stop状态的容器，启动，docker create+start和docker run都可以让容器到started状态
    - restart：相当于stop+start
- 暂停
    - pause：常用于容器需要暂停打个快照（docker export）的场景，此时不占用CPU
    - unpause：从pause恢复
- 删除
    - ```
      docker rm ...
      ```

ps：**rm**是删除容器，**rmi**是删除镜像

- state machine



    - 只有oom造成的die会判断是否需要restart，kill 或 stop则不会

container的资源限制

- 内存限额
      ```

    - -m/--memory，指定内存使用额度
    - --memory-swap，指定内存+swap使用额度

```
docker run -m 200m --memory-swap=300m ubuntu
```

swap在linux一般指硬盘上的swap分区，即一个资源（可能是启动的时候占用较大内存的进程）启动之后将不常用资源放入到磁盘的swap分区，提高资源的利用率，空出内存干其他事情。swap一般要指定大于等于划分的物理内存。将swap中的资源移动到内存叫做swap in，反之叫swap out。

- cpu限额
    - -c/--cpu-shares，指定cpu使用额度

```
docker run --name "container_a" -c 1024 ubuntu
docker run --name "container_b" -c 512 ubuntu
```

指定cpu权重为1024:512,即2:1，启动后在host机上top查看发现cpu使用率即2:1

- 磁盘限额，只对direct io有效（direct io即数据落盘前无cache，通常对硬盘的读写都是cache io）
    - -blkio-weight，指定读写磁盘的权重，类似于--cpu-shares

```
docker run -it --name container_a --blkio-weight 600 ubuntu
docker run -it --name container_b --blkio-weight 300 ubuntu
```

    - 限制bps（byte per second，每秒读写的数据量）和iops（io per second，每秒读写次数）
        - --device-read-bps，限制对某个空间读的bps
        - --device-write-bps
        - --device-read-iops，限制对某个空间读的iops
        - --device-write-bps

```
docker run -it --device-write-bps /dev/sda:30mb ubuntu
```

docker的底层技术

docker使用了cgroup（control group）来实现资源配额和namespace来实现资源隔离。

- cgoup

  前面提到的--device-read-bps就是配置cgroup，进入到/sys/fs/cgroup可以找到cgroup的存在，对于资源的配额可以进入到container id命名的文件夹下看到详细文本。

- namespace，存6种资源

namespace的作用就是host掌控唯一的全局资源，让容器们彼此隔离

- mount：让容器拥有file system，拥有自己的/目录
- UTS:让容器拥有自己的host name，默认是container短id
- IPC：让容器进程拥有自己的内存，以及实现了semaphore来通信
- PID：让容器进程拥有自己的一套pid，所有的容器下挂在dockerd进程下，容器内部的进程下挂在host pid=1的父进程下
- network：让容器拥有自己的独立网卡
- user：让容器拥有自己的用户组，host看不到容器内部的用户组

docker的网络

分为null/host/bridge

- None：不做任何网络配置，在容器不需要联网的存储密码等场景下使用
- host：重用host网络
  - 优点：使用便捷，容器对网络要求高的场景可以使用
  - 缺点：多容器环境下存在端口冲突
- bridge：用于多容器互联的场景，多容器桥接

  什么是桥接?

  > 家庭无线路由器一般有四个LAN口，这四个LAN口任意选取其中的两个，每个LAN口连接一台电脑A、B，两个LAN口就是**桥接关系**。通信的双方感觉不到桥接的存在，桥接对用户透明，仿佛A、B之间只有一根网线相连。通俗来讲，桥接可以看成一条网线。交换机/路由器既是网桥。

references

- kubernetes docs
- linux 命令查询