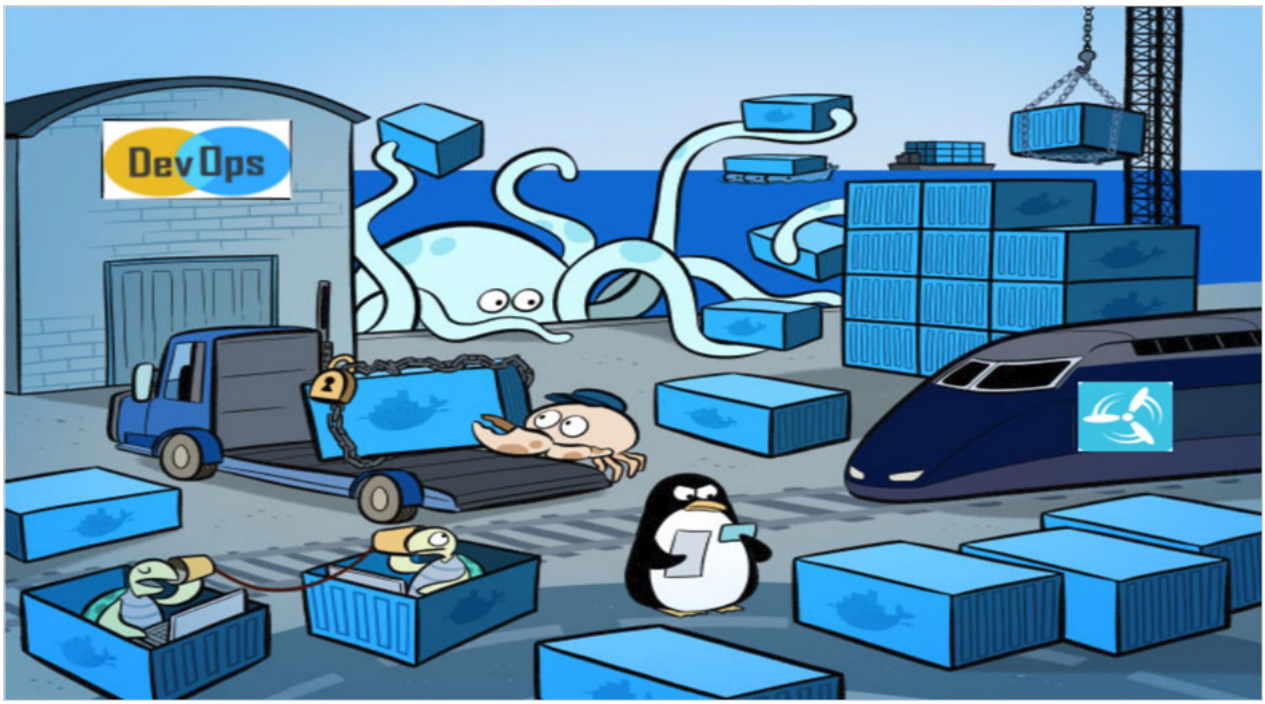
Docker基础教程

由 温兆钦创建, 最终由 张洋修改于大约10小时以前

1. Docker容器介绍

Docker是一个开源的容器引擎，可以让开发者把他的应用和依赖环境打包到一个可移植的容器环境中。

> 容器： 可以理解为一个轻量级的“虚拟机”，应用程序的运行环境。

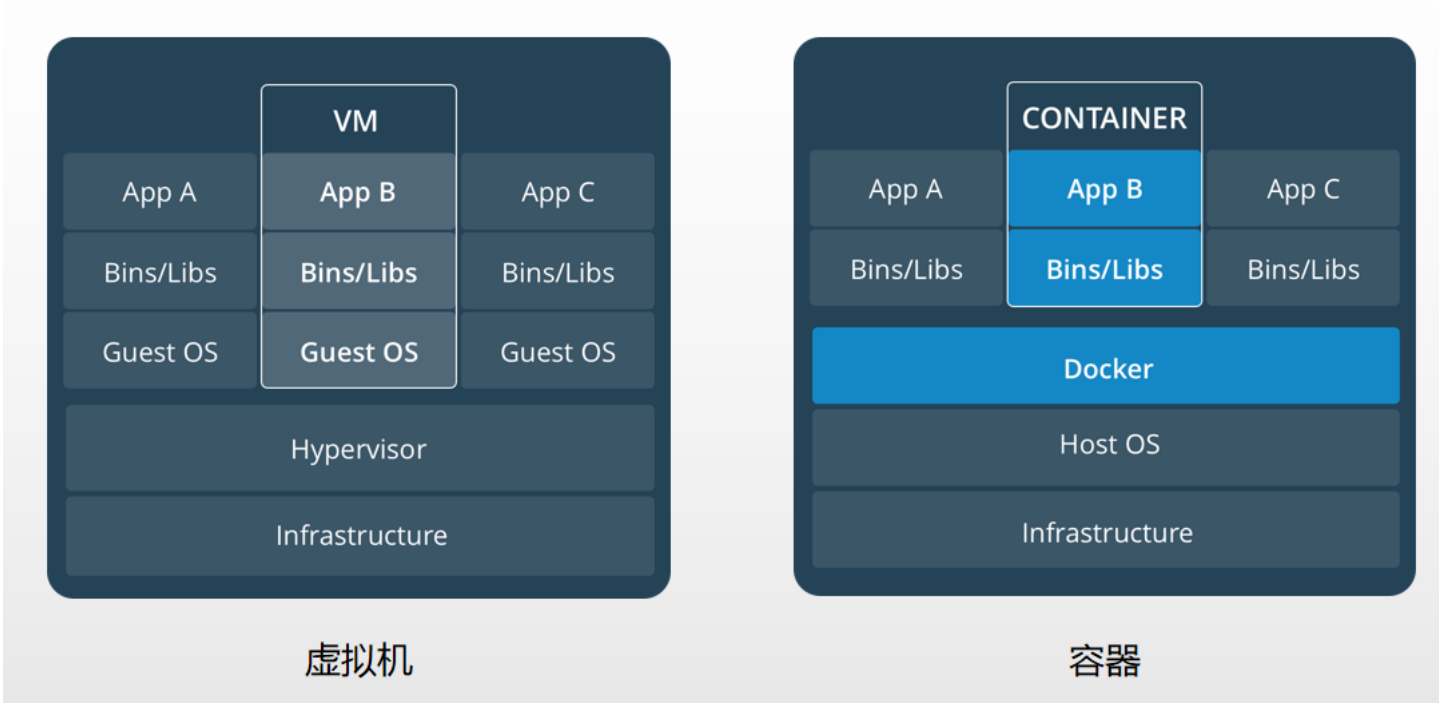


Docker的特点：

- 应用隔离
- 轻量级的虚拟化方案
- 扩展性，可以轻松扩展出成千上万的容器实例。
- 移植性，统一开发、测试、生产环境，可以在任意环境运行容器实例。

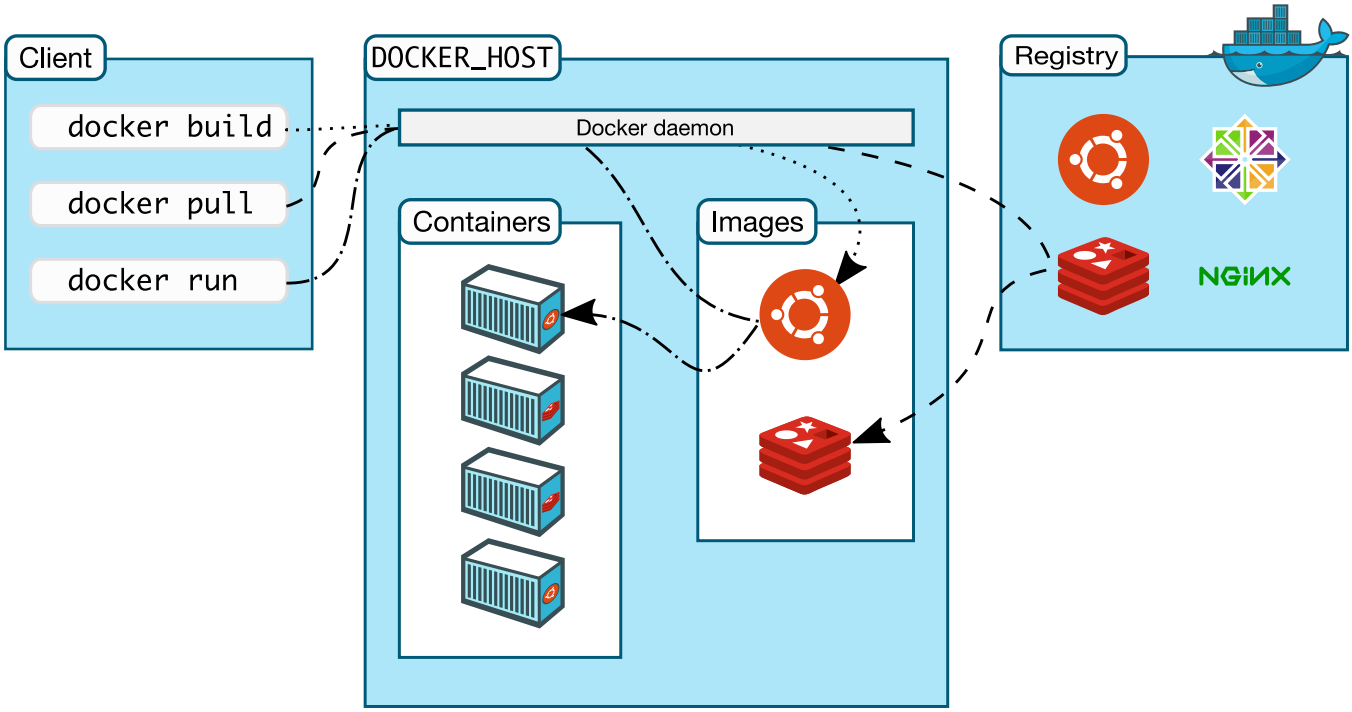
2. 容器与虚拟机的区别

虚拟机和容器的架构对比：



通过对比容器和虚拟机的架构图，虚拟机拥有独占的操作系统，虚拟化比较彻底，容器是共享操作系统通过资源隔离使得容器拥有独立的环境；虚拟机和容器都共享硬件资源。因此容器比较“轻”，启动一个容器可能就是一个进程，可以秒级启动容器。

3. Docker架构



主要包含下面几个部分：

- **Docker守护进程 (Docker daemon)**
  负责管理镜像、容器、容器网络、数据卷等。
- **Client**
  负责发送Docker操作指令，日常主要通过client完成镜像和容器的管理。
- **镜像 (Image)**
  即容器的模版，镜像是可以继承的，镜像主要通过Dockerfile 文件定义。
- **镜像仓库 (Registry)**
  类似git仓库，只不过镜像仓库用于存储镜像和管理镜像的版本。
- **容器**
  容器是通过镜像创建的，所以说容器是一个镜像运行的实例，类似面向对象编程中类和对象的关系。

4. 安装docker

下面以centos安装docker ce社区版本为例:

```
//先安装一些依赖的驱动和配置yum
$ yum install -y yum-utils \
  device-mapper-persistent-data \
  lvm2

$ yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

//安装docker
$ yum install docker-ce docker-ce-cli containerd.io

//启动docker
$ systemctl start docker

//测试下是否安装成功，这里运行一个hello-world镜像，正常的话会输出 Hello from Docker!
$ docker run hello-world
```

5. 容器基本用法

5.1. 快速入门

下面通过打包一个**静态网站**讲解容器的基本用法。
首先对于一个静态网站，我们需要什么样的环境才能跑起来？

对于静态网站，我们只需要一个Nginx软件就能跑起来，下面一步步讲解怎么通过容器部署这个网站。
下面是网站的项目目录结构：

```
├── Dockerfile       // Docker镜像定义文件
└── site             //网站代码目录，只是一个简单的静态网站，只有一个页面
    └── index.html
```

5.1.1. 定义镜像

创建镜像，我们首先需要定一个镜像，定义镜像的目的说白了就是**我们要在容器里面安装什么东西。**

下面是Dockerfile的定义：

> '#' 井号表示注释.

```
#FROM 指令表示要继承的镜像，这里继承nginx官方镜像，版本号是1.15.6
#定义镜像，都是通过继承某个已经存在的镜像开始
FROM nginx:1.15.6

#设置工作目录
WORKDIR /workspace
```

```
#/usr/share/nginx/html目录是nginx官方镜像的默认网站目录。
#复制site目录的内容，到/usr/share/nginx/html目录。
COPY ./site/* /usr/share/nginx/html

#删除掉/workspace的内容，因为上面的指令已经把网站内容复制到指定的目录，这里的内容现在没用了
RUN set -ex \
        && rm -rf /workspace

#从新设置工作空间
WORKDIR /usr/share/nginx/html

#声明容器需要暴露的端口
#EXPOSE 80
#容器启动命令，这里启动nginx，参数-g daemon off; 的意思是关闭nginx的守护进程模式。
#CMD ["nginx", "-g", "daemon off;"]

# EXPOSE 和 CMD命令都被注释掉了，因为nginx基础镜像已经定义过了，所以这里不需要重复定义，这里只是为了讲解目的，说明下容器是怎么运行nginx的。
```

5.1.2. 创建镜像

下面我们通过docker客户端创建镜像。

```
//先切换到项目的根目录
//通过docker build命令创建镜像
//格式：docker build -t 镜像名:版本号 上下文路径
//上下文路径的意思就是我们需要把那个目录上传到docker守护进程作为镜像创建的工作目录。
//这里把当前目录作为上下文环境目录
$ docker build -t demo1:v1.0.0 .

执行成功会输出如下结果：
Sending build context to Docker daemon  3.584kB      //先打包上传上下文目录内容
Step 1/3 : FROM nginx:1.15.6
1.15.6: Pulling from library/nginx          //下载nginx镜像
a5a6f2f73cd8: Already exists
67da5fbcb7a0: Pull complete
e82455fa5628: Pull complete
Digest: sha256:31b8e90a349d1fce7621f5a5a08e4fc519b634f7d3feb09d53fac9b12aa4d991
Status: Downloaded newer image for nginx:1.15.6
 ---> e81eb098537d      //下面开始执行我们Dockerfile定义的命令
Step 2/3 : WORKDIR /usr/share/nginx/html
 ---> Running in 155da3e24b72
Removing intermediate container 155da3e24b72
 ---> 608d1e7fc6cd
Step 3/3 : COPY --chown=nginx:nginx . /usr/share/nginx/html
 ---> 894f22f15a7c
Successfully built 894f22f15a7c
Successfully tagged demo1:v1.0.0

//查看当前机器上的镜像，这个时候就看到刚才创建的demo1镜像
$ docker image ls
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
demo1                                       v1.0.0              894f22f15a7c        11 minutes ago      109MB
```

5.1.3. 启动容器/运行网站

启动容器的过程是： 根据镜像启动容器 -> 容器根据Dockerfile配置CMD命令启动我们应用。
下面启动demo1镜像，docker会创建一个容器运行nginx。

```
//通过docker run命令运行一个镜像
//命令格式：docker run -p 容器绑定的外部端口:容器内部端口 镜像:镜像版本
$ docker run -p 8080:80 demo1:v1.0.0
```

通过浏览器访问：http://localhost:8080 即可访问网站

5.2. Dockfile详解

下面介绍Dockfile常用的指令

5.2.1. FROM

FROM是Dockerfile 文件的第一条指令，标示要继承的镜像。

格式:
FROM image[:tag]

参数说明:
image : 镜像名或者镜像地址。
tag : 可选参数，镜像版本或者叫镜像标签。

例子:

```
#这里使用的是官方nginx镜像，标签为:1.15.6
FROM nginx:1.15.6

#这里使用的是完整的镜像地址，标签为：v1.3.0
FROM registry-vpc.cn-hangzhou.aliyuncs.com/lewaimai/lewaimai-base-env:v1.3.0
```

5.2.2. COPY

复制指令，主要用于复制目录和文件。
格式:
COPY [--chown=user:group] src dest

参数说明:
src : 待复制的文件或者目录
dest : 复制文件的目的地
--chown=user:group : 可选参数，设置复制文件或者目录的用户组

例子:

```
#复制./site/目录下面的所有文件，到/usr/share/nginx/html目录，同时设置文件的用户和组为nginx
COPY --chown=nginx:nginx ./site/* /usr/share/nginx/html
```

5.2.3. RUN

用于执行命令
格式:
RUN command

只有一个参数command， 就是我们想要执行的命令。

5.2.4. CMD

用于设置容器默认执行的命令。
格式:
CMD ["executable","param1","param2"]

参数说明:
executable : 执行程序，可以包含完整路径
param1, param2 ...param N : 执行程序的任意参数

5.2.5. ENV

设置容器环境变量
格式:
ENV = ...

例子:

```
#用空格分隔键值对
ENV myName="John Doe" myCat="fluffy"
```

5.2.6. USER

用于设置执行RUN, CMD 和 ENTRYPOINT命令时候的以什么用的身份运行。
格式:
USER user[:group]

参数说明:
user : 用户名
group : 可选参数，用户组

> 注意：如果设置的用户不存在，则以root用户身份运行。 例子:

```
#设置执行RUN、CMD、ENTRYPOINT的时候用户和组为www
USER www:www
```

5.2.7. WORKDIR

用于为RUN、CMD、ENTRYPOINT、COPY命令设置工作目录。

> 如果设置的工作目录不存在，则自动创建一个。

例子:

```
WORKDIR /path/to/workdir
```

5.3. .dockerignore文件

docker build命令在发送上下文目录的文件到docker守护进程之前，可以通过.dockerignore配置文件，设置需要过滤的文件或者目录。

> .dockerignore文件保存在上下文目录的根目录。

.dockerignore例子:

> .dockerignore配置语法类似.gitignore

```
# comment
#忽略所有目录下面的.log后缀的文件
**/*.log
#忽略data目录
/data

#忽略所有的.md结尾的文件，除了README.md
*.md
!README.md
```

6. 镜像管理

镜像管理问题:

- 如果我们为不同的项目创建了多个镜像，每个镜像又有多个版本怎么管理它们?
- 在服务器集群中要根据不同的镜像创建容器，去哪里找到镜像呢?

为管理镜像我们需要一个类似github的镜像仓库，Docker官方为我们提供了一个公开的镜像仓库。 官方仓库地址：https://hub.docker.com/, 在上面可以找到各种各样的镜像，类似nginx、redis、mysql这些知名的开源软件官方也到这里发布了自己的官方镜像。

> 提示：知名开源软件提供的官方镜像会有Official Image标记， 这些镜像会比较可靠，其他的镜像都是网友上传的。

下载镜像不需要账号密码，如果要发布自己的镜像就需要注册帐号。

对于个人使用公开镜像当然没问题，但是对于公司就不行，对于公司要么自己搭建一个私有的镜像仓库，要么使用阿里云这种云平台提供的镜像服务，创建一个私有仓库。

> 无论是自己搭建的镜像仓库还是使用公开的镜像仓库用法是一样的。

镜像仓库的用法类似git, 下面是镜像仓库的基本用法：

```
#登陆仓库命令：
#公开仓库不需要登录
$ docker login --username=帐号 仓库地址

#例子：
#登陆阿里云容器服务仓库
#回车后输入密码即可
docker login --username=123456@qq.com registry.cn-hangzhou.aliyuncs.com

#下载镜像，命令格式
docker pull 镜像:[镜像版本号]

#例子1：
#下载nginx最新镜像
docker pull nginx

#下载阿里云镜像仓库的镜像，镜像版本号为:v1.2.0
docker pull registry.cn-hangzhou.aliyuncs.com/lewaimai/lewaimai:v1.2.0

#上传镜像，命令格式
docker push 镜像:[镜像版本号]

#例子：
#推送镜像，之前docker login命令登陆到什么仓库，就推送到那里。
docker push registry.cn-hangzhou.aliyuncs.com/lewaimai/lewaimai:v1.2.0
```

7. 容器数据管理

7.1. 镜像分层技术

一个Docker镜像是有多个layer(层)组成的；每一层都存储一些文件数据，每一层都是在另一层的基础上进行CRUD操作，每一层只是记录相对前面几层的差异部分的文件数据，Docker镜像分层技术，很像git代码管理，git记录每一次代码文件的变化历史，每一次提交的版本都是上一个版本的差异部分。

> 注意：镜像每一层的文件数据都是只读的。

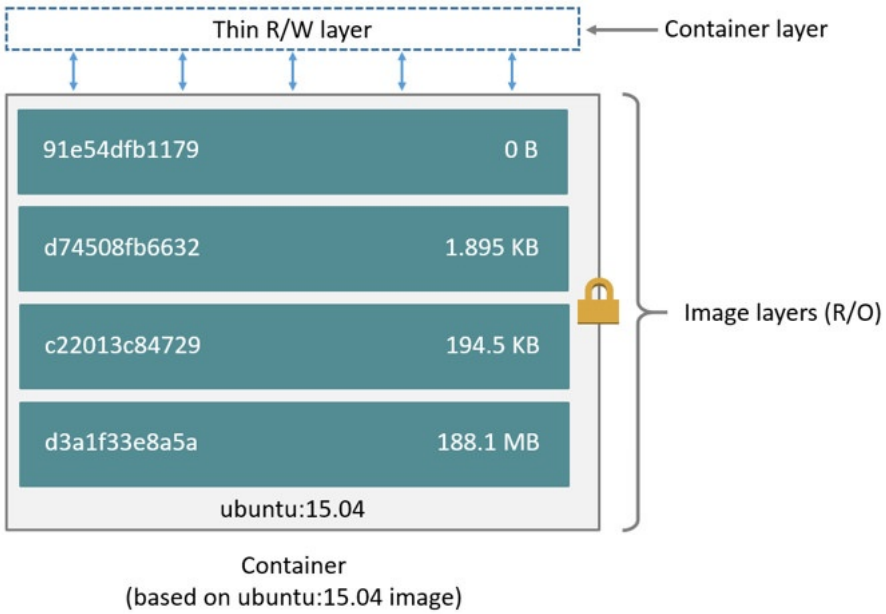
当创建一个容器的时候，会在顶层增加一层可写的容器层，我们容器的数据都是写在这上面。

> 容器层的数据不是持久化的，删除容器数据就丢了。

例子:
Dockerfile定义如下

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

dockerfile由4条指令构成，创建镜像后，分层示意图如下：



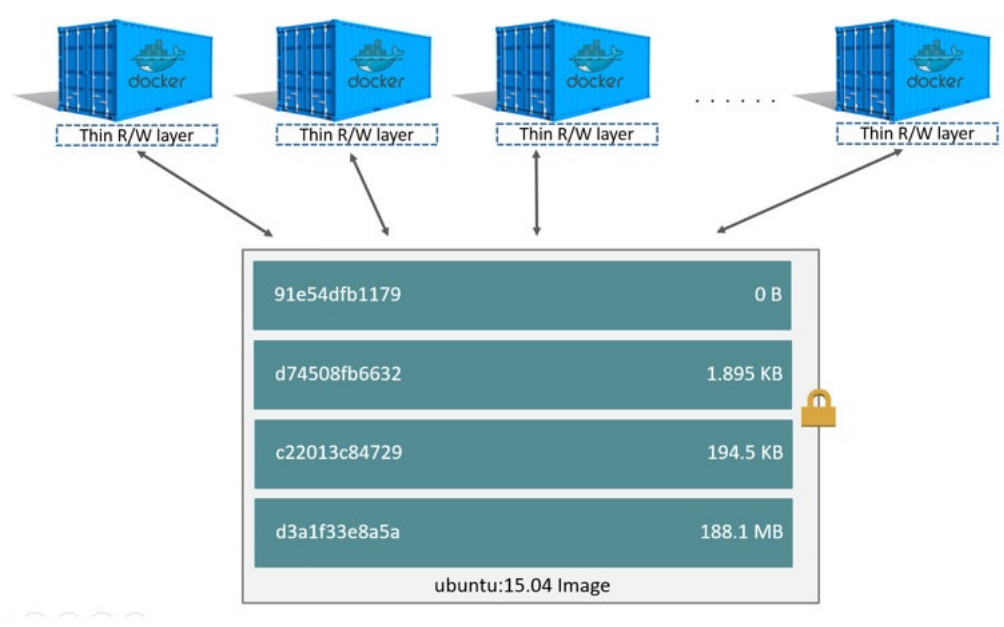
从底层往上看，最底下的一层保存dockerfile的FROM指令运行后产生的数据，倒数第二层对应COPY指令复制文件产生的数据，以此类推。

大家可以看到每一层都一个ID，例如最底下一层的id为: d3a1f33e8a5a, 很像git的提交记录吧。

当根据镜像创建容器的时候会在顶层增加一层容器层（Container layer），保存容器运行的数据。

> 提示：docker可以分层缓存数据，如果本地镜像已经下载过对应层的数据，那么docker不会重复下载对应的层；所以有些镜像看起来几百M，但是瞬间就下载完成，原因是分层缓存对应的数据。

下图是根据同一个镜像创建多个容器的情况：
每一个容器都有自己单独的容器层（Container layer），容器之间数据是独立的；但是他们底层的镜像数据是一样的。



7.2. 容器文件数据管理

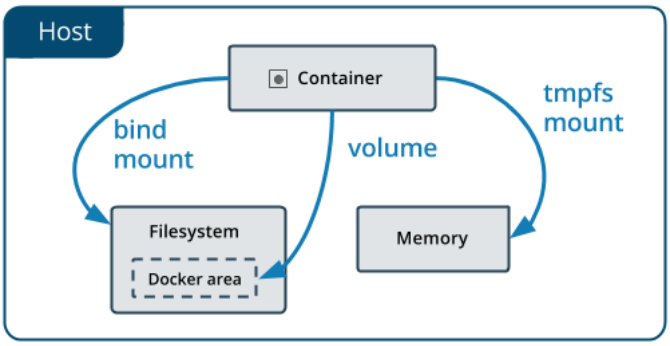
在Docker容器里面写文件数据，这些数据不是持久化的，随着容器被删除数据将会丢失，下面是Docker提供的容器文件数据管理方式。

主要支持下面三种方式管理文件数据：

- **Volumes**
  共享服务器上的某个目录，但是这个共享目录限制在/var/lib/docker/volumes/目录下面，由docker维护。
- **Bind mounts**
  共享服务器上的文件或者目录，相对于Volumes区别就是不限制目录，可以是系统任何位置。
- **tmpfs**
  这种方式文件数据保存在服务器内存中。

> 提示：无论使用那种方式管理容器文件数据，对容器内部来说，看起来跟本地文件系统一样。区别就是真实数据保存在什么地方。

容器管理文件数据示意图：



7.2.1. Volumes方式

使用Volumes方式管理文件数据需要**先创建volume数据卷，然后挂载到容器**里面。

例子：

```
//创建数据卷 demo1-data
$ docker volume create demo1-data

//查看当前机器创建的数据卷
$ docker volume ls
#输出
DRIVER              VOLUME NAME
local               demo1-data

//查看数据卷详情，可以查看数据具体保存在什么地方
$ docker volume inspect demo1-data
#输出
[
    {
        "CreatedAt": "2019-04-18T15:59:58+08:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/demo1-data/_data",
        "Name": "demo1-data",
        "Options": {},
        "Scope": "local"
    }
]

//删除数据卷
$ docker volume rm demo1-data

//将创建好的数据卷挂载到容器里面
$ docker run -d --name demo1 -v demo1-data:/mnt/data demo1:v1.0.0
```

```
参数说明：
-d ： 让容器在后台运行.
--name ： 给容器起个名字.
-v  ： 挂载数据卷, 格式: -v 数据卷名字:容器目录
demo1:v1.0.0 ： 镜像的名字
```

7.2.2. Bind mounts方式

任意目录或者文件挂载方式.
例子:

```
//一般我们本地开发启动nginx，配置文件都是在本地，我们希望容器可以读取本地的nginx配置。
//这个例子是把主机上的nginx配置文件挂载到nginx容器的配置文件里面，这里挂载的是文件。
$ docker run --name my-custom-nginx-container -v /host/path/nginx.conf:/etc/nginx/nginx.conf:ro -d nginx
参数说明：
--name my-custom-nginx-container ： 容器名字
-v /host/path/nginx.conf:/etc/nginx/nginx.conf:ro ： -v 挂载参数，将主机上的/host/path/nginx.conf文件挂载到容器的/etc/nginx/nginx.conf文件，rc

//除了上面的挂载文件，我们也可以挂载目录，接着上面的例子
$ docker run --name my-custom-nginx-container -v /host/path/nginx/conf:/etc/nginx/:ro -d nginx
参数说明：
-v /host/path/nginx/conf:/etc/nginx/:ro ：  这里把主机的/host/path/nginx/conf目录以只读方式挂载到容器的/etc/nginx/目录。
```

7.2.3. tmpfs方式

tmpfs主要是将数据存储在主机内存中。
例子:

```
$ docker run -d --name demo1 --tmpfs /mnt/data demo1:v1.0.0
参数说明：
--tmpfs /mnt/data ：  挂载主机内存空间到容器的/mnt/data目录
```

8. 容器互相通信

在实际应用的时候我们往往会启动多个容器，分别运行不同应用，又或者为了负载均衡启动大量的容器实例；那么容器之间怎么通信?

容器之间的通信方式跟服务器之间的通信方式差不多，除此之外在同一台主机的容器之间Docker支持直接互相连接的通信方式。
容器之间的通信方式大体上有两种：

- 基于tcp/ip的网络通信。
- 同一台主机上创建容器的时候，使用--link参数连接本地的其他容器。

例子1： 同一台主机上的容器之间通过link参数互联.

```
//启动一个redis server，容器名字叫：redis，redis其他参数配置默认
$ docker run -d --name redis

//另外启动一个容器，连接redis容器，这样在demo1容器内部可以使用redis名字当成redis服务器地址
$ docker run -d --name demo1 --link redis demo1:v1.0.0
参数说明：
--link ：连接容器，  格式：--link 容器名 ，连接多个容器，可以使用多个--link参数即可
```

例子2：主机怎么访问容器里面的应用?
例如，本地开发测试，我们使用容器运行应用，那么容器外面的主机怎么访问?

容器需要暴露容器内部的端口给主机，主机才能访问容器内部的应用。

```
//例如，我们使用nginx容器部署了一个网站应用，nginx容器需要暴露80端口出来。
$ docker run --name myapp -p 8080:80 -d nginx
参数说明：
-p  ： 绑定端口参数，格式 -p 主机端口:容器内部端口

主机端口可以任意选择，只要没被其他程序占用即可。
```

> 主机端口不一定要跟容器内部端口一致， 只是我们习惯用一样的端口，便于理解。

9. Docker常用命令

```
//查看正在运行的容器
$ docker ps

//启动容器  - 容器创建之后不需要再次使用docker run命令创建容器， 这个时候只要启动容器即可。
//启动容器名为redis的容器
$ docker start redis

//关闭容器
$ docker stop redis

//删除容器
$ docker rm redis

//容器的启动、关闭、删除除了使用容器名字之外，还可以使用容器id
//例如:
$ docker stop f648f34beabb

//执行容器里面的命令
$ docker exec -it redis bash
参数说明：
-it   ：  打开一个交互的终端。
redis  ：  容器的名字叫做redis
bash    ：执行容器里面的bash命令
//上面这条命令的其实就是连接容器打开一个shell窗口，类似登录服务器一样。
//  命令格式：  docker exec 容器名 要执行的命令

//删除镜像
```

```
docker image rm 镜像名

//给镜像打标签
docker tag 镜像id  标签
例子:
$ docker tag ImageId registry.cn-hangzhou.aliyuncs.com/lewaimai/dts:v2.0.1

//清理未使用的镜像
$ docker image prune
```

10. Docker实践建议。

10.1. 尽量减小镜像大小和layer。

不安装跟应用无关的软件，清理掉没有用的文件； 例如我们在编译安装软件的是最产生很多中间文件，安装成功后可以删除这些数据。
因为Docker镜像的层级是有限的，在Dockfile配置文件中FROM、COPY、RUN这些指令都会产生layer, 用的最多的主要是RUN指令，**推荐把连续的多条RUN指令合并成一条。**

例子:

```
FROM centos

COPY . /app
WORKDIR /app
#安装依赖包
#下面每一条RUN指令都会产生一个layer, 光是RUN指令就产生了6个layer
RUN yum -y install gcc automake autoconf libtool make gcc-c++
#编译安装
RUN ./configure
RUN make
RUN make install
#清理安装文件
RUN rm -rf /app
#清理yum 缓存
RUN yum clean all
```

优化后的Dockfile

```
FROM centos

COPY . /app
WORKDIR /app

#通过&&把多条命令连成一条命令，每行末尾的反斜杠是为了可读性，一行书写一条命令。
#这里只会产生一个layer
RUN yum -y install gcc automake autoconf libtool make gcc-c++ \
    && ./configure \
    && make \
    && make install \
    && rm -rf /app \
    && yum clean all
```

10.2. 避免在容器里面存储数据

因为在实际生产环境中容器是经常被销毁、重建的，可能是因为应用异常容器退出了，也可能是某台服务器负载太高，容器被调度到别的服务器运行； 当容器被销毁的时候存储在容器的数据会丢失。

生产环境中存储持久化数据一般有以下方案:

- 通过数据卷存储数据，一般可以选择把数据存储在主机服务器上、也可以存储在分布式文件系统上。
- 通过外部数据库、云存储、日志服务等可持久化的存储引擎或者第三方云服务存储数据。

10.3. 一个容器只跑一个应用

要保证容器的"轻"的特性，我们一般不会把容器当成服务器来用，什么东西都往里面安装。

例如：我们经常会在一台服务器上安装nginx、php、mysql、redis，同时部署一堆应用系统。
如果在容器这么干，升级维护就变得复杂了，而且不容易扩展，如果要升级redis，或者升级某个应用系统就得大家一起更新，而且还没法对里面的单个应用进行扩容。

一个容器跑一个应用，可以保证容器的轻量级特性，又相对独立，便于单独升级和扩容。

例如：一个PHP系统，需要nginx、 php fpm、 mysql、redis 那么镜像和容器可以这么设计。

镜像制作情况:

- nginx + php-fpm + PHP系统源码 制作一个镜像。
- redis 单独一个镜像。
- mysql 单独一个镜像。

容器情况:
根据上面的镜像制作情况，至少需要同时运行三个容器，上面的三个镜像分别创一个容器。

10.4. 为项目或者公司制作基础镜像

一个项目或者一个公司的基础环境一般都是一样的，没有必要每次都重新制作基础环境镜像。
例如一个项目的基础环境是: nginx + php-fpm + PHP扩展库(curl、kafka、redis、swoole)， 我们只要先根据环境需要制作一个基础镜像，项目只要继承这个基础镜像，然后设置项目自己的参数即可。

10.5. 容器应用主进程不能切换到后台运行。

容器通过CMD设置应用的启动命令，当容器启动应用的主进程后，应用主进程的生命周期就跟容器捆绑了，主进程退出，容器就退出了。

这个地方跟我们一般服务器部署应用不一样， 一般在服务器部署应用都是把进程切换到后台保持运行就可以，容器部署应用不能切换到后台运行，否则容器会任务应用结束了，容器自己就退出了。

> 创建容器的时候可以通过-d参数把容器切换到后台运行，例如: docker run -d --name redis redis

无标签