

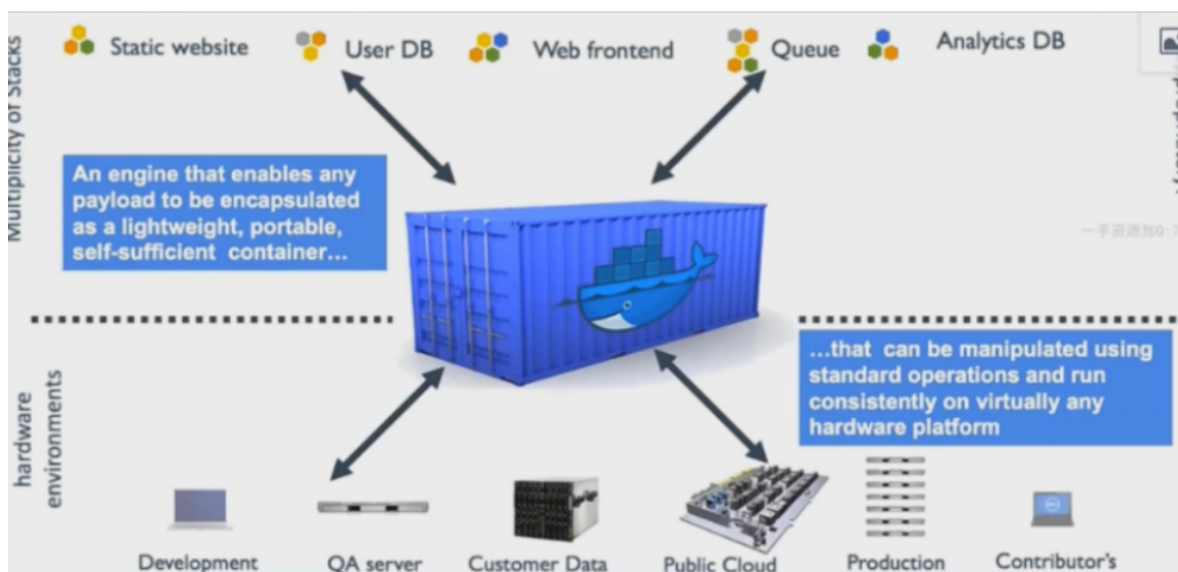
Docker容器化技术(上)

Docker容器化技术(上)

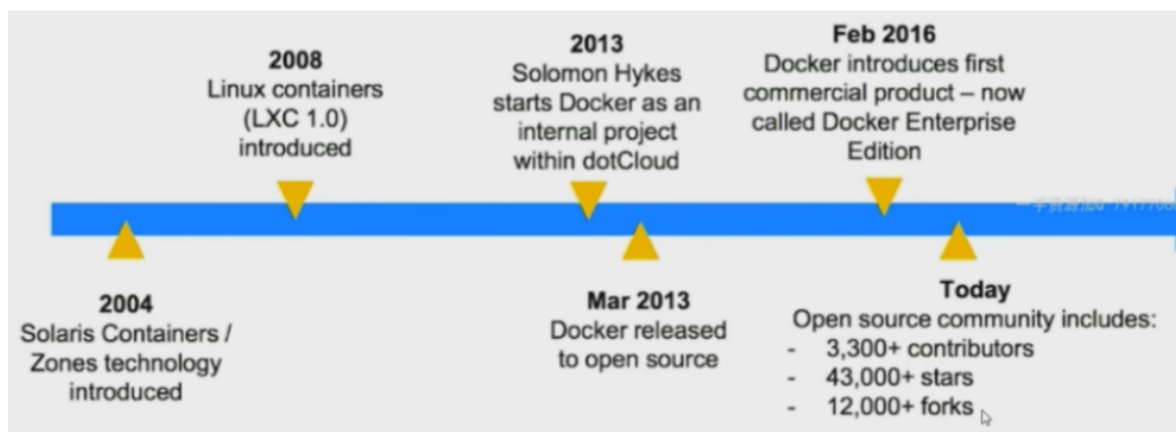
- 一、介绍
- 二、Docker的发展
- 三、Docker安装
- 四、阿里云Docker镜像加速
- 五、Docker的基本概念
- 六、命令
- 七、Docker宿主机与容器通信
- 八、容器内部结构
- 九、容器生命周期
- 十、Dockerfile构建镜像

一、介绍

- 开源的应用容器引擎，基于Go语言开发
- 容器是完全使用沙箱，容器开销极低
- Docker就是容器化技术代名词
- Docker也具备一定虚拟化职能
- 标准的应用打包



二、Docker的发展



三、Docker安装

1.安装utils

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```

后两个工具是持久化工具，yum-utils是省去安装源配置过程

2.yum-config-manager 是yum-utils的命令，如果没有，则会采用去复制repo到某目录，然后刷新yum（比较麻烦）

```
sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

3.让yum检测最快的源并使用

```
yum makecache fast
```

4.安装docker，ce是社区版本

```
yum -y install docker-ce
```

5.启动服务

```
service docker start  
或者  
systemctl start docker
```

6.查看docker版本

由于docker是CS架构，所以我们这次查看有Server和Client，在第四步安装，我们就已经安装了Server和Client

```
docker version
```

7.拉取hello-world

```
docker pull hello-world
```

8. 执行hello-world

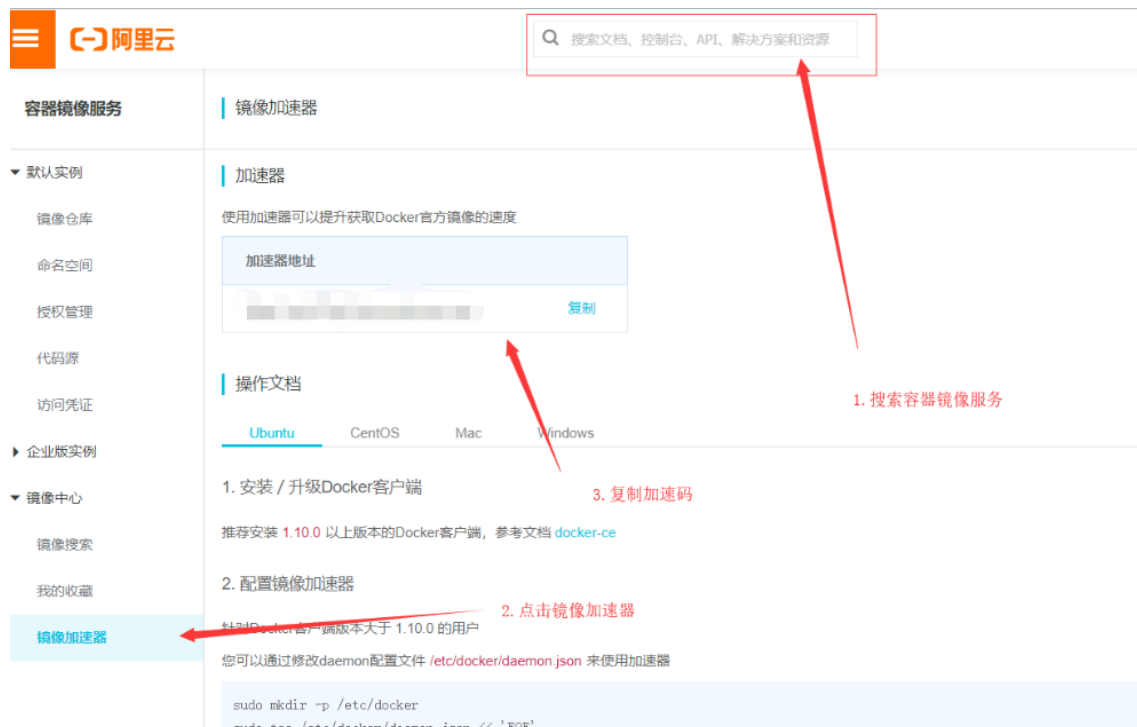
```
docker run hello-world
```

注意:

- 第七步如果下载失败, 则可以使用阿里云的加速代理, 加速代理的教程四。
- 目前Windows 10 64bit也支持了docker, 其他版本windows均不支持。

四、阿里云Docker镜像加速

登录阿里云, 进入后台



```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<- 'EOF'
{
  "registry-mirrors": ["你的码"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

- 使用tee进行输出{}这段json文本
- 辅助进程重载
- 重启docker

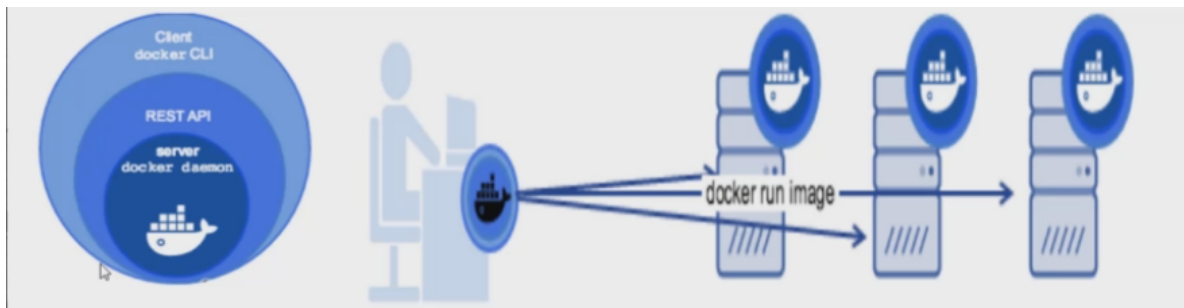
五、Docker的基本概念

- Docker是提供应用打包, 部署与运行应用的容器化平台



总体架构

- Docker引擎分为3个，内层是一个docker daemon，中间层REST API进行通讯，最外层是CLI，我们通过REST API进行与server进行通讯。
- 因为通讯是REST API，所以我们使用协议是HTTP 协议



Docker引擎

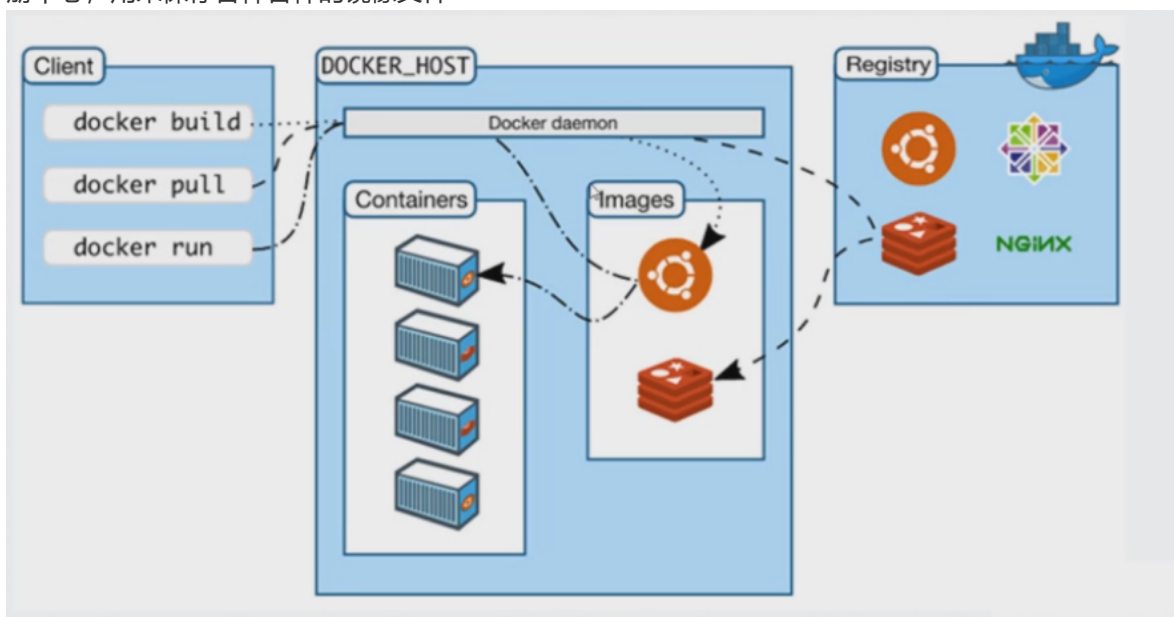
- 容器与镜像

镜像：镜像是文件，只读的，提供了运行完整软硬件应用程序的集装箱

容器：是镜像的实例，由Docker负责创建，容器之间彼此隔离

- Docker执行流程

客户端来向服务端发送命令，服务端由Docker daemon来负责管理Containers和Images，最右边是注册中心，用来保存各种各样的镜像文件



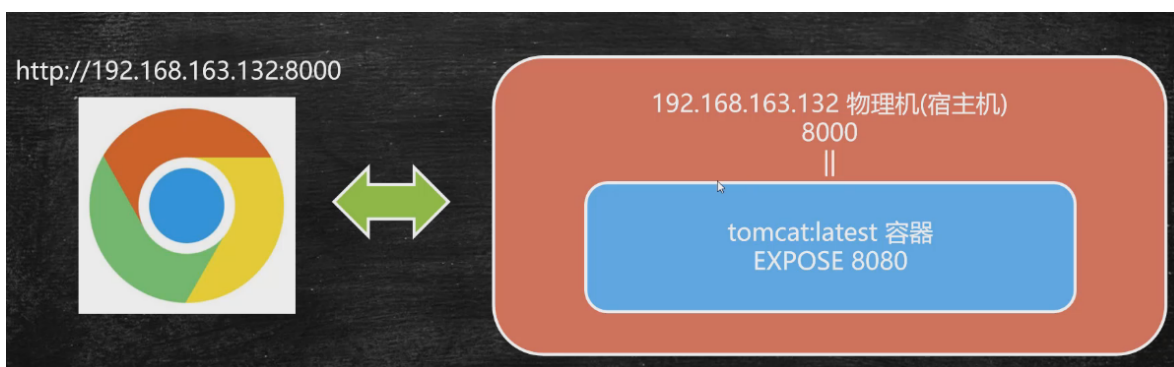
docker执行流程

六、命令

- docker pull imageName<:tags> 远程仓库抽取镜像
- docker images 查看远程抽取镜像
- docker run 创建容器，启动应用
- docker ps 查看正在运行的镜像
- docker rm 删除容器
- docker rmi 删除镜像

七、Docker宿主机与容器通信

我们可以使用docker进行端口映射



ip运行原理

容器正常启动，内部8080映射到8000端口

```
docker run -p 8000:8080 tomcat
```

容器后台运行

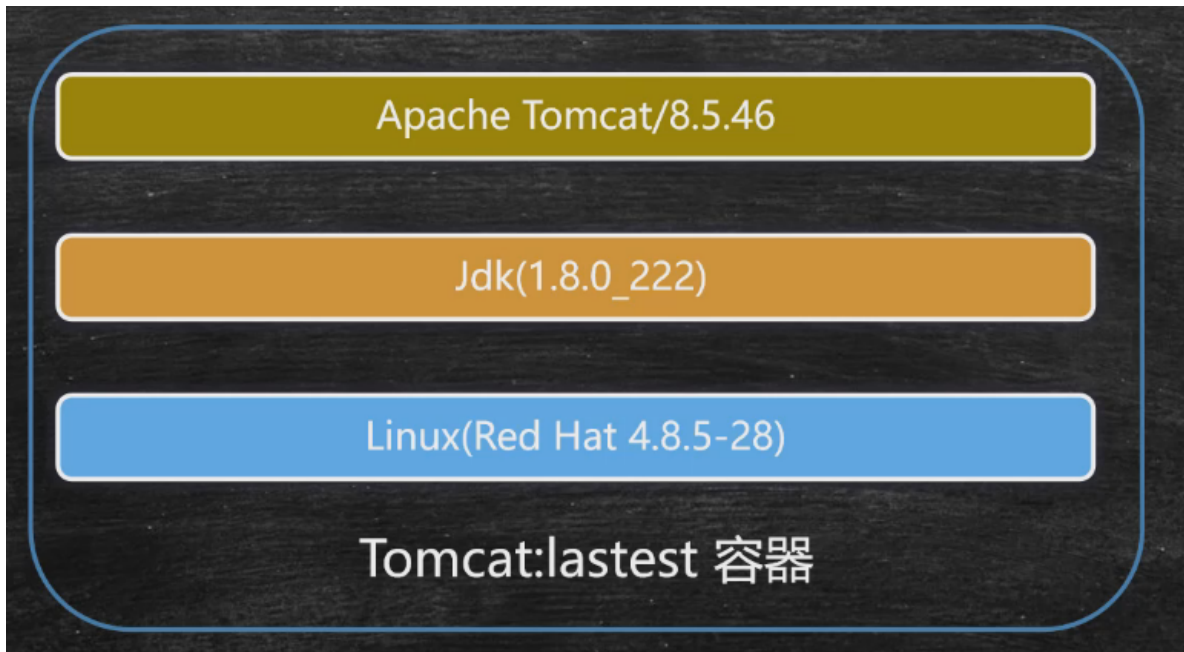
```
docker run -p 8000:8080 -d tomcat
```

杀死后台容器

```
docker kill containerID
```

八、容器内部结构

- 以tomcat为例，Tomcat容器内部组件



Tomcat组件

- 思考：为什么有个linux文件还这么小？

理由：linux仅支持这个应用，其他组件一律没有安装，所以占用资源较少

- 容器中执行命令

```
docker exec [-it] 容器id 命令
```

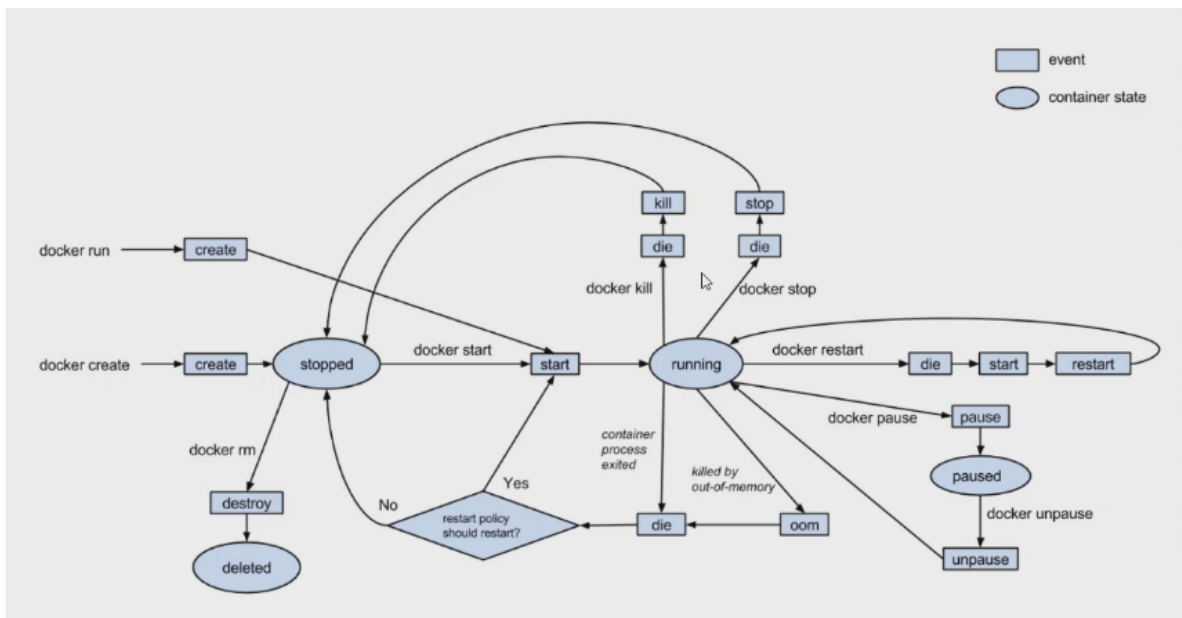
例子

```
docker exec -it 098988czcd /bin/bash
```

退出命令

```
exit
```

九、容器生命周期



容器生命周期图

- docker run = docker create + docker start
- docker create 单执行，则会进入停止状态
 - 执行docker start 进入开始状态
 - docker destroy 删除容器
- docker kill 或者 docker stop 都置die状态，紧接着进入stop状态
 - docker kill 后进行docker start 创建新进程
 - docker stop 后进行docker start 进入恢复
- docker restart 可以重启
- docker pause 进行暂停状态
- 内存溢出OOM，会置die
- 查看状态 docker ps -a 我们会发现更多子状态

十、Dockerfile构建镜像

- Dockerfile是一个包含用于组合镜像命令的文本文件
- Docker读取Dockerfile中指令，进行自动生成镜像
- docker build -t 机构/镜像名<:tags> Dockerfile目录

案例dockerfile

```

from tomcat:latest //设置基准镜像
MAINTAINER xxx.com //拥有者
WORKDIR /opt/tomcat/webapps //结合web容器路径进行放置打包，如果目录不存在，则创建
ADD docker-web ./docker-web //添加 docker-web ./docker-web
  
```

实际案例

创建dockerweb文件夹，内部放一个index.html

在外部书写Dockerfile，无扩展名

```
FROM consol/tomcat-7.0
```

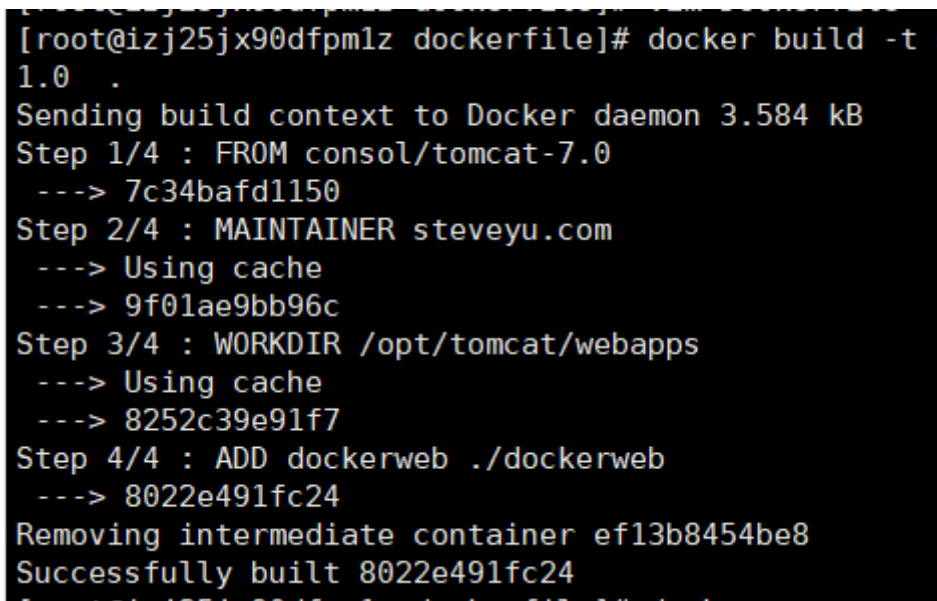
```
MAINTAINER steveyu.com
```

```
WORKDIR /opt/tomcat/webapps
```

```
ADD docker-web ./docker-web
```

执行 `docker build -t steveyu/mywebapp:1.0 /usr/local`

- 镜像分层



```
[root@izj25jx90dfpmlz dockerfile]# docker build -t steveyu/mywebapp:1.0 .
Sending build context to Docker daemon 3.584 kB
Step 1/4 : FROM consol/tomcat-7.0
--> 7c34bafd1150
Step 2/4 : MAINTAINER steveyu.com
--> Using cache
--> 9f01ae9bb96c
Step 3/4 : WORKDIR /opt/tomcat/webapps
--> Using cache
--> 8252c39e91f7
Step 4/4 : ADD dockerweb ./dockerweb
--> 8022e491fc24
Removing intermediate container ef13b8454be8
Successfully built 8022e491fc24
```

镜像构件图

我们在处理的时候，每一层都是一个镜像，处理过的镜像不必要重新处理，极大加快了构建镜像的速度