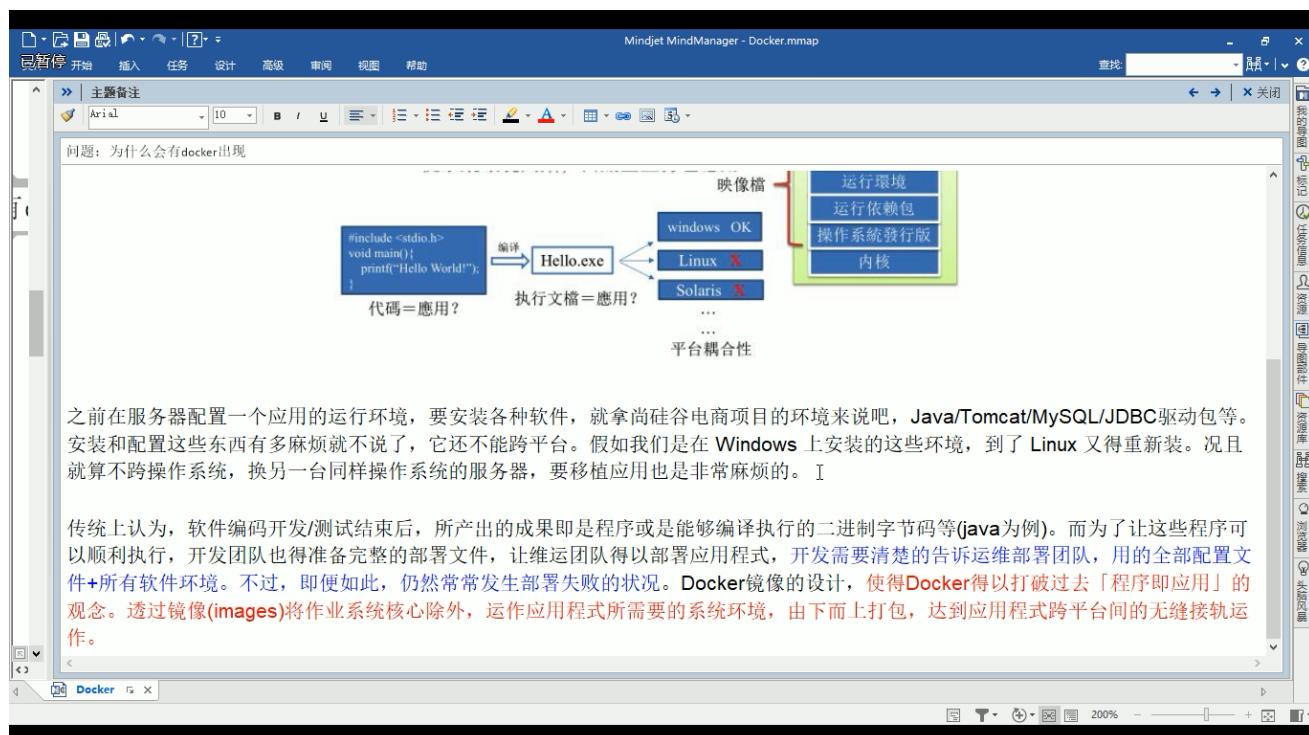
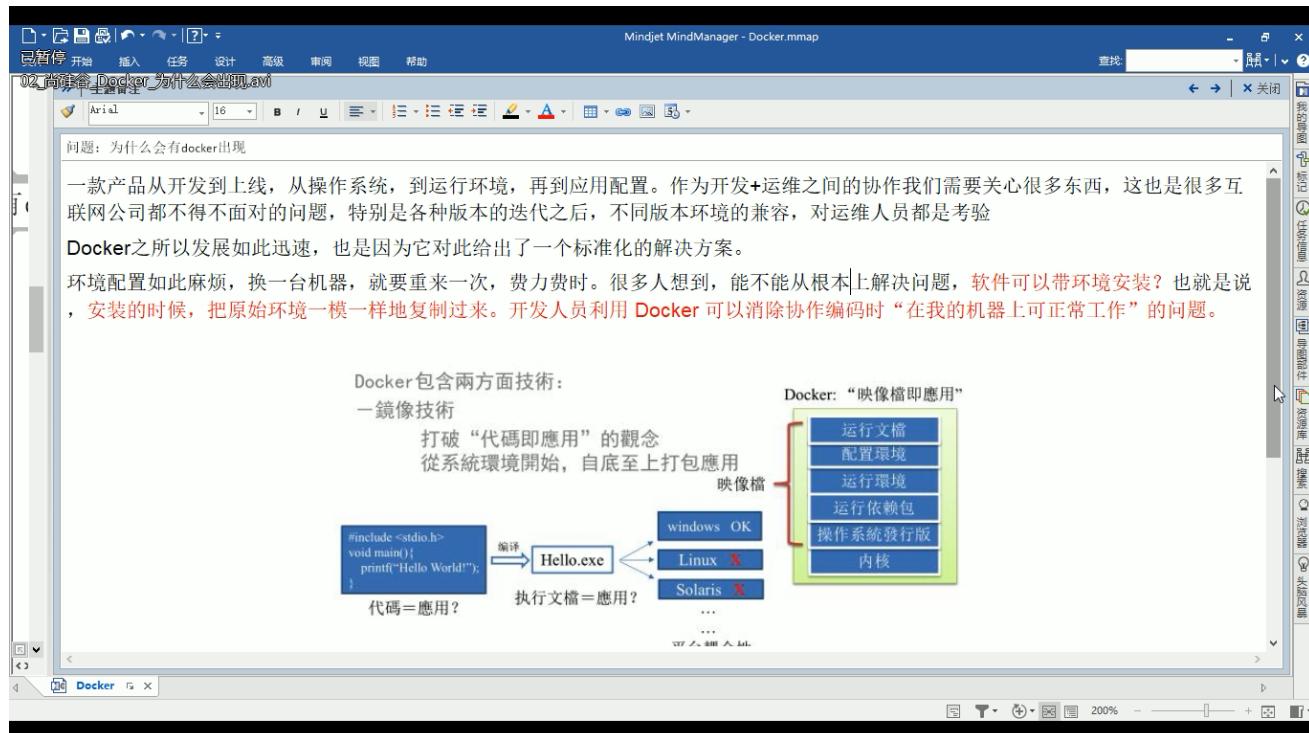


# 一、docker简介

docker是一种解决了运行环境和配置问题的软件容器，方便做持续集成并有助于整体发布的容器虚拟化技术。



**docker理念**

Docker是基于Go语言实现的云开源项目。

Docker的主要目标是“Build, Ship and Run Any App, Anywhere”，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管理，使用户的APP（可以是一个WEB应用或数据库应用等等）及其运行环境能够做到“一次封装，到处运行”。

Linux 容器技术的出现就解决了这样一个问题，而 Docker 就是在它的基础上发展过来的。将应用运行在 Docker 容器上面，而 Docker 容器在任何操作系统上都是一致的，这就实现了跨平台、跨服务器。只需要一次配置好环境，换到别的机子上就可以一键部署好，大大简化了操作。

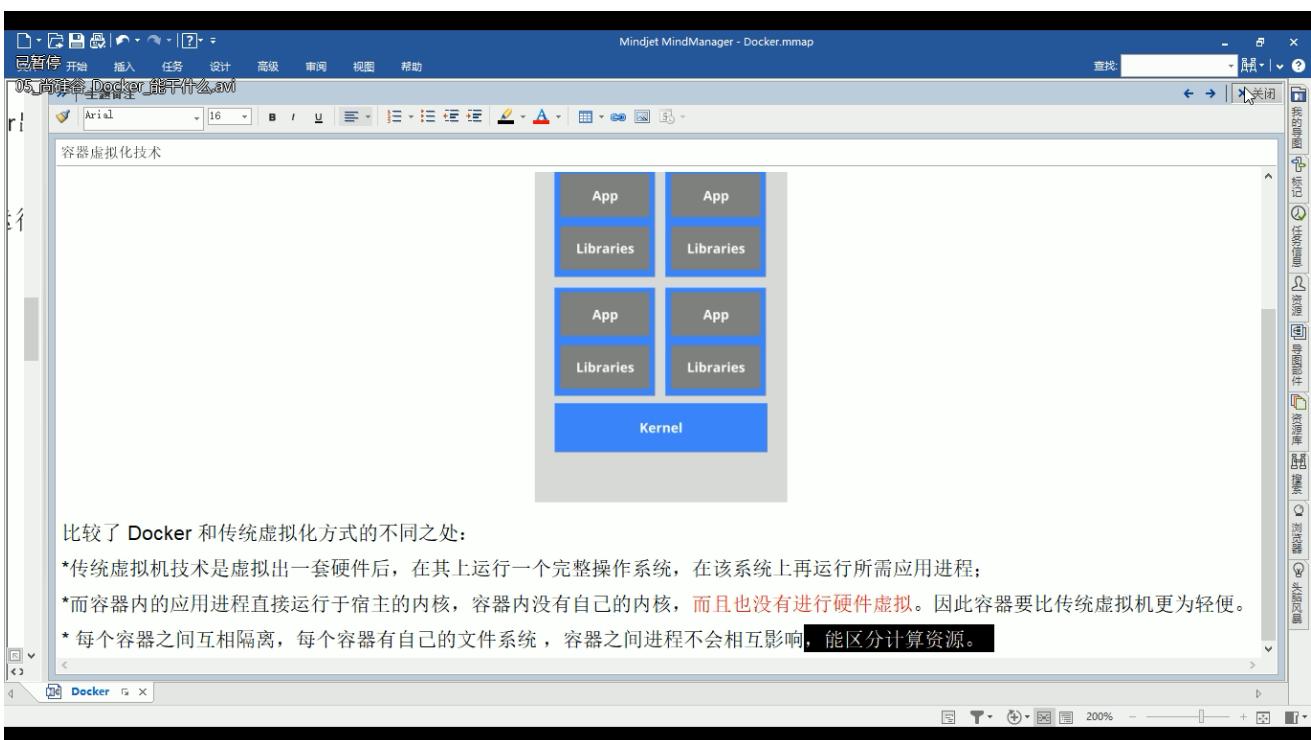
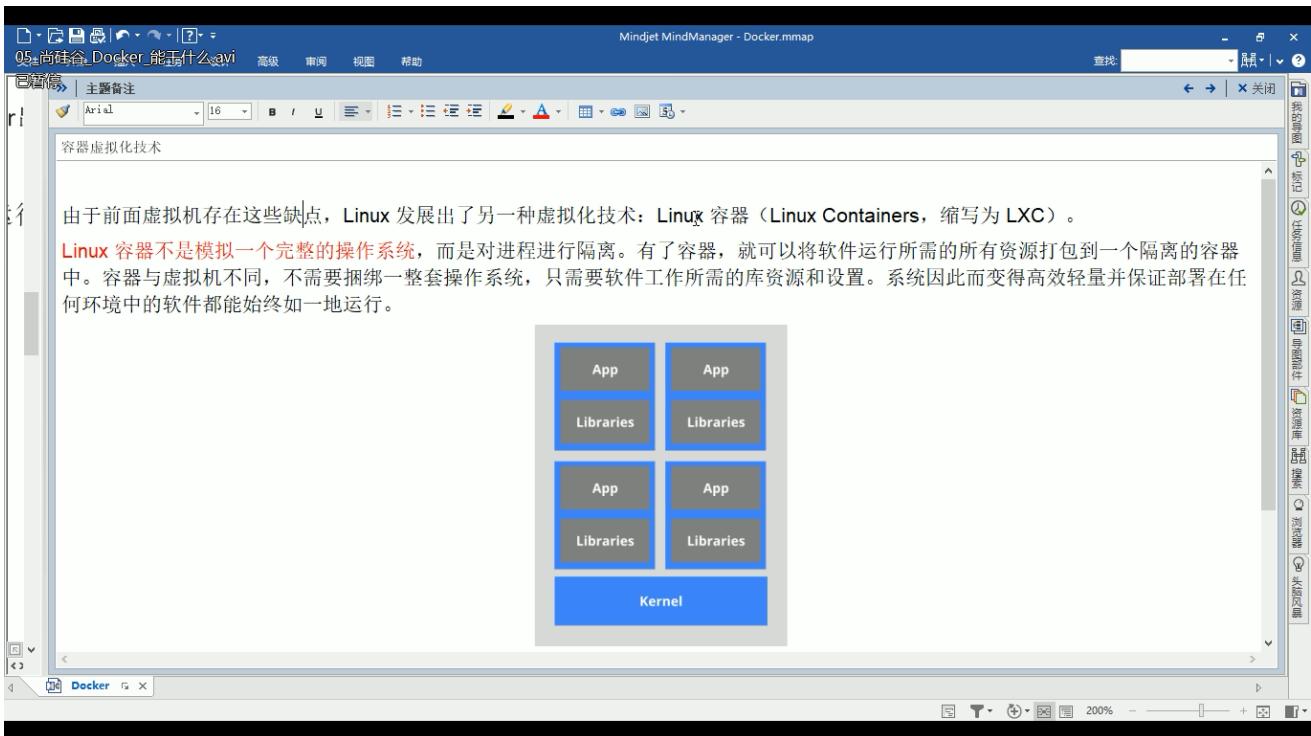
## 1. 虚拟机技术和容器虚拟化技术比较：

**之前的虚拟机技术**

它可以在一种操作系统里面运行另一种操作系统，比如在Windows 系统里面运行Linux 系统。应用程序对此毫无感知，因为虚拟机看上去跟真实系统一模一样，而对于底层系统来说，虚拟机就是一个普通文件，不需要了就删掉，对其他部分毫无影响。这类虚拟机完美的运行了另一套系统，能够使应用程序，操作系统和硬件三者之间的逻辑不变。

虚拟机的缺点：

- 1 资源占用多
- 2 元余步骤多
- 3 启动慢



## 2. 容器虚拟化技术优点：

- 更快速的应用交付和部署
- 更便捷的升级和扩容
- 更简单的运维系统
- 更高效的计算资源利用

## 二、docker安装

docker官网：[www.docker.com](http://www.docker.com)

docker中文网：<https://www.docker-cn.com/>

### 1. 环境支持

docker支持的Centos版本

Centos7(64-bit), 系统内核版本为3.10以上

Centos6.5(64-bit)或更高的版本，系统内核版本为2.6.32-431或者更高的版本

#### 1. 查看系统版本信息

两种方式

```
lsb_release -a
```

```
[root@iZm5e696jej4tmf30u41nrZ ~]# lsb_release -a
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: CentOS
Description:    CentOS Linux release 7.4.1708 (Core)
Release:        7.4.1708
Codename:       Core
```

```
cat /etc/redhat-release
```

```
[root@iZm5e696jej4tmf30u41nrZ etc]# cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
```

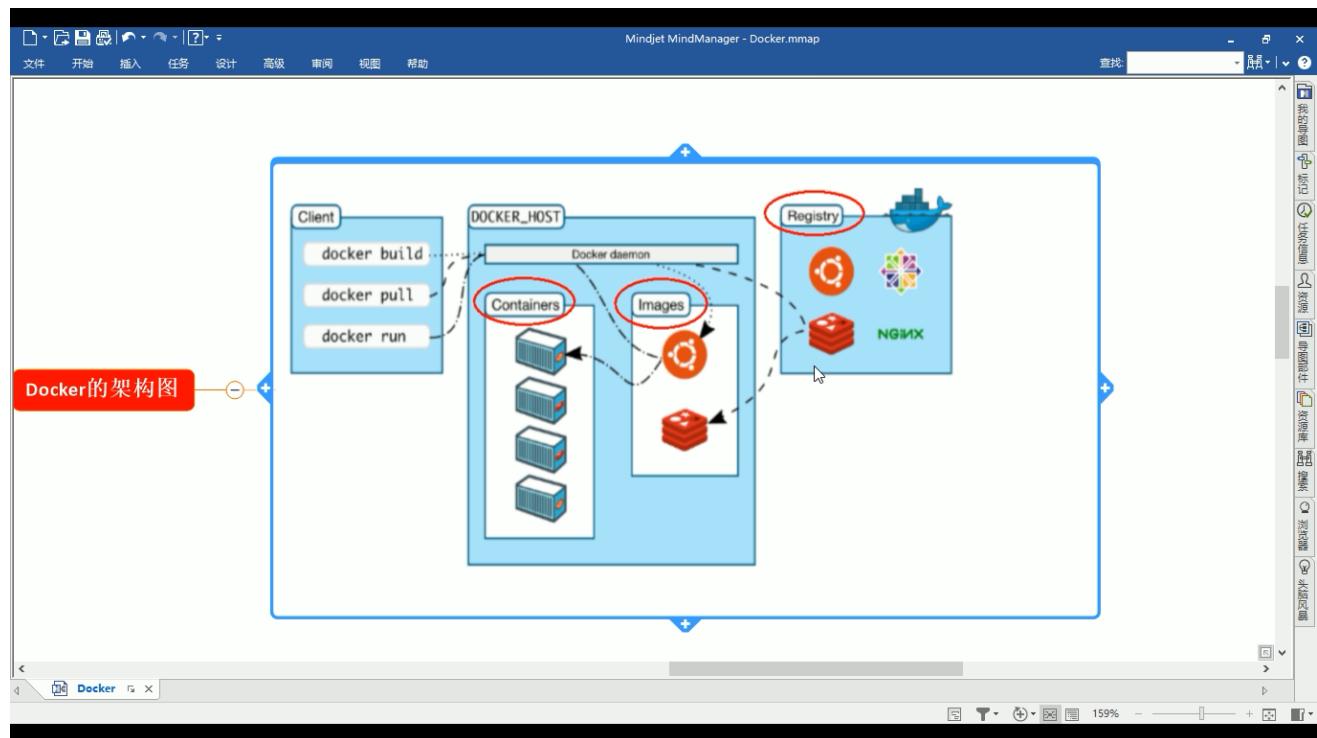
#### 2. 查看系统内核版本

```
uname -r
```

```
[root@iZm5e696jej4tmf30u41nrZ ~]# uname -r  
3.10.0-693.2.2.el7.x86_64
```

## 2. 重要概念

### docker架构图



## 1. 镜像

Docker 镜像（Image）就是一个只读的模板。镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。

容器与镜像的关系类似于面向对象编程中的对象与类。

面向对象	
Docker	面向对象
容器	对象
镜像	类

## 2. 容器

Docker 利用容器（Container）独立运行的一个或一组应用。容器是用镜像创建的运行实例。

它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。

可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

容器的定义和镜像几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

## 3.仓库

仓库（Repository）是集中存放镜像文件的场所。

仓库（Repository）和仓库注册服务器（Registry）是有区别的。仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

最大的公开仓库是 Docker Hub(<https://hub.docker.com/>)，

存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云、网易云等

## 总结：

需要正确的理解仓储/镜像/容器这几个概念：

Docker 本身是一个容器运行载体或称之为管理引擎。我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就似乎 image 镜像文件。只有通过这个镜像文件才能生成 Docker 容器。image 文件可以看作是容器的模板。Docker 根据 image 文件生成容器的实例。同一个 image 文件，可以生成多个同时运行的容器实例。

- \* image 文件生成的容器实例，本身也是一个文件，称为镜像文件。
- \* 一个容器运行一种服务，当我们需要的时候，就可以通过 docker 客户端创建一个对应的运行实例，也就是我们的容器
- \* 至于仓储，就是放了一堆镜像的地方，我们可以把镜像发布到仓储中，需要的时候从仓储中拉下来就可以了。| I

## 3. docker的安装

### 1. centos6.8安装docker

1. yum install -y epel-release 

2. yum install -y docker-io 

3. 安装后的配置文件: /etc/sysconfig/docker 

4. 启动Docker后台服务: service docker start

5. docker version验证 

```
yum install -y epel-release
```

```
yum install -y docker-io
```

```
service docker start #重启docker服务
```

```
docker version #检查docker版本
```

注: docker使用EPEL发布, RHEL系的OS首先要保证已经有EPEL仓库, 否则先检查OS的版本, 然后安装相应的EPEL包

## 2. centos6.8配置阿里云镜像加速

加速地址获取

[https://cr.console.aliyun.com/?  
spm=5176.1971733.0.2.6c045aaa6jqmCd#/accelerator](https://cr.console.aliyun.com/?spm=5176.1971733.0.2.6c045aaa6jqmCd#/accelerator)

自己的阿里云镜像加速

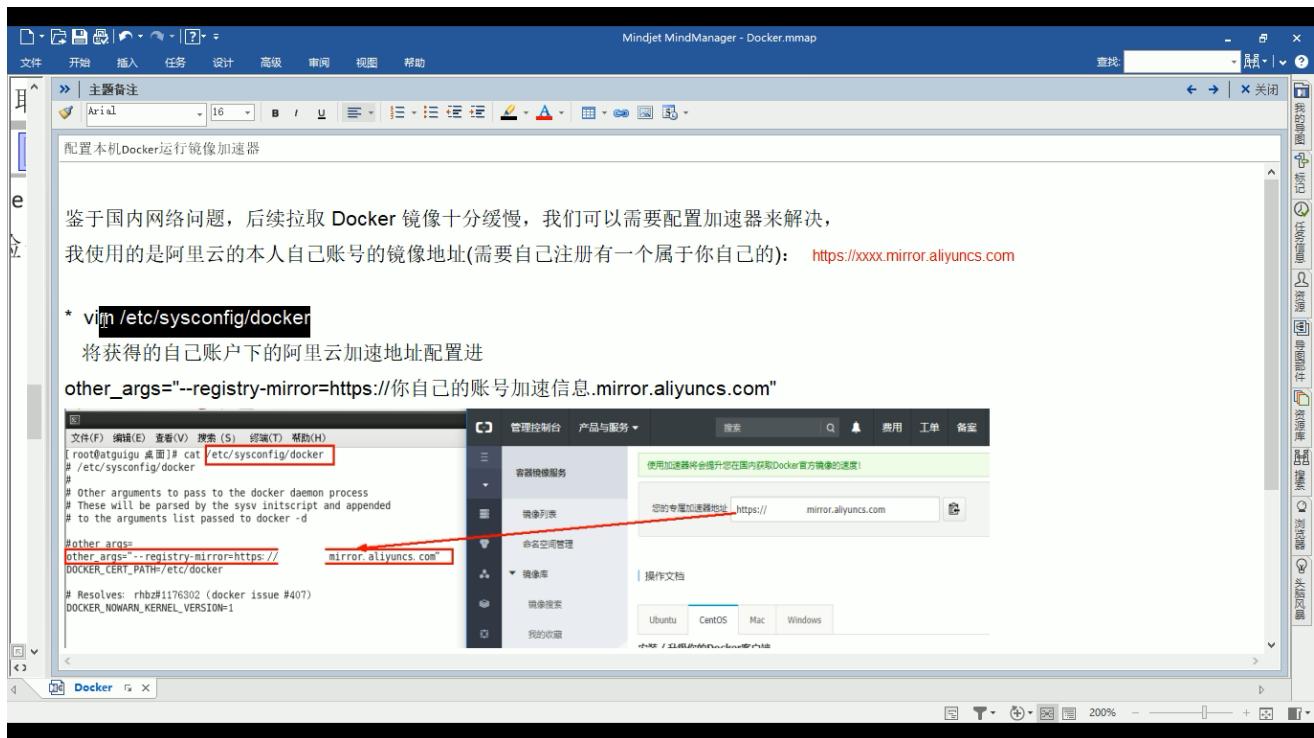
专属加速地址: <https://fvrybxjp.mirror.aliyuncs.com>

```

sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<- 'EOF'
{
  "registry-mirrors":
  ["https://fvrybxjp.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker

```

## 修改配置文件 /etc/sysconfig/docker



## 重启docker服务

```
service docker restart
```

## 3. centos 7安装

官网：<https://docs.docker.com/install/linux/docker-ce/centos/#install-docker-ce>

中文版：<https://docs.docker-cn.com/engine/installation/linux/docker-ce/centos/#%E5%8D%B8%E8%BD%BD-docker-ce> (不一定是最新的)

### 3.1 安装gcc相关

```
yum -y install gcc  
yum -y install gcc-c++  
gcc -v      #查看gcc版本
```

### 3.2 卸载旧版本

```
#如果是系统root权限，则不用写sudo  
sudo yum remove docker \  
    docker-client \  
    docker-client-latest \  
    docker-common \  
    docker-latest \  
    docker-latest-logrotate \  
    docker-logrotate \  
    docker-selinux \  
    docker-engine-selinux \  
    docker-engine
```

3.3 安装所需的软件包。`yum-utils` 提供了 `yum-config-manager` 实用程序，并且 `devicemapper` 存储驱动需要 `device-mapper-persistent-data` 和 `lvm2`。

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

### 3.4 使用下列命令设置 **stable** 镜像仓库。您始终需要使用 **stable** 镜像仓库

```
#国外地址    \相当于换行的连接符
sudo yum-config-manager \
--add-repo \
https://download.docker.com/linux/centos/docker-
ce.repo
```

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

### 3.5 更新 `yum` 软件包索引。

```
sudo yum makecache fast
```

### 3.6 安装DOCKER CE

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

### 3.7 启动docker

```
sudo systemctl start docker
```

### 3.8 测试

```
sudo docker run hello-world #运行hello world  
docker version #查看docker版本
```

### 3.9 配置镜像加速

```
mkdir -p /etc/docker  
vim /etc/docker/daemon.json
```

在daemon.json中添加

```
{  
  "registry-mirrors":  
  ["https://fvrybxjp.mirror.aliyuncs.com"]  
}
```

然后执行以下命令

```
#重新加载配置文件  
systemctl daemon-reload  
#重启docker  
systemctl restart docker
```

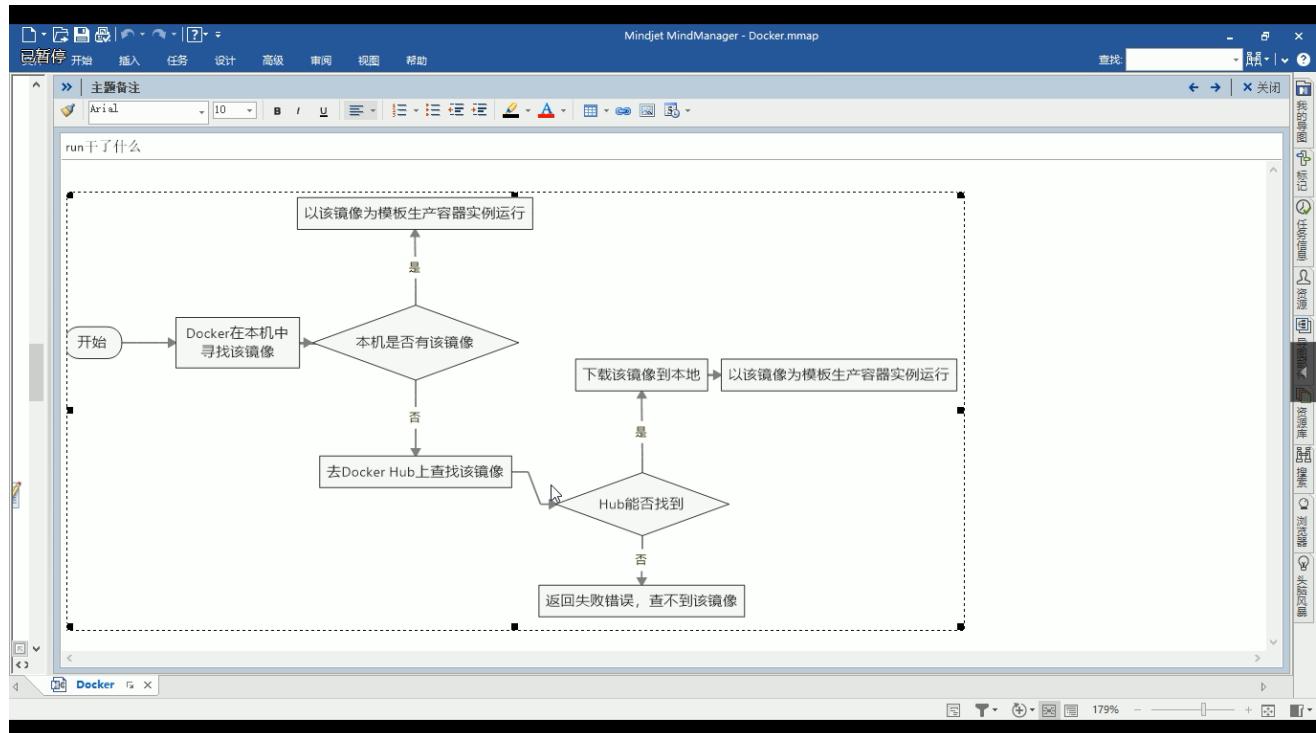
### 3.10 卸载docker

```
#停止docker  
sudo systemctl stop docker  
#卸载 Docker 软件包  
sudo yum remove docker-ce  
#主机上的镜像、容器、存储卷、或定制配置文件不会自动删除。如  
需删除所有镜像、容器和存储卷，请运行下列命令  
  
sudo rm -rf /var/lib/docker
```

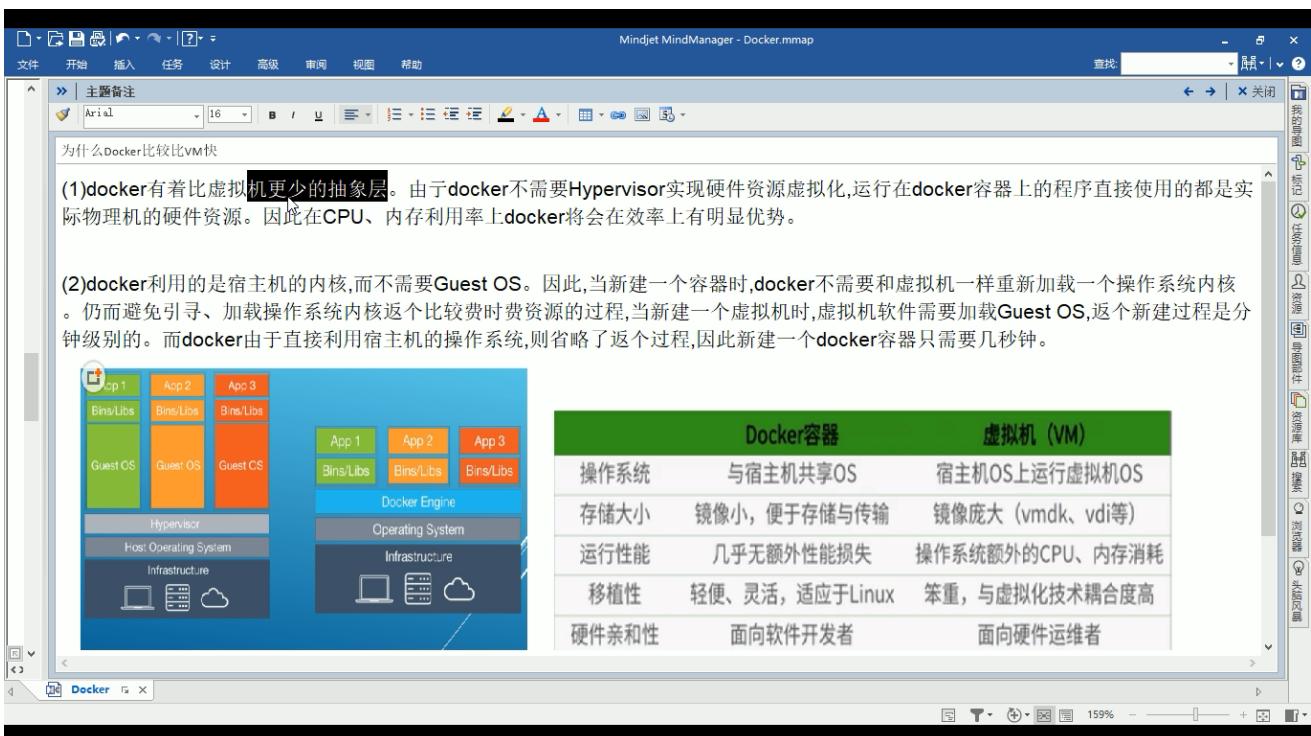
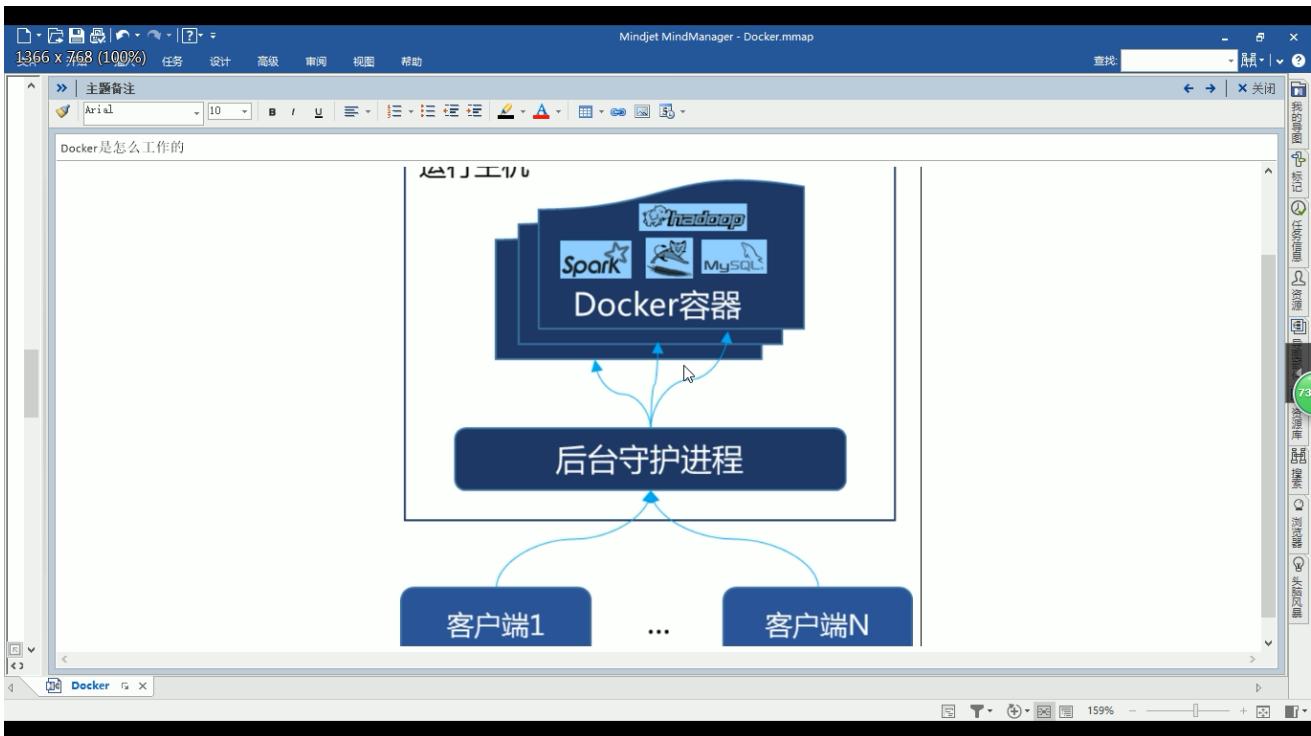
## 4. docker run

```
docker run hello-world
```

docker run 命令工作过程



Docker是一个Client-Server结构的系统，Docker守护进程运行在主机上，然后通过Socket连接从客户端访问，守护进程从客户端接受命令并管理运行在主机上的容器。容器，是一个运行时环境，就是我们前面说到的集装箱。



Docker容器		虚拟机 (VM)
操作系统	与宿主机共享OS	宿主机OS上运行虚拟机OS
存储大小	镜像小，便于存储与传输	镜像庞大 (vmdk、vdi等)
运行性能	几乎无额外性能损失	操作系统额外的CPU、内存消耗
移植性	轻便、灵活，适应于Linux	笨重，与虚拟化技术耦合度高
硬件亲和性	面向软件开发者	面向硬件运维者
部署速度	快速，秒级	较慢，10s以上

## 三、 docker常用命令

### 1. 帮助命令

```
docker version
docker info
docker --help
```

### 2. 镜像命令

#### 1. docker images

```
#显示本地镜像
docker images
```

参数：

参数	释义	示例
-a	列出本地所有镜像层	docker images -a
-q	只显示镜像ID	docker images -q
--digests	显示镜像的摘要信息	
--no-trunc	显示完整的镜像信息	

## 组合命令

docker images -qa 返回本地所有镜像的ID

```

root@atguigu:~# docker images
REPOSITORY      TAG          IMAGE ID      CREATED        VIRTUAL SIZE
tomcat          latest       cb315192e016 7 days ago   465.3 MB
hello-world     latest       83f0de727d85  5 weeks ago  1.848 kB

```

各个选项说明:

- REPOSITORY: 表示镜像的仓库源
- TAG: 镜像的标签
- IMAGE ID: 镜像ID
- CREATED: 镜像创建时间
- SIZE: 镜像大小

同一仓库源可以有多个 TAG, 代表这个仓库源的不同个版本, 我们使用 REPOSITORY:TAG 来定义不同的镜像。  
如果你不指定一个镜像的版本标签, 例如你只使用 ubuntu, docker 将默认使用 ubuntu:latest 镜像

## 2. docker search

#从docker hub查询指定镜像信息  
docker search [OPTIONS]镜像名称

参数	释义	示例
--no-trunc	显示完成的镜像描述	
-s	列出收藏不小于指定值的镜像	
-automated	只列出automated build类型的镜像	

```
docker search -s 30 tomcat
```

### 3. docker pull

#从仓库中拉取镜像（默认拉取docker hub上的，配了阿里云加速之后从阿里云仓库下载）

```
docker pull 镜像名称[:TAG]
```

#示例

```
docker pull tomcat
```

#默认下载最新的，等价于 docker pull tomcat:latest

### 4. docker rmi

```
docker rmi -f 镜像ID #删除单个镜像
```

```
docker rmi -f 镜像名1:TAG 镜像名2:TAG #删除指定多个镜像
```

```
docker rmi -f $(docker images -qa) #删除全部镜像
```

-f 表示强制删除

示例：

```
docker rmi -f hello-world  
docker rmi -f hello-world nginx
```

### 3. 容器命令

#### 1.docker run

##### 新建并启动一个容器

参数：

参数	释义
--name	为容器起一个别名
-d	后台运行容器，并且返回容器ID，即启动守护式容器
-i	以交互模式运行容器，通常与-t同时使用
-t	为容器重新分配一个输入伪终端，常与-i一起使用
-P	随机端口映射
-p	指定端口映射，有以下四种格式

ip:hostPort:containerPort

ip::containerPort

hostPort:containerPort

containerPort

例子：

```
docker run -it 49f7960eb7e4 #启动交互式容器  
docker run -d 容器ID或容器名 #启动守护式容器
```

```
docker run -d 容器名  
  
#使用镜像centos:latest以后台模式启动一个容器  
docker run -d centos  
  
问题：然后docker ps -a 进行查看，会发现容器已经退出  
很重要的要说明的一点：Docker容器后台运行，必须有一个前台进程。  
容器运行的命令如果不是那些一直挂起的命令（比如运行top, tail），就是会自动退出的。  
  
这个是docker的机制问题，比如你的web容器，我们以nginx为例，正常情况下，我们配置启动服务只需要启动响应的service即可。例如  
service nginx start  
但是，这样做，nginx为后台进程模式运行，就导致docker前台没有运行的应用，  
这样的容器后台启动后，会立即自杀因为他觉得他没事可做了。  
所以，最佳的解决方案是，将你要运行的程序以前台进程的形式运行
```

## 2. docker ps

列出当前正在运行中的容器

- -a :列出当前正在运行的容器和历史上运行过的所有容器
- -l: 显示最近创建的容器
- -n: 显示最近n个创建的容器
- -q: 静默模式，只显示容器编号
- --no-trunc: 不截断输出

```
docker ps -ql  
docker ps -n 2
```

## 3. 退出容器

两种方式

- exit : 容器退出并且停止

- **ctrl+P+Q**: 容器不停止退出

## 4. 启动容器

docker start + 容器ID或者名称

## 5. 停止容器

docker stop 容器ID或容器名称

## 6. 强制停止容器

docker kill 容器ID或者容器名称

## 7. 删除容器

docker rm -f 容器ID或者容器名称

删除多个容器

```
docker rm -f $(docker ps -a -q)
docker ps -a -q | xargs docker rm
```

## 8. 查看容器日志

docker logs -f -t --tail 容器ID

-f : 跟随最新的日志打印

-t: 加入时间戳

--tail 数字: 显示最后多少条

```
docker logs -f -t --tail 5 f886265e1980
```

## 9. 查看容器内运行的进程

docker top 容器ID

## 10. 查看容器内部细节

docker inspect 容器ID

## 11. 进入正在运行中的容器以命令行进行交互

#在容器中打开新的终端，并且可以启动新的进程

docker exec -it 容器ID baseShell

#重新进入容器启动命令的终端，不会启动新的进程

docker attach 容器ID

示例：

```
docker run -it centos /bin/bash    #以交互式启动centos  
ctrl+P+Q                      #不停止退出
```

#不进入centos，在宿主机上执行命令操作centos

docker exec -t f48a49415cd8 ls -l

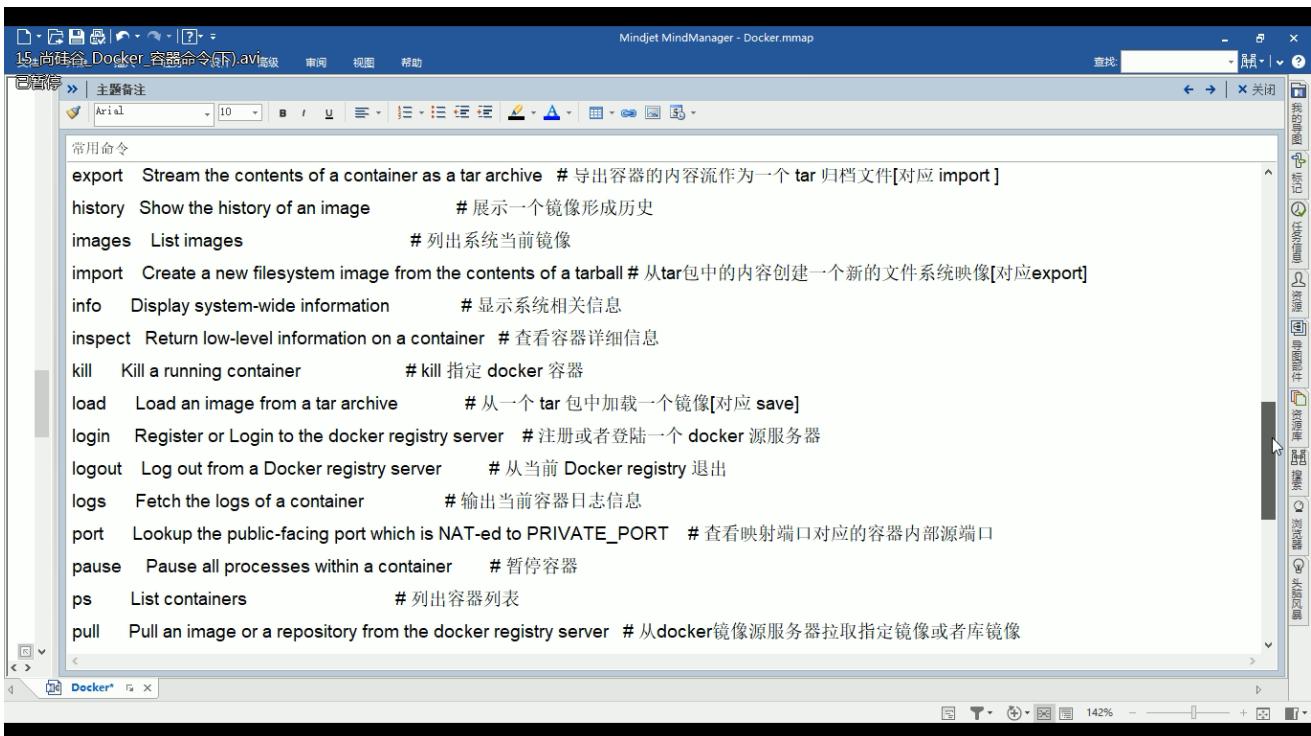
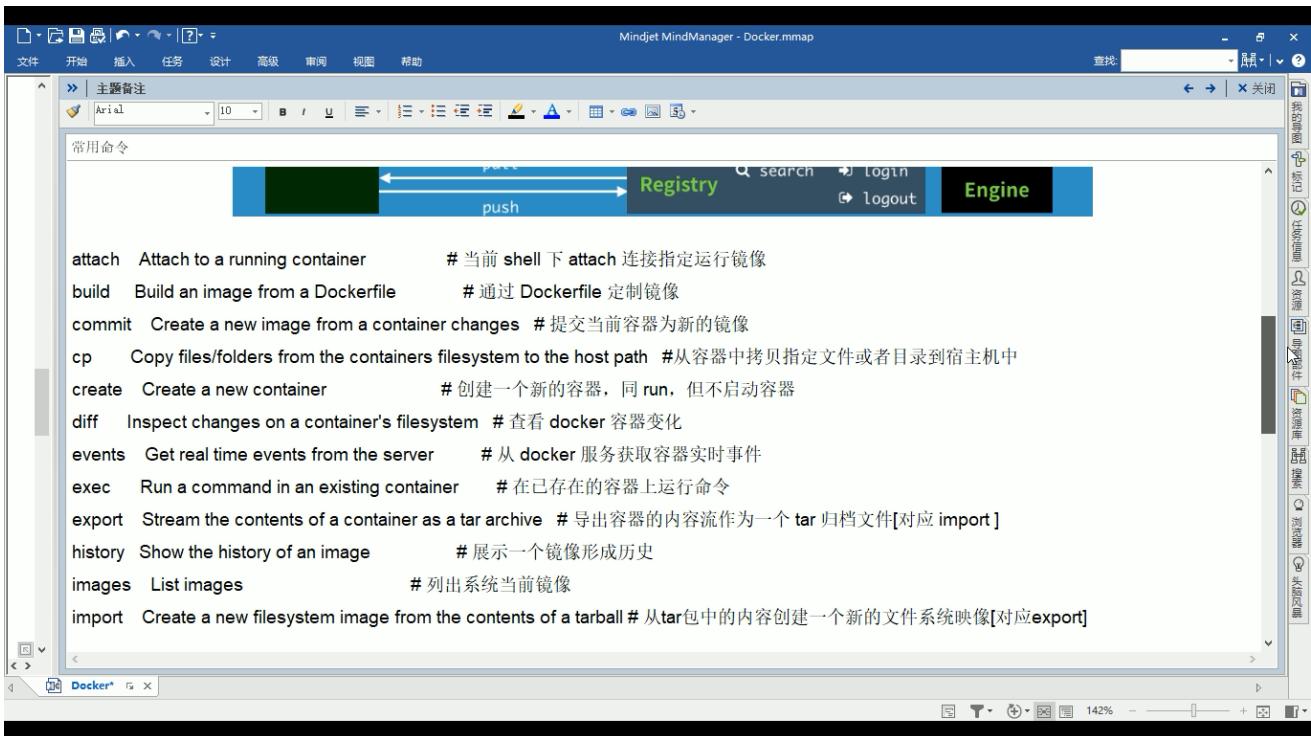
docker attach f48a49415cd8 #重新进入启动中的centos

## 12. 从容器内拷贝文件到主机上

docker cp 容器ID: 容器内路径 目的路径

```
docker cp f48a49415cd8:/tmp/yum.log /root
```

## 13 常用命令总结



Mindjet MindManager - Docker.mmap

```

文件 开始 插入 任务 设计 高级 审阅 视图 帮助
主题备注
Arial 14
常用命令

push Push an image or a repository to the docker registry server # 推送指定镜像或者库镜像至docker源服务器
restart Restart a running container # 重启运行的容器
rm Remove one or more containers # 移除一个或者多个容器
rmi Remove one or more images # 移除一个或多个镜像[无容器使用该镜像才可删除，否则需删除相关容器才可继续或-f 强制删除]
run Run a command in a new container # 创建一个新的容器并运行一个命令
save Save an image to a tar archive # 保存一个镜像为一个 tar 包[对应 load]
search Search for an image on the Docker Hub # 在 docker hub 中搜索镜像
start Start a stopped containers # 启动容器
stop Stop a running containers # 停止容器
tag Tag an image into a repository # 给源中镜像打标签
top Lookup the running processes of a container # 查看容器中运行的进程信息
unpause Unpause a paused container # 取消暂停容器
version Show the docker version information # 查看 docker 版本号
wait Block until a container stops, then print its exit code # 截取容器停止时的退出状态值

```

## 四、docker镜像

### 1. 什么是镜像

镜像是一种轻量级的，可执行的独立软件包。用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码，运行时、库、环境变量和配置文件。

### 2. 联合文件系统

**UnionFS（联合文件系统）：** Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。



特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

### 3. docker镜像加载原理

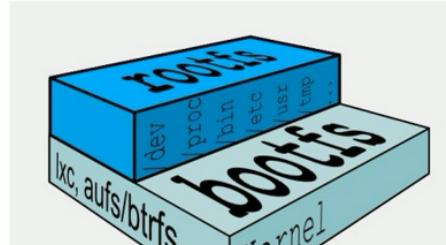
Docker镜像加载原理

#### Docker镜像加载原理：

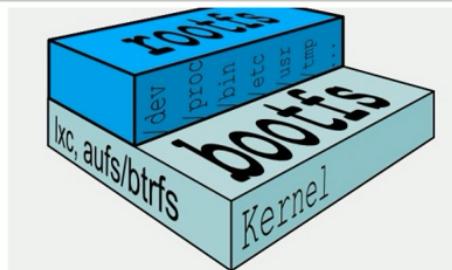
docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader和kernel, bootloader主要是引导加载kernel, Linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu, Centos等等。



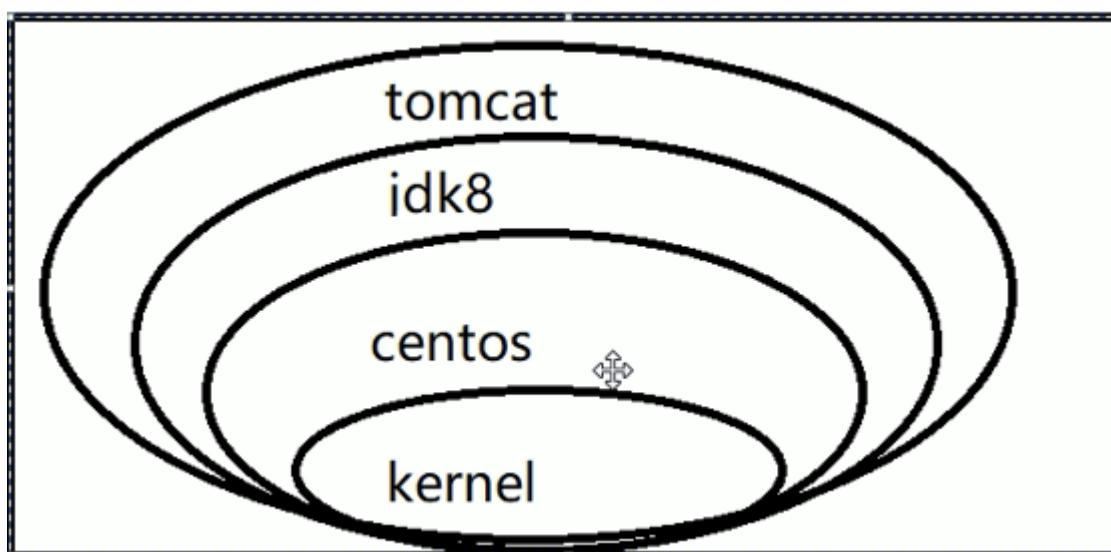
Docker镜像加载原理



平时我们安装进虚拟机的CentOS都是好几个G，为什么docker这里才200M？？

```
[root@atguigu docker]# docker images centos
REPOSITORY      TAG          IMAGE ID      CREATED        VIRTUAL SIZE
centos          latest       88ec626ba223   2 weeks ago   199.7 MB
[root@atguigu docker]#
```

对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了，因为底层直接用Host的kernel，自己只需要提供rootfs就行了。由此可见对于不同的linux发行版，bootfs基本是一致的，rootfs会有差别，因此不同的发行版可以公用bootfs。



### 4. docker镜像采用分层结构好处

可以实现资源共享，比如多个镜像从相同的base镜像而来，则宿主机上只需要保存一份base镜像即可。同时内存中也只需要加载一份base镜像，就可以为所有的镜像服务了。而且镜像的每一层都可以被共享。

docker镜像的特点：

docker镜像都是只读的，当容器启动的时候，一个新的可写层被加载到镜像的顶部。这一层通常称作“容器层”，“容器层”之下的都叫“镜像层”。

## 5. docker镜像commit操作

docker commit 可以提交容器副本使之成为一个新的镜像

命令：

```
docker commit -m="" -a="" 容器ID 要创建的目标镜像名称：[标签]
```

示例：将tomcat中的doc文档删除之后以他为模板commit一个新镜像

```
#以交互式运行tomcat
docker run -it -p 8888:8080 tomcat
#退出不停止该tomcat
ctrl+P+Q
#重新进入该tomcat
docker exec -it 9ee76ad893c8 /bin/bash
cd webapps          #进入webapps
ls -l               #列出当前目录
```

```
rm -rf docs      #删除docs
ctrl+P+Q        #退出不停止该tomcat
#以这个删除docs的tomcat (id为9ee76ad893c8) 为模板创建镜像，lidonghao 为命名空间，1.0为版本号
docker commit -a='ldh' -m='del docs tomcat'
9ee76ad893c8 lidonghao/tomcat01:1.0
#运行我们生成的镜像，该tomcat没有docs
docker run -d -p 8887:8080 lidonghao/tomcat01:1.0
#运行原来的tomcat，该tomcat有docs
docker run -d -p 8886:8080 tomcat
```

## 五、docker容器数据卷

### 1.docker容器数据卷介绍

有点类似我们redis中的rdb和aof文件

先来看看Docker的理念：

- \* 将应用与运行的环境打包形成容器运行，运行可以伴随着容器，但是我们对数据的要求希望是持久化的
- \* 容器之间希望有可能共享数据

Docker容器产生的数据，如果不通过docker commit生成新的镜像，使得数据做为镜像的一部分保存下来，那么当容器删除后，数据自然也就没有了。

为了能保存数据在docker中我们使用卷。

### 2.作用

可以实现容器持久化

可以使容器间继承和共享数据

卷就是目录或文件，存在于一个或多个容器中，由docker挂载到容器，但不属于联合文件系统，因此能够绕过Union File System提供一些用于持续存储或共享数据的特性：

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷

特点：

- 1: 数据卷可在容器之间共享或重用数据
- 2: 卷中的更改可以直接生效
- 3: 数据卷中的更改不会包含在镜像的更新中
- 4: 数据卷的生命周期一直持续到没有容器使用它为止

## 3. 容器内添加数据卷

### 1. 命令行添加

#### 命令行格式

```
docker run -it -v /宿主机绝对路径目录: /容器内目录 镜像名
```

#以交互式启动centos，并且添加数据卷

```
docker run -it -v /hostVolume:/containerVolume centos
```

#### 查看数据卷是否挂载成功

```
docker inspect 6cd97d1e3261
```

```
AppArmorProfile": "",
"ExecIDs": null,
"HostConfig": {
    "Binds": [
        "/hostVolume:/containerVolume"
    ],
    "ContainerIDFile": "",
    "LogConfig": {}
```

最终效果：宿主机上的hostVolume和容器中的containerVolume中的文件完全同步。

我们可以在hostVolume和containerVolume中分别创建不同的文件，最终发现两个文件夹中的文件完全同步。并且即使容器停止，我们修改宿主机hostVolume文件夹中的文件，等容器再次启动的时候hostVolume中的改动依然可以同步到containerVolume中。

## 测试流程：

- 1.退出启动中的centos :exit
- 2.修改hostVolume文件夹中的文件
- 3.执行 `docker ps -a` `docker start 容器ID` `docker attach 容器ID`
- 4.查看containerVolume文件夹下的文件，发现和hostVolume中的同步

## 加带权限的数据卷

```
docker run -it -v /hostVolume:/containerVolume: ro  
centos
```

*ro: read only*

这个时候我们容器中的文件夹containerVolume中的文件只能依据hostVolume中的文件进行同步，而我们不可以手动修改containerVolume中的文件或者在containerVolume中创建新的文件。

## 查看带权限的数据卷是否挂载成功

```
ProcessLabel": "",  
"AppArmorProfile": "",  
"ExecIDs": null,  
"HostConfig": {  
    "Binds": [  
        "/hostVolume:/containerVolume:ro"  
    ],  
    "ContainerIDFile": "",  
    "LogConfig": {  
        "Type": "journald",  
        "Config": {}  
    }  
}
```

## 常见错误

如果docker挂载主机目录Docker访问出现cannot open directory:  
Permission denied

解决办法：在挂载目录后加一个 --privileged=true参数即可

```
docker run -it -v /hostVolume:/containerVolume --  
privileged=true centos
```

## 2.DockerFile添加

### 操作步骤

1.在根目录下新建mydocker文件夹并进入

2.在mydocker中新建dockerFile文件，并且编辑

```
#volume test  
FROM centos  
VOLUME  
[ "/dataContainerVolume1", "/dataContainerVolume2" ]  
CMD echo "finished,.....successful"  
CMD /bin/bash
```

说明：

处于可移植和分享的考虑。用-v 主机目录：容器目录这种方式不可以直接在dockerFile中实现。

由于宿主目录是依赖于特定宿主机的，并不能够保证在所有的宿主机上都有这样特定的目录。

### 3.build创建新的镜像

```
#在当前目录下使用dockerFile文件创建lidonghao/centos01镜像
docker build -f /mydocker/dockerFile -t
lidonghao/centos01 .
```

```
[root@iZm5e696jej4tmf30u41nrZ mydocker]# docker build -f /mydocker/dockerFile -t lidonghao/centos01 .
Sending build context to Docker daemon 2.048 kB
Step 1/4 : FROM centos
--> 49f7960eb7e4
Step 2/4 : VOLUME /dataContainerVolume1 /dataContainerVolume2
--> Running in 03ab7f7dafca
--> 95fd6ea50e0d
Removing intermediate container 03ab7f7dafca
Step 3/4 : CMD echo "finished,.....successful"
--> Running in 381befd65469
--> 656465e8fdf2
Removing intermediate container 381befd65469
Step 4/4 : CMD /bin/bash
--> Running in de07f2934c23
--> 90475ae06084
Removing intermediate container de07f2934c23
Successfully built 90475ae06084
```

## 4.数据卷容器

### 1.数据卷容器是什么

数据卷容器是命名的容器挂载数据卷，其他容器通过挂载这个（父容器）实现数据共享，挂载数据卷的容器，称之为数据卷容器。

### 2.作用

可以实现容器间的传递共享

### 3.测试

## 启动一个父容器

```
docker run -it --name dc01 lidonghao/centos01
```

## 分别启动dc02,dc03,dc04继承自前一容器

```
docker run -it --name dc02 --volumes-from dc01  
lidonghao/centos01
```

```
docker run -it --name dc03 --volumes-from dc02  
lidonghao/centos01
```

```
docker run -it --name dc04 --volumes-from dc03  
lidonghao/centos01
```

他们都有数据卷dataContainerVolume1, dataContainerVolume2

我们在dc01中的dataContainerVolume1创建aa.txt。发现dc02,dc03,dc04中都有aa.txt。执行`docker rm -f dc02`修改dc03中的aa.txt，发现dc01,dc04中的aa.txt也得到了同步。

**结论：**容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止。

## 六、Dockerfile解析

### 1.概述

Dockerfile是用来构建Docker镜像的构建文件，是一系列命令和参数构成的脚本。

**构建步骤：**编写Dockerfile文件----docker build----docker run

## centos的Dockerfile文件:

```
FROM scratch
MAINTAINER The CentOS Project <cloud-ops@centos.org>
ADD c68-docker.tar.xz /
LABEL name="CentOS Base Image" \
      vendor="CentOS" \
      license="GPLv2" \
      build-date="2016-06-02"

# Default command
CMD ["/bin/bash"]
```

## 2.基本知识

- 1.每个保留字指令都必须为大写字母并且后面要跟随至少一个参数
- 2.指令按照从上到下，顺序执行
- 3.#表示注释
- 4.每条指令都会创建一个新的镜像层，并对镜像进行提交。

## 3.docker执行Dockerfile的大致流程

- 1.docker从基础镜像运行一个容器
- 2.执行一条指令并且对容器进行修改
- 3.执行类似docker commit的操作提交一个新的镜像层
- 4.docker再基于刚才提交的镜像运行一个新的容器
- 5.重复上述过程直到所有指令都执行完成

# 总结

从应用软件的角度来看，Dockerfile、Docker镜像与Docker容器分别代表软件的三个不同阶段，

- \* Dockerfile是软件的原材料
- \* Docker镜像是软件的交付品
- \* Docker容器则可以认为是软件的运行态。

Dockerfile面向开发，Docker镜像成为交付标准，Docker容器则涉及部署与运维，三者缺一不可，合力充当Docker体系的基石。



## 4. Dockerfile体系结构 (保留字)

**FROM**: +基础镜像，表示新镜像是基于哪个镜像来的

**MAINTAINER**: 镜像维护者的姓名和邮箱地址

**RUN**: 容器构建时需要运行的命令

**EXPOSE**: 当前容器对外暴露的端口

**WORKDIR**: 制定在容器创建之后，终端默认登陆进来的工作目录，一个落脚点

**ENV**: 用来在构建镜像过程中设置环境变量

**ADD**: 将宿主机目录下的文件拷贝至镜像并且add命令会自动处理url和解压tar压缩包

**COPY**: 类似ADD命令，拷贝文件和目录到镜像中。将从构建上下文目录中（源路径）的文件/目录复制到新的层镜像内的<目标路径>位置

格式: copy src dest

```
copy ["src","dest"]
```

**VOLUME**: 容器数据卷，用于数据保存和数据持久化的工作

**CMD**: 制定一个容器启动的时候需要运行的命令，Dockerfile 中可以有多个CMD命令，但是只有最后一个生效，CMD会被docker run之后参数所替换

注：CMD容器启动命令的格式和run相似，也是两种格式：

- shell格式：CMD命令
- exec格式：CMD ["可执行文件","参数1","参数2"...]
- 参数格式列表：CMD ["参数1","参数2"...]，在制定了ENTRYPOINT指令后，用CMD制定具体的参数

**ENTRYPOINT**: 制定一个容器启动的时候要运行的具体命令，和CMD命令一样，都是制定容器启动时命令及参数

**ONBUILD**: 当构建一个被继承的Dockerfile 时运行的命令，父镜像在被子继承后，父镜像的ONBUILD被触发

**总结：**

BUILD	Both	RUN
FROM	WORKDIR	CMD
MAINTAINER	USER	ENV
COPY		EXPOSE
ADD		VOLUME
RUN		ENTRYPOINT
ONBUILD		
.dockerignore		

## 5. Dockerfile使用例子

Docker Hub 中99%的镜像都是通过在base镜像中安装和配置需要的软件构建出来的

### 1. 自定义镜像mycentos

#### 编写Dockerfile文件

```

FROM centos
MAINTAINER lidonghao<861914994@qq.com>
ENV MYPATH /usr/local
WORKDIR $MYPATH
RUN yum -y install vim
RUN yum -y install net-tools
EXPOSE 80
CMD echo $MYPATH
CMD echo "successful ...."
CMD /bin/bash

```

#### 构建

```
docker build -f Dockerfile -t mycentos:1.3 .
docker build -f mycentos_dockerfile -t mycentos:1.3 .
```

构建成功

```
Step 8/10 : CMD echo $MYPATH
--> Running in 67cae1f6c0d
--> 855e0e488c2f
Removing intermediate container 67cae1
Step 9/10 : CMD echo "successful ....."
--> Running in 102be0b9ebda
--> f4e21306ea2a
Removing intermediate container 102be0b
Step 10/10 : CMD /bin/bash
--> Running in 6f5005617257
--> ebd165fd8788
Removing intermediate container 6f50056
Successfully built ebd165fd8788
[root@12m5eb96jej4tmf30u4lnrz dockertil
REPOSITORY TAG
```

运行

```
docker run -it mycentos:1.3
```

#列出镜像的变更历史命令

```
docker history 镜像名称
```

## 2. CMD/ENTRYPOINT

这两个命令都是制定一个容器启动时需要运行的命令

### CMD命令

Dockerfile中可以有多个CMD命令，但是只有最后一个会生效，  
CMD命令会被docker run之后的参数替换

**测试：**tomcat的Dockerfile文件最后一行为 `CMD ["catalina.sh", "run"]`, 正常运行tomcat `docker run -it -p 8888:8080 tomcat` 会打印tomcat的日志，但是我们运行命令 `docker run -it -p 8888:8080 tomcat ls -l` 则会列出当工作目录下的文件。这个时候 `ls -l` 即为 `docker run` 之后的参数。

## ENTRYPOINT命令

`docker run` 之后的参数会当作参数传递给ENTRYPOINT,之后形成新的命令组合。

**测试：**制作可以查询IP的容器

## 编写Dockerfile文件

```
FROM centos
RUN yum install -y curl
CMD ["curl", "-s", "http://ip.cn"]
```

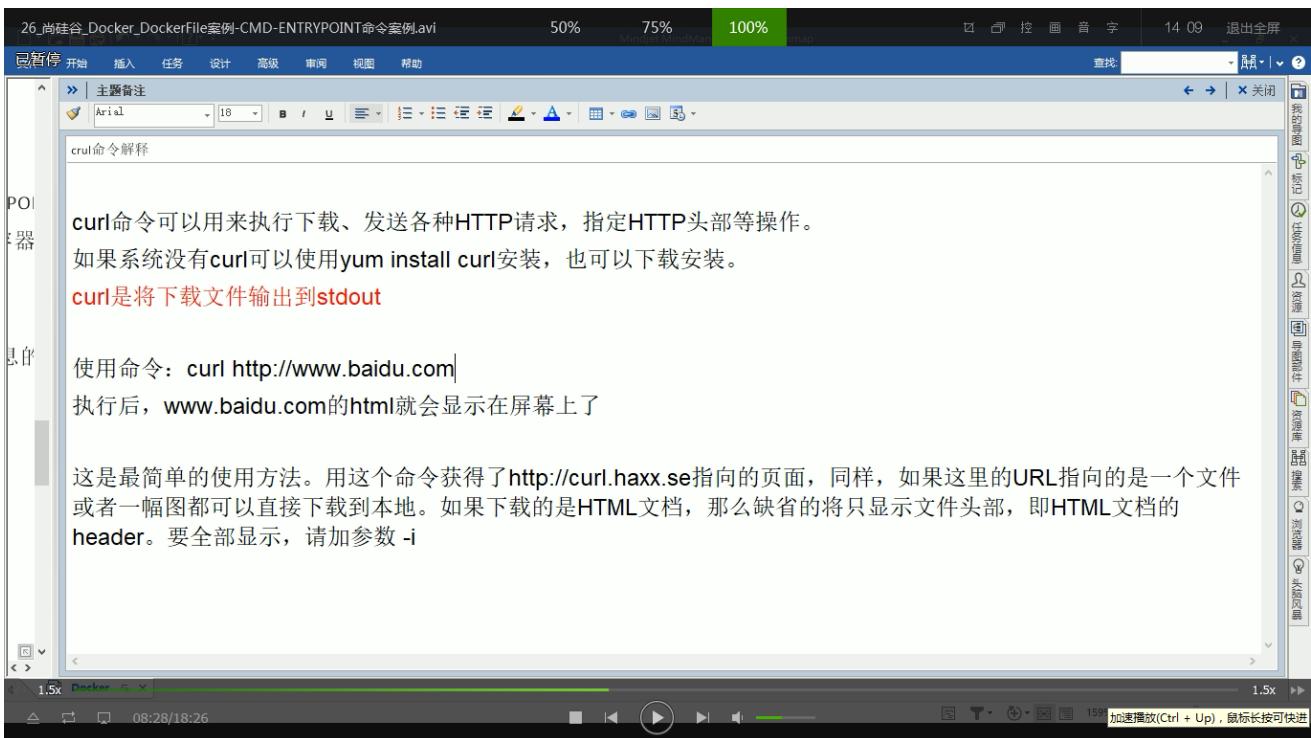
## 构建

```
docker build -f ip_dockerfile -t myip .
```

## 运行

```
docker run -it myip
```

## CURL命令说明：



如果我们还想显示文件头，这个时候就需要加上-i参数

```
curl -s -i http://ip.cn
```

但是这个时候在 `docker run -it myip` 之后加上 -i 参数会报错，因为我们在 dockerfile 中使用的时 CMD, 加上 -i 参数之后会替换 dockerfile 文件中的 CMD 命令。这个时候我们在 dockerfile 文件中改用 ENTRYPOINT 命令。他会将我们传的 -i 参数拼接在 ENTRYPOINT 命令中形成新的命令，并且执行。

编写 ENTRYPOINT 版的 dockerfile

```
FROM centos
RUN yum install -y curl
ENTRYPOINT ["curl", "-s", "http://ip.cn"]
```

构建

```
docker build -f ip_dockerfile2 -t myip2 .
```

## 运行

```
docker run myip2          #不显示文件头  
docker run myip2 -i      #显示文件头
```

## 3. ONBUILD示例

构建父镜像：

### (1) dockerfile

```
FROM centos  
RUN yum install -y curl  
ENTRYPOINT ["curl","-s","http://ip.cn"]  
ONBUILD RUN echo "this is father images....."
```

### (2)构建

```
docker build -f father_dockerfile -t father .
```

### (3)运行

```
docker run father
```

构建子镜像：

### (1) dockerfile

```
FROM father
RUN yum install -y curl
ENTRYPOINT ["curl","-s","http://ip.cn"]
```

## (2)构建

```
docker build -f son_dockerfile -t son .
```

**构建子镜像的时候会执行父镜像dockerfile文件中的ONBUILD命令**

## (3)运行

```
docker run son
```

## 4. 自定义tomcat

### 1. 编写Dockerfile文件(文件名称Dockerfile)

```
#基础镜像为centos
FROM centos
#设置维护者及维护者邮箱
MAINTAINER lidonghao<861914994@qq.com>
#将aa.txt文件拷贝至容器中/usr/local文件夹下并且改名为
mm.txt
COPY aa.txt /usr/local/mm.txt
#将tomcat和jdk压缩包拷贝至容器中/usr/local下并且解压
ADD apache-tomcat-8.0.26.tar.gz /usr/local/
ADD jdk-8u60-linux-x64.gz /usr/local/
#安装vim工具
RUN yum -y install vim
#设置环境变量
ENV MYPATH /usr/local
```

```
#制定工作目录
WORKDIR $MYPATH
#设置jdk和tomcat的环境变量
ENV JAVA_HOME /usr/local/jdk1.8.0_60
ENV CLASSPATH
$JAVA_HOME/lib/rt.jar:$JAVA_HOME/lib/tools.jar
ENV CATALINA_HOME /usr/local/apache-tomcat-8.0.26
ENV CATALINA_BASE /usr/local/apache-tomcat-8.0.26
ENV PATH
$PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/
bin
#制定对外暴露端口为8080
EXPOSE 8080
#制定容器启动的时候启动tomcat并且打印日志
CMD /usr/local/apache-tomcat-8.0.26/bin/startup.sh &&
tail -F /usr/local/apache-tomcat-
8.0.26/bin/logs/catalina.out
```

## 2.构建

```
docker build -t ldhtomcat9 .
```

这儿dockerfile文件名称为Dockerfile，所以不需要我们使用-f参数进行dockerfile文件的指定

## 3.运行

```
#-v设置容器卷
```

```
docker run -d -p 8089:8080 --name mytomcat99 -v  
/data/docker/app/mytomcat9/test:/usr/local/apache-  
tomcat-8.0.26/webapps/test -v  
/data/docker/app/mytomcat9/logs/:/usr/local/apache-  
tomcat-8.0.26/logs --privileged=true ldtomcat9
```

#### 4.查看

```
docker exec e117d8c99d9c java -version  
docker exec e117d8c99d9c pwd  
docker exec e117d8c99d9c ls -l  
docker exec e117d8c99d9c cat mm.txt
```

#### 5.web项目的发布

在/data/docker/app/mytomcat9/test目录下分别创建a.jsp文件和WEB-INF文件夹，在WEB-INF文件夹中创建web.xml文件

##### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xmlns="http://java.sun.com/xml/ns/javaee"  
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"  
id="WebApp_ID" version="3.0">  
<display-name>test</display-name>  
</web-app>
```

##### a.jsp

```
<%@ page language="java" contentType="text/html;
charset=utf-8" pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>

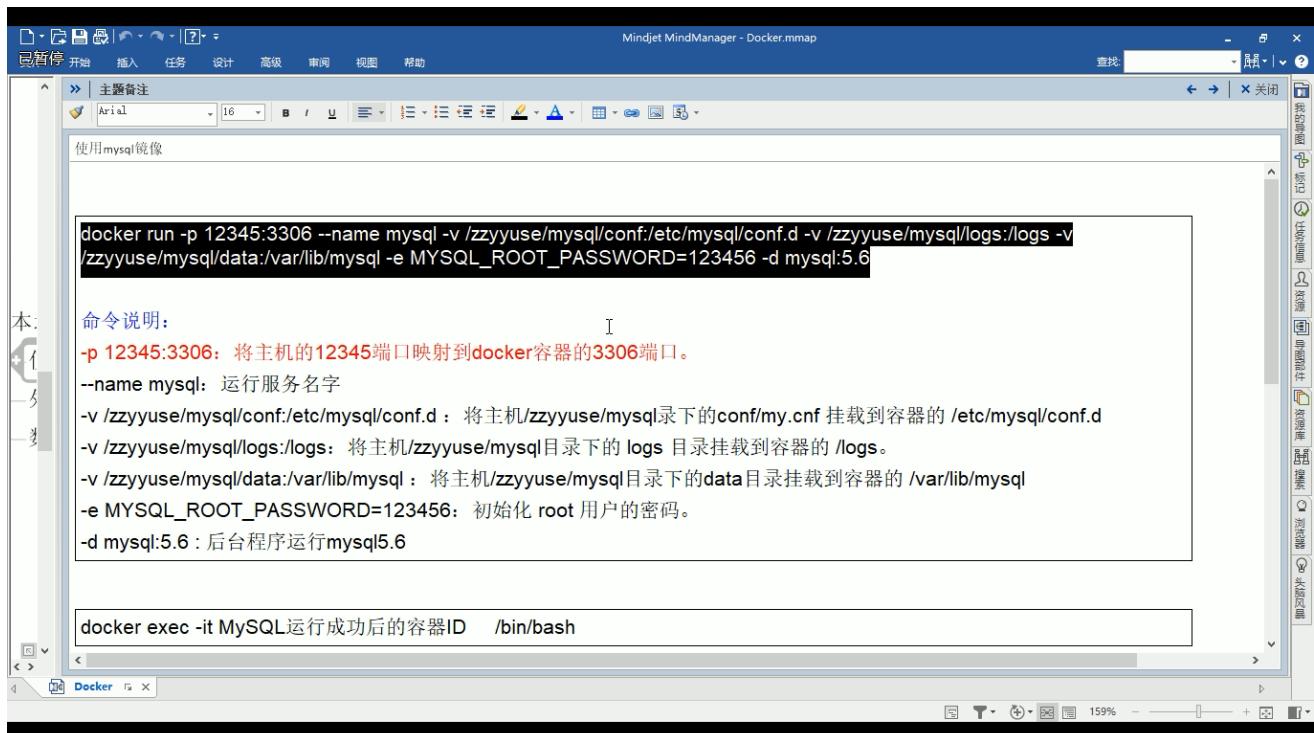
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>测试页面</title>
</head>
<body>
-----welocme-----
<%"I am in docker tomcat"%>
<br>
<br>
<% System.out.println("Hello World");%>
</body>
</html>
```

然后重启mytomcat99 `docker restart d0a77d8ce634`

浏览器访问<http://47.105.103.45:8089/test/a.jsp>

## 5.docker上安装mysql

```
docker run -p 3306:3306 --name mysql \
-v /data/docker/app/mysql5.6/conf:/etc/mysql/conf.d \
-v /data/docker/app/mysql5.6/logs:/logs \
-v /data/docker/app/mysql5.6/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.6
```



安装完执行 `docker exec -it 15a4658343ff /bin/bash` 进入交互式

```
mysql -uroot -p #连接mysql
```

数据库备份命令测试

```
docker exec 15a4658343ff sh -c 'exec mysqldump --all-databases -uroot -p "123456"' > /data/docker/app/mysql5.6/all-database.sql
```

6.docker上安装redis

```
docker run -p 6379:6379 -v  
/data/docker/app/myredis/data:/data -v  
/data/docker/app/myredis/conf/redis.conf:/usr/local/etc  
/redis/redis.conf -d redis redis-server  
/usr/local/etc/redis/redis.conf --appendonly yes
```

在主机/data/docker/app/myredis/conf/redis.conf文件夹下新建文件redis.conf。这个是redis的配置文件

连接redis:docker exec -it 7c4d512635a9 redis-cli

退出redis shutdown

/data/docker/app/myredis/data 目录下是redis中数据存放的位置。

## 七、将本地镜像推送到阿里云

### 1.根据容器创建镜像

```
docker commit -a ldh -m "new vim centos 1.4"  
b5ba2cd1f8c7 mycentos:1.4
```

查看我们新生成的镜像

dockers ps

运行 docker run -it mycentos:1.4

### 2.将本地镜像推送至阿里云

登陆阿里云镜像仓库管理控制台<https://cr.console.aliyun.com>

创建镜像仓库的时候代码源暂时选择本地仓库

步骤： 1. 登陆阿里云

```
sudo docker login --username=lidonghao4 registry.cn-
hangzhou.aliyuncs.com
```

2.

```
sudo docker tag [ImageId] registry.cn-
hangzhou.aliyuncs.com/lidonghao/mycentos:[镜像版本号]
#命令模板
```

```
docker tag 532033853ef3 registry.cn-
hangzhou.aliyuncs.com/lidonghao/mycentos:1.4      #
命令示例
```

3.

```
sudo docker push registry.cn-
hangzhou.aliyuncs.com/lidonghao/mycentos:[镜像版本号]
#命令模板
```

```
#命令示例
docker push registry.cn-
hangzhou.aliyuncs.com/lidonghao/mycentos:1.4
```