

"为了拯救电脑的内存"

感谢狂神视频所教授的docker

视频地址是【狂神说Java】Docker最新超详细版教程通俗易懂哔哩哔哩bilibili

# Docker总结笔记

---

{% note blue 'fas fa-bullhorn' simple %}

## (1)基本介绍

{% endnote %}

Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从 Apache2.0 协议开源。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化

容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app），更重要的是容器性能开销极低。

Docker 从 17.03 版本之后分为 CE (Community Edition: 社区版) 和 EE (Enterprise Edition: 企业版)，用社区版就可以了。官网：<https://docs.docker.com/>

{% note red 'fas fa-fan' flat%}

## (2)应用场景

{% endnote %}

- Web 应用的自动化打包和发布。
- 自动化测试和持续集成、发布。
- 在服务型环境中部署和调整数据库或其他的后台应用。
- 从头编译或者扩展现有的 OpenShift 或 Cloud Foundry 平台来搭建自己的 PaaS 环境。

{% note orange 'fas fa-battery-half' flat %}

## (3)Docker 的优势

{% endnote %}

Docker 是一个用于开发，交付和运行应用程序的开放平台。Docker 使您能够将应用程序与基础架构分开，从而可以快速交付软件。借助 Docker，您可以与管理应用程序相同的方式来管理基础架构。通过利用 Docker 的方法来快速交付，测试和部署代码，您可以大大减少编写代码和在生产环境中运行代码之间的延迟。

1、快速，一致地交付您的应用程序。

Docker 允许开发人员使用您提供的应用程序或服务的本地容器在标准化环境中工作，从而简化了开发的生命周期。

容器非常适合持续集成和持续交付 (CI / CD) 工作流程，请考虑以下示例方案：

您的开发人员在本地编写代码，并使用 Docker 容器与同事共享他们的工作。他们使用 Docker 将其应用程序推送到测试环境中，并执行自动或手动测试。当开发人员发现错误时，他们可以在开发环境中对其进行修复，然后将其重新部署到测试环境中，以进行测试和验证。

测试完成后，将修补程序推送给生产环境，就像将更新的镜像推送到生产环境一样简单。

## 2、响应式部署和扩展

Docker 是基于容器的平台，允许高度可移植的工作负载。Docker 容器可以在开发人员的本机上，数据中心的物理或虚拟机上，云服务上或混合环境中运行。

Docker 的可移植性和轻量级的特性，还可以使您轻松地完成动态管理的工作负担，并根据业务需求指示，实时扩展或拆除应用程序和服务。

## 3、在同一硬件上运行更多工作负载

Docker 轻巧快速。它为基于虚拟机管理程序的虚拟机提供了可行、经济、高效的替代方案，因此您可以利用更多的计算能力来实现业务目标。Docker 非常适合于高密度环境以及中小型部署，而您可以用更少的资源做更多的事情。

# 虚拟化技术和容器化技术

---

{% note purple 'far fa-hand-scissors' flat %}

虚拟化技术特点：

{% endnote %}

1.资源占用多

2.冗余步骤多

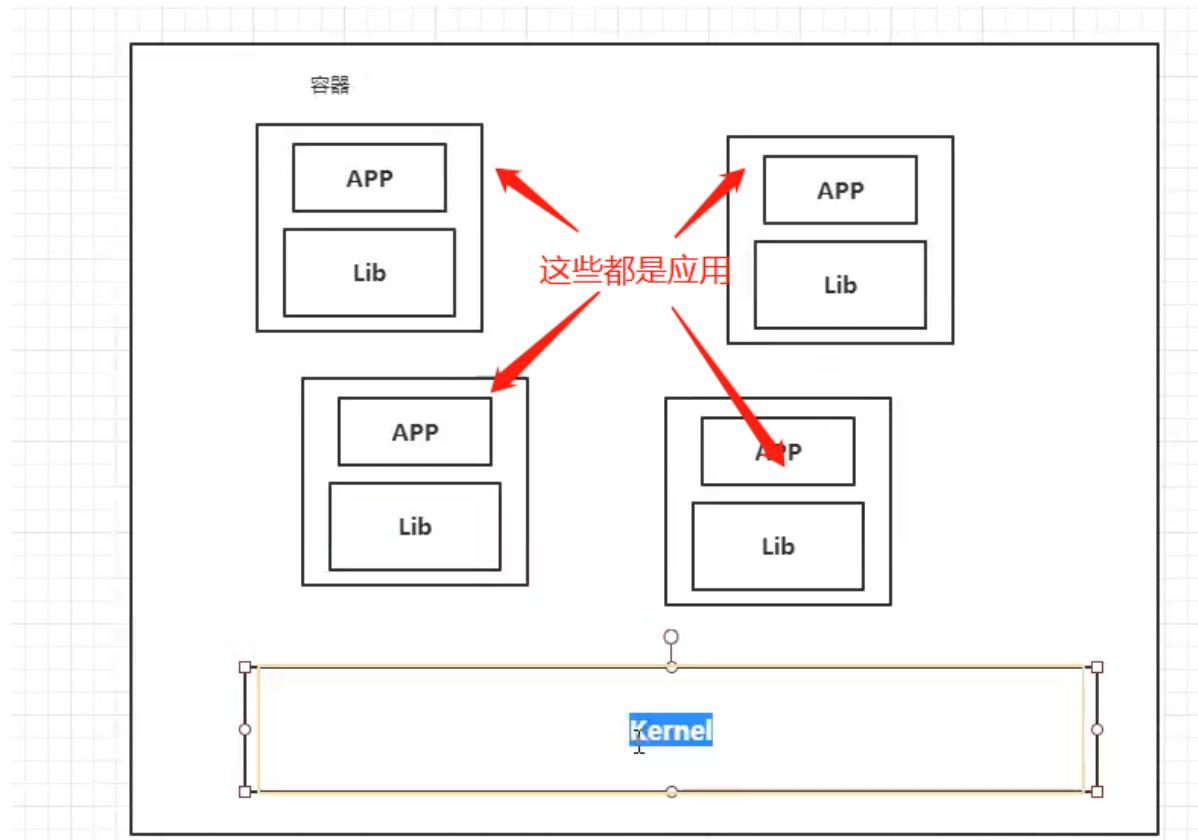
3.启动很慢

{% note green 'fab fa-internet-explorer' flat %}

容器化技术：

{% endnote %}

容器化技术**不是**模拟的一个完整的操作系统



比较Docker和虚拟机的不同：

- 1.传统虚拟机，虚拟出硬件，运行一个完整的操作系统，然后在这个系统上安装和运行软件。
- 2.Docker容器内的应用直接运行在宿主机的内容，容器是没有自己的内核的（所有容器公用内核），也没有虚拟硬件。
- 3.每个容器都是相互隔离的，每个容器都有属于自己的文件系统，互不影响。

容器化带来的好处：

#### **应用更快速的交付和部署**

传统：通过运维读一堆帮助文档，再安装程序

Docker：可以打包镜像发布测试，一键运行

#### **更便捷的升级和扩缩容**

使用了Docker之后，我们部署应用就和搭积木一样！

项目打包为一个镜像可以扩展成不同的容器

#### **更简单的系统运维**

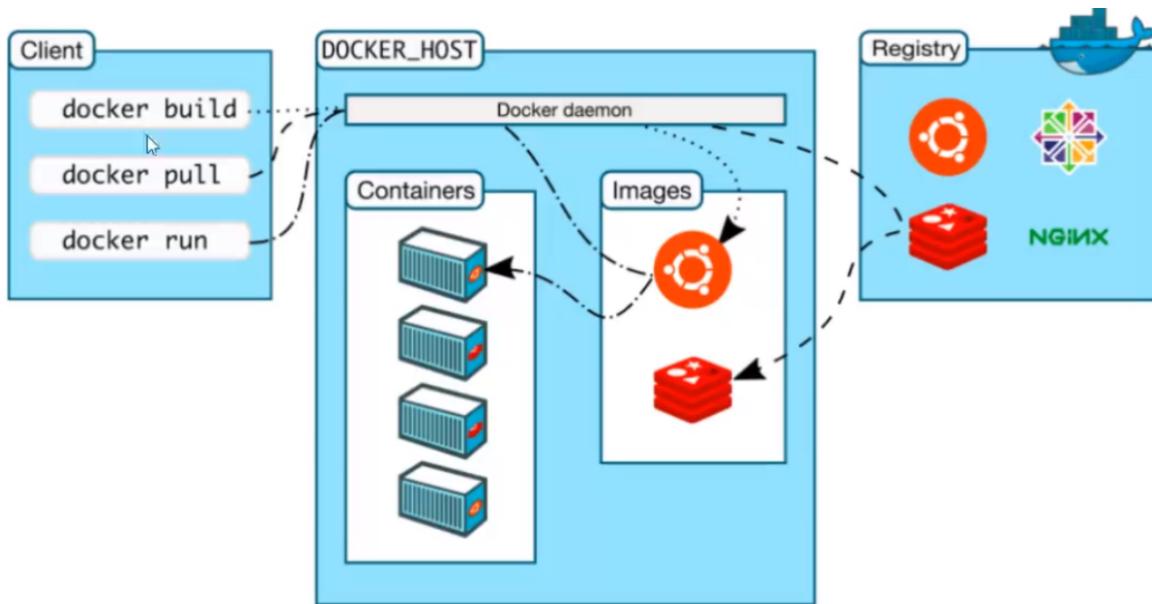
在容器化之后，我们的开发、测试环境都是高度一致的。不会出现“在你的电脑能用，在我的电脑不能用”

#### **更高效的计算资源利用：**

Docker是内核级别的虚拟化，可以在一个物理机上运行很多的容器实例！服务器的性能可以被榨到极致。

## **Docker的基本组成**

Docker的基本组成图如下：



专业名词解释:

镜像( image) :

docker镜像就好比是一个模板,可以通过这个模板来创建容器服务, tomcat镜像==> run ==> tomcat01 容器(提供服务器), (镜像并没有改变, 改变的是容器里面的内容, 进一步保证镜像是不被污染的) 通过这个镜像可以创建多个容器(最终服务运行或者项目运行就是在容器中的)。

容器( container) :

Docker利用容器技术,独立运行一个或者一个组应用,通过镜像来创建的。

目前就可以把这个容器理解为就是一个简易的linux系统 (因为是基于linux内核的)

仓库( repository) :

仓库就是存放镜像的地方! (可以存放别人上传的镜像或者你自己创建的镜像)

仓库分为公有仓库和私有仓库!

## Docker的安装

查看系统的内核:

```
[root@izwz99sm8v95sckz8bd2c4z ~]# uname -r
3.10.0-957.21.3.el7.x86_64
```

查看系统配置

cat /etc/os-release

```
[root@izwz99sm8v95sckz8bd2c4z ~]# cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
```

```
CENTOS_MANTISBT_PROJECT_VERSION="7"  
REDHAT_SUPPORT_PRODUCT="centos"  
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

## Docker的安装步骤：

### (1) 卸载旧的版本 (如果之前下载过docker)

```
yum remove docker \  
    docker-client \  
    docker-client-latest \  
    docker-common \  
    docker-latest \  
    docker-latest-logrotate \  
    docker-logrotate \  
    docker-engine
```

### (2) 下载需要的安装包 (docker需要这些安装包)

```
yum install -y yum-utils
```

### (3) 设置镜像的仓库

```
yum-config-manager \  
    --add-repo \  
    https://download.docker.com/linux/centos/docker-ce.repo #国外的地址 (用下面这条  
指令)  
  
# 设置阿里云的Docker镜像仓库  
yum-config-manager \  
    --add-repo \  
    https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo #国外的地址
```

### (4) 更新yum软件包索引

```
yum makecache fast
```

### (5) 安装docker相关的配置

docker-ce 是社区版， docker-ee 企业版

```
yum install docker-ce docker-ce-cli containerd.io
```

出现了completed即安装成功。

### (6) 启动Docker

```
systemctl start docker  
# 查看当前版本号，是否启动成功  
docker version  
# 设置开机自启动  
systemctl enable docker
```

# Docker的卸载

```
# 1. 卸载依赖  
yum remove docker-ce docker-ce-cli containerd.io  
# 2. 删除资源 . /var/lib/docker是docker的默认工作路径  
rm -rf /var/lib/docker
```

## 配置阿里云镜像加速

(1) 进入阿里云官网，搜索容器镜像服务（这个地方每个人是不一样的，可以根据速度需求执行这条命令）

The screenshot shows the Alibaba Cloud Container Registry interface. On the left, there's a sidebar with various options like 'Container Registry', 'Default Instances', 'Image Library', etc. The 'Mirror Accelerator' option is highlighted with a red box. In the main content area, there's a note about using an accelerator to speed up Docker image download. Below it, the 'Accelerator Address' is set to 'https://axvfsf7e.mirror.aliyuncs.com'. A 'Copy' button is next to the address. Further down, under 'Operation Document', the 'Ubuntu' tab is selected. It provides two steps: 1. Install / Upgrade Docker Client, which recommends version 1.10.0 or higher and links to the 'docker-ce' documentation; 2. Configure Registry Accelerator. This step includes a note for Docker clients with versions above 1.10.0, a warning about modifying the daemon configuration file, and a command-line snippet:

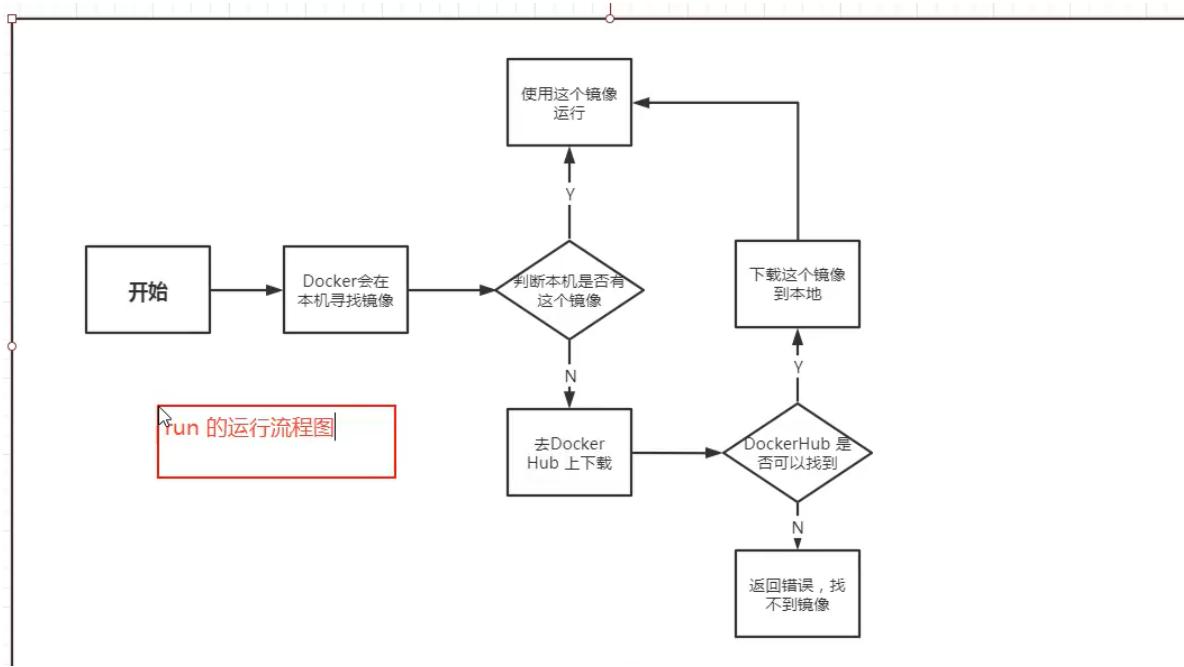
```
sudo mkdir -p /etc/docker  
sudo tee /etc/docker/daemon.json << EOF  
{  
    "registry-mirrors": ["https://axvfsf7e.mirror.aliyuncs.com"]  
}  
EOF  
sudo systemctl daemon-reload  
sudo systemctl restart docker
```

(2)依次执行官方的这四条命令

```
sudo mkdir -p /etc/docker  
  
sudo tee /etc/docker/daemon.json << 'EOF'  
{  
    "registry-mirrors": ["https://axvfsf7e.mirror.aliyuncs.com"]  
}  
EOF  
  
sudo systemctl daemon-reload  
  
sudo systemctl restart docker
```

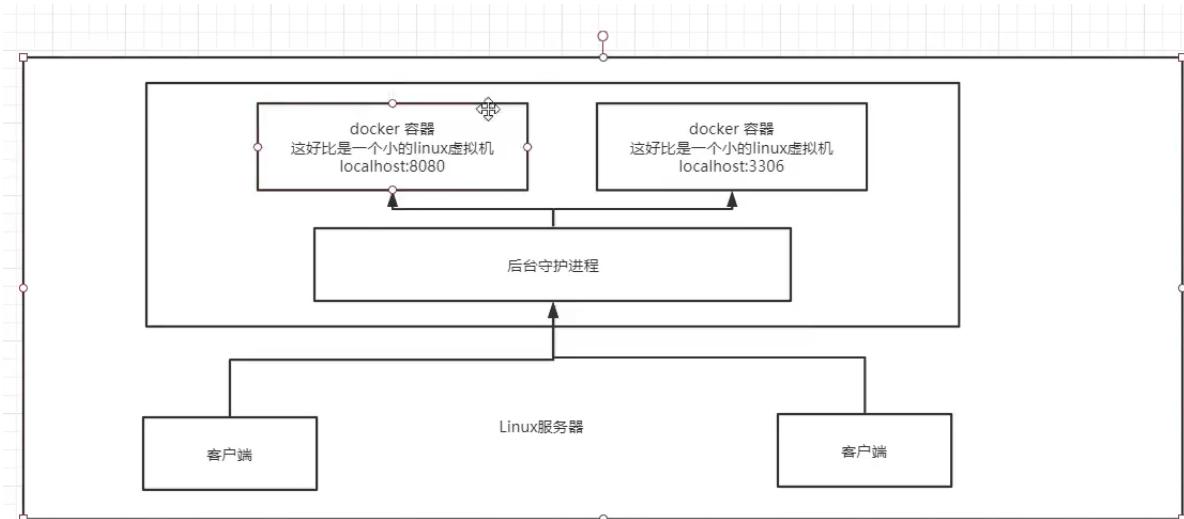
## Docker容器运行流程

启动一个容器，Docker的运行流程如下图：



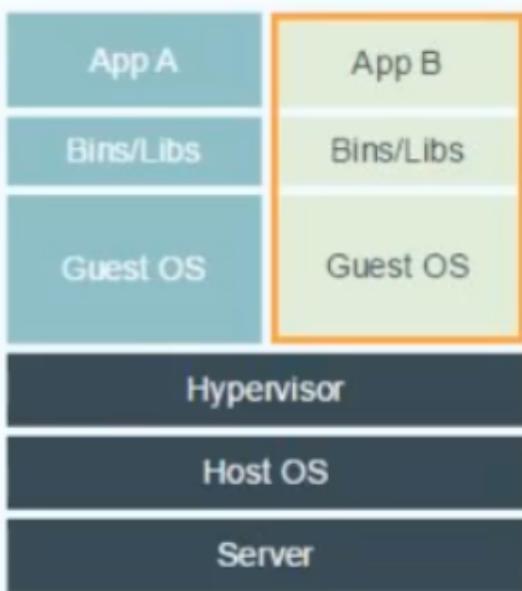
## 底层原理

Docker是一个Client-Server结构的系统，Docker的守护进程运行在主机上，通过Socket从客户端访问！Docker Server接收到Docker-Client的指令，就会执行这个指令



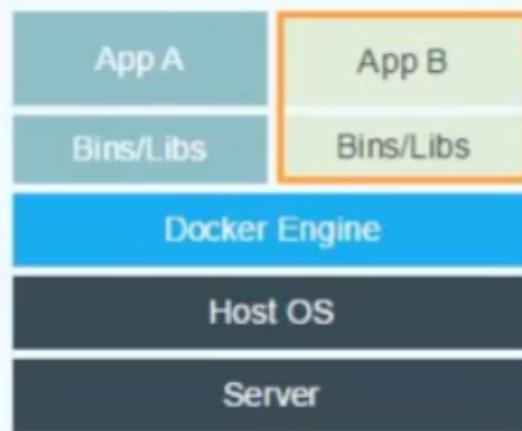
Docker为什么比VM Ware快？

- 1、Docker比虚拟机更少的抽象层
- 2、docker利用宿主机的内核，VM需要的是Guest OS (VM还需要下载内核)



VM (Virtual Machine)

Docker新建一个容器的时候，不需要像虚拟机一样重新加载一个操作系统内核，直接利用宿主机的操作系统，而虚拟机是需要加载Guest OS。



Docker

Docker和VM的对比如下：

	Docker容器	LXC	VM
虚拟化类型	OS虚拟化	OS虚拟化	硬件虚拟化
性能	=物理机性能	=物理机性能	5%-20%损耗
隔离性	NS 隔离	NS 隔离	强
QoS	Cgroup 弱	Cgroup 弱	强
安全性	中	差	强
GuestOS	只支持Linux	只支持Linux	全部

<https://blog.csdn.net/huangjhai>

## Docker常用命令

### 基础命令

docker version	#查看docker的版本信息
docker info	#查看docker的系统信息，包括镜像和容器的数量
docker 命令 --help	#帮助命令(可查看可选的参数)
docker COMMAND --help	

命令的帮助文档地址:<https://docs.docker.com/engine/reference/commandline/docker/>

## 镜像命令

### 1.docker images 查看本地主机的所有镜像

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
hello-world    latest        bf756fb1ae65  11 months ago  13.3kB
```

#解释：

1.REPOSITORY 镜像的仓库源（名字，可以通过这个名字进行下载）

2.TAG 镜像的标签

3.IMAGE ID 镜像的id

4.CREATED 镜像的创建时间

5.SIZE 镜像的大小

# 可选参数（参数添加在命令后面）

#docker images -a

-a/--all 列出所有镜像

-q/--quiet 只显示镜像的id

### 2.docker search 搜索镜像

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker search mysql
NAME                           DESCRIPTION
STARS      OFFICIAL      AUTOMATED
mysql           MySQL is a widely used, open-source relation...
10308       [OK]
mariadb        MariaDB is a community-developed fork of Mys...
3819        [OK]
mysql/mysql-server      Optimized MySQL Server Docker images. Create...
754         [OK]
percona         Percona Server is a fork of the MySQL relati...
517         [OK]
centos/mysql-57-centos7     MySQL 5.7 SQL database server
86
mysql/mysql-cluster      Experimental MySQL Cluster Docker images. Cr...
79
centurylink/mysql        Image containing mysql. Optimized to be link...
60         [OK]
```

#可选参数

Search the Docker Hub [for](#) images

Options:

[-f, --filter](#) filter Filter output based on conditions provided

```
--format string    Pretty-print search using a Go template
--limit int        Max number of search results (default 25)
--no-trunc         Don't truncate output
```

#搜索收藏数大于3000的镜像

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker search mysql --filter=STARS=3000
NAME          DESCRIPTION                           STARS      OFFICIAL      AUTOMATED
mysql         MySQL is a widely used, open-source relation...  10308      [OK]
mariadb       MariaDB is a community-developed fork of MyS...  3819       [OK]
```

```
[root@VM-4-3-centos /]# docker search mysql -f=stars=3000
NAME          DESCRIPTION                           STARS      OFFICIAL      AUTOMATED
mysql         MySQL is a widely used, open-source relation...  12976      [OK]
mariadb       MariaDB Server is a high performing open sou...  4972       [OK]
[root@VM-4-3-centos /]#
```

### 3.docker pull 镜像名[:tag] 下载镜像

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker pull mysql
using default tag: latest          #如果不写tag默认就是latest
latest: Pulling from library/mysql
6ec7b7d162b2: Pull complete        #分层下载,docker image的核心-联合文件系统
fedd960d3481: Pull complete
7ab947313861: Pull complete
64f92f19e638: Pull complete
3e80b17bff96: Pull complete
014e976799f9: Pull complete
59ae84fee1b3: Pull complete
ffe10de703ea: Pull complete
657af6d90c83: Pull complete
98bfb480322c: Pull complete
6aa3859c4789: Pull complete
1ed875d851ef: Pull complete
Digest: sha256:78800e6d3f1b230e35275145e657b82c3fb02a27b2d8e76aac2f5e90c1c30873
#签名
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest #下载来源的真实地址
#所以
#docker pull mysql==docker pull docker.io/library/mysql:latest
```

### 指定版本下载

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker pull mysql:5.7
5.7: Pulling from library/mysql
6ec7b7d162b2: Already exists
fedd960d3481: Already exists
7ab947313861: Already exists
64f92f19e638: Already exists
3e80b17bff96: Already exists
014e976799f9: Already exists
59ae84fee1b3: Already exists
7d1da2a18e2e: Pull complete
301a28b700b9: Pull complete
529dc8dbeaf3: Pull complete
```

```
bc9d021dc13f: Pull complete
Digest: sha256:c3a567d3e3ad8b05dfce401ed08f0f6bf3f3b64cc17694979d5f2e5d78e10173
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

#### 4.docker rmi 删除镜像

```
#1.删除指定的镜像id
[root@izwz99sm8v95sckz8bd2c4z ~]# docker rmi -f 镜像id
#2.删除多个镜像id
[root@izwz99sm8v95sckz8bd2c4z ~]# docker rmi -f 镜像id 镜像id 镜像id
#3.删除全部的镜像id
[root@izwz99sm8v95sckz8bd2c4z ~]# docker rmi -f $(docker images -aq)
```

\$(docker images -aq)查询所有镜像

## 容器命令

如拉取一个centos镜像

```
docker pull centos
```

运行容器的命令说明：

```
docker run [可选参数] image

#参数说明
--name="名字"          指定容器名字
-d                      后台方式运行
-it                     使用交互方式运行,进入容器查看内容
-p                      指定容器的端口
(
    -p ip:容器端口:配置主机端口映射到容器端口
    -p 主机端口:容器端口 (通过主机端口进入容器端口)
    -p 容器端口
)
-P                      随机指定端口(大写的P)
```

运行并进入容器centos

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker run -it centos /bin/bash
[root@bd1b8900c547/]# ls
bin dev etc home lib lib64 lost+found media mnt opt proc root run
sbin srv sys tmp usr var
```

退出容器命令：

```
#exit 停止并退出容器 (后台方式运行则仅退出)
#Ctrl+p+q 不停止容器退出
[root@bd1b8900c547/]# exit
exit
```

列出运行过的容器命令：

```
#docker ps # 列出当前正在运行的容器
```

```
-a    # 列出所有容器的运行记录  
-n=? # 显示最近创建的n个容器  
-q    # 只显示容器的编号
```

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES  
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps -a  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES  
          PORTS           NAMES  
bca129320bb5      centos             "/bin/bash"       4 minutes ago     Exited (0) 3 minutes ago  
ago                 optimistic_shtern  
bd1b8900c547      centos             "/bin/bash"       6 minutes ago     Exited (0) 5 minutes ago  
ago                 cool_tesla  
cf6adbfb1b506     bf756fb1ae65   "/hello"          5 hours ago      Exited (0) 5 hours ago  
ago                 optimistic_darwin
```

删除容器命令：

```
docker rm 容器id          #删除指定的容器,不能删除正在运行的容器,强制删除使用 rm -f  
docker rm -f $(docker ps -aq)  #删除所有的容器  
docker ps -a -q|xargs docker rm #删除所有的容器
```

启动和停止容器命令：

```
docker start 容器id      #启动容器  
docker restart 容器id     #重启容器  
docker stop 容器id        #停止当前运行的容器  
docker kill 容器id        #强制停止当前容器
```

## 其他常用命令

### 日志的查看

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker logs --help

Usage: docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

Options:
  --details            Show extra details provided to logs
  -f, --follow         Follow log output
  --since string      Show logs since timestamp (e.g. 2013-01-02T13:23:37Z) or
relative (e.g. 42m for 42 minutes)
  -n, --tail string   Number of lines to show from the end of the logs (default
"all")
  -t, --timestamps    Show timestamps
  --until string      Show logs before a timestamp (e.g. 2013-01-02T13:23:37Z)
or relative (e.g. 42m for 42 minutes)
```

常用：

```
docker logs -tf 容器id #查看xx容器的日志
docker logs --tail number 容器id #num为要显示的日志条数
```

```
#docker容器后台运行，必须要有一个前台的进程，否则会自动停止 docker发现自己没有提供服务，就会自动停止
#编写shell脚本循环执行，使得centos容器保持运行状态
[root@izwz99sm8v95sckz8bd2c4z ~]# docker run -d centos /bin/sh -c "while true;do
echo hi;sleep 5;done"
c703b5b1911ff84d584390263a35707b6024816e1f46542b61918a6327a570dc
```

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS     NAMES
c703b5b1911f        centos     "/bin/sh -c 'while t..."   13 seconds ago   Up 10 seconds
                                         pedantic_banach
```

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker logs -tf --tail 10 c703b5b1911f
2020-12-27T03:34:07.255599560Z hi
2020-12-27T03:34:12.257641517Z hi
2020-12-27T03:34:17.259706294Z hi
2020-12-27T03:34:22.261693707Z hi
2020-12-27T03:34:27.262609289Z hi
2020-12-27T03:34:32.267862677Z hi
2020-12-27T03:34:37.270382873Z hi
2020-12-27T03:34:42.272414182Z hi
2020-12-27T03:34:47.274823243Z hi
2020-12-27T03:34:52.277419274Z hi
```

## 10.4.2 查看容器中进程信息

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker top c703b5b1911f
UID          PID      PPID      C
STIME        TTY      TIME      CMD
root        11156    11135      0
11:31        ?      00:00:00  /bin/sh -c while
true;do echo hi;sleep 5;done
root        11886    11156      0
11:43        ?      00:00:00  /usr/bin/coreutils -
-coreutils-prog-shebang=sleep /usr/bin/sleep 5
```

## 查看容器的元数据(彻底了解容器里面有哪些数据)

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker inspect 容器id
```

## 进入当前正在运行的容器

因为通常我们的容器都是使用后台方式来运行的，有时需要进入容器修改配置

方式一：

```
[root@iZwz99sm8v95scKz8bd2c4z ~]# docker exec -it c703b5b1911f /bin/bash
[root@c703b5b1911f/]# ls
bin dev etc home lib lib64 lost+found media mnt opt proc root run
sbin srv sys tmp usr var
```

### 方式二：

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker attach c703b5b1911f
```

docker exec 进入容器后开启(相当于新开一个窗口)一个新的终端，可以在里面操作

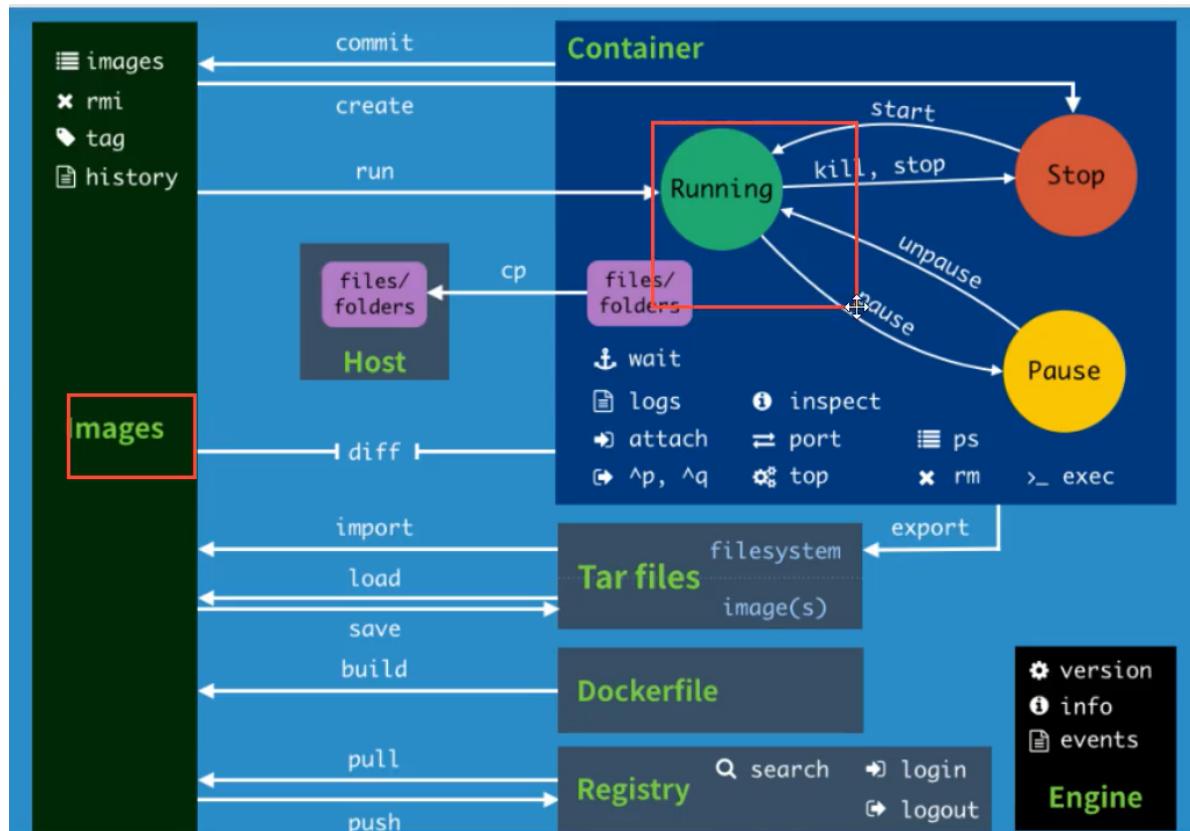
docker attach 进入容器正在执行的终端，不会启动新的进程

## 拷贝操作

拷贝操作的命令如下：

```
#拷贝容器的文件到主机中  
docker cp 容器id:容器内路径 目的主机路径  
  
#拷贝宿主机的文件到容器中  
docker cp 目的主机路径 容器id:容器内路径
```

命令小节的图解如下：



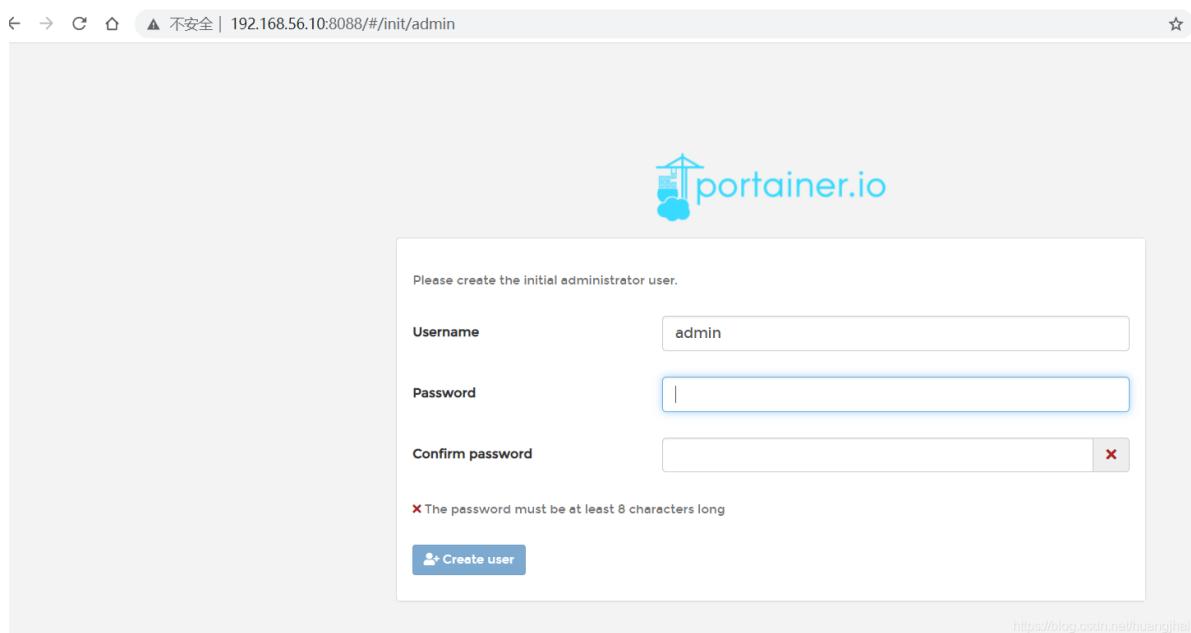
## 图形化管理工具Portaniner安装

Portaniner是Docker的图形化管理工具，类似的工具还有Rancher(CI/CD再用)

下载运行Portaniner镜像并运行，设置本机映射端口为8088

```
[root@localhost conf]# docker run -d -p 8088:9000 --restart=always -v
/var/run/docker.sock:/var/run/docker.sock --privileged=true portainer/portainer
Unable to find image 'portainer/portainer:latest' locally
latest: Pulling from portainer/portainer
94cfa856b2b1: Pull complete
49d59ee0881a: Pull complete
a2300fd28637: Pull complete
Digest: sha256:fb45b43738646048a0a0cc74fcee2865b69efde857e710126084ee5de9be0f3f
Status: Downloaded newer image for portainer/portainer:latest
8c525a0137be22965bd1e3944da622a2c4248f8ad20883f4b3ea4f8a6b11e163
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
7789d4505a00        portainer/portainer   "/portainer"      6 seconds ago     Up 5 seconds      0.0.0.0:8088->9000/tcp    quirky_sinoussi
```

### 第一次登录设置admin用户的密码



<https://blog.csdn.net/huanghai>

如果是阿里云服务器记得设置安全组，选择连接本地的Docker,整体界面预览如下图：

Id	Tags	Size
sha256:09361feeb4753ac9da80ead4d46e2b...	mysql:5.7	447 MB
sha256:580c0e4e98b06d258754cf28c55f21...	portainer/portainer:latest	79.1 MB

# Docker镜像详解

## 什么是镜像

镜像是一种轻量级、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要的所有内容，包括代码，运行时（一个程序在运行或者在被执行的依赖）、库，环境变量和配置文件。

所有的应用，直接打包docker镜像，就可以直接跑起来

## 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是UnionFS联合文件系统。

联合文件系统简单的来说就是每个应用都分成很多的小层，这些应用如果有相同的层的话可以公用

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

<https://blog.csdn.net/huangjihai>

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。

## 分层理解

Docker镜像分层结构的好处，莫过于资源共享，如果多个镜像都有一个基础镜像层，那只需要保留一份基础镜像层，就可以为所有包含这层的容器服务，而且镜像的每一层都可以被共享

并且镜像分层可以用docker image inspect命令查看

注意：

Docker镜像都是只读的,当容器启动时,一个新的可写层被加载到镜像的顶部!  
这一层就是我们通常说的容器层,容器之下的都叫镜像层!

## 提交镜像

使用docker commit 命令提交容器成为一个新的版本

```
docker commit -m="提交的描述信息" -a="作者" 容器id 目标镜像名:[TAG]
```

由于默认的Tomcat镜像的webapps文件夹中没有任何内容，需要从webapps.dist中拷贝文件到webapps文件夹。下面自行制作镜像：就是从webapps.dist中拷贝文件到webapps文件夹下，并提交该镜像作为一个新的镜像。使得该镜像默认的webapps文件夹下就有文件。具体命令如下：

#1. 复制文件夹

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker run -it tomcat /bin/bash
root@2a3bf3eaa2e4:/usr/local/tomcat# cd webapps
root@2a3bf3eaa2e4:/usr/local/tomcat/webapps# ls
root@2a3bf3eaa2e4:/usr/local/tomcat/webapps# cd ..
```

```

root@2a3bf3eaa2e4:/usr/local/tomcat# cp -r webapps.dist/* webapps
root@2a3bf3eaa2e4:/usr/local/tomcat# cd webapps
root@2a3bf3eaa2e4:/usr/local/tomcat/webapps# ls
ROOT docs examples host-manager manager
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS
PORTS           NAMES
2a3bf3eaa2e4  tomcat         "/bin/bash"   4 minutes ago Up 4 minutes
  8080/tcp      competent_torvalds
7789d4505a00  portainer/portainer  "/portainer"  24 hours ago Up 24 hours
  0.0.0.0:8088->9000/tcp  quirky_sinoussi
[root@izwz99sm8v95sckz8bd2c4z ~]# docker exec -it 2a3bf3eaa2e4 /bin/bash
root@2a3bf3eaa2e4:/usr/local/tomcat#
root@2a3bf3eaa2e4:/usr/local/tomcat# cd webapps
root@2a3bf3eaa2e4:/usr/local/tomcat/webapps# ls
ROOT docs examples host-manager manager
root@2a3bf3eaa2e4:/usr/local/tomcat/webapps# cd ../
root@2a3bf3eaa2e4:/usr/local/tomcat# read escape sequence
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS
PORTS           NAMES
2a3bf3eaa2e4  tomcat         "/bin/bash"   8 minutes ago Up 8 minutes
  8080/tcp      competent_torvalds
7789d4505a00  portainer/portainer  "/portainer"  24 hours ago Up 24 hours
  0.0.0.0:8088->9000/tcp  quirky_sinoussi

```

## #2. 提交镜像作为一个新的镜像

```

[root@izwz99sm8v95sckz8bd2c4z ~]# docker commit -m="add webapps" -a="Ethan"
2a3bf3eaa2e4 mytomcat:1.0
sha256:f189aac861de51087af5bc88a5f1de02d9574e7ee2d163c647dd7503a2d3982b
[root@izwz99sm8v95sckz8bd2c4z ~]# docker images
REPOSITORY          TAG        IMAGE ID      CREATED      SIZE
mytomcat            1.0       f189aac861de  7 seconds ago  653MB
mysql               5.7       f07dfa83b528  6 days ago   448MB
tomcat              latest    feba8d001e3f  10 days ago  649MB
nginx               latest    ae2fefff98a0c  12 days ago  133MB
centos              latest    300e315adb2f  2 weeks ago  209MB
portainer/portainer latest    62771b0b9b09  5 months ago  79.1MB
elasticsearch       7.6.2     f29a1ee41030  9 months ago  791MB

```

# 常用容器部署

## Nginx部署

### (1) 搜索并下载镜像

```

[root@izwz99sm8v95sckz8bd2c4z ~]# docker search nginx
NAME                  DESCRIPTION
STARS    OFFICIAL   AUTOMATED
nginx                official build of Nginx.
14207    [OK]

```

jwilder/nginx-proxy		Automated Nginx reverse proxy <b>for</b> docker con...
1932	[OK]	
richarvey/nginx-php-fpm		Container running Nginx + PHP-FPM capable of...
797	[OK]	
linuxserver/nginx		An Nginx container, brought to you by LinuxS...
137		
jc21/nginx-proxy-manager		Docker container <b>for</b> managing Nginx proxy ho...
123		
tiangolo/nginx-rtmp		Docker image with Nginx using the nginx-rtmp...
107	[OK]	

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
6ec7b7d162b2: Already exists
cb420a90068e: Pull complete
2766c0bf2b07: Pull complete
e05167b6a99d: Pull complete
70ac9d795e79: Pull complete
Digest: sha256:4cf620a5c81390ee209398ecc18e5fb9dd0f5155cd82adcbae532fec94006fb9
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker images;
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
mysql            5.7          f07dfa83b528   5 days ago    448MB
nginx           latest        ae2feff98a0c   11 days ago   133MB
centos          latest        300e315adb2f   2 weeks ago   209MB
```

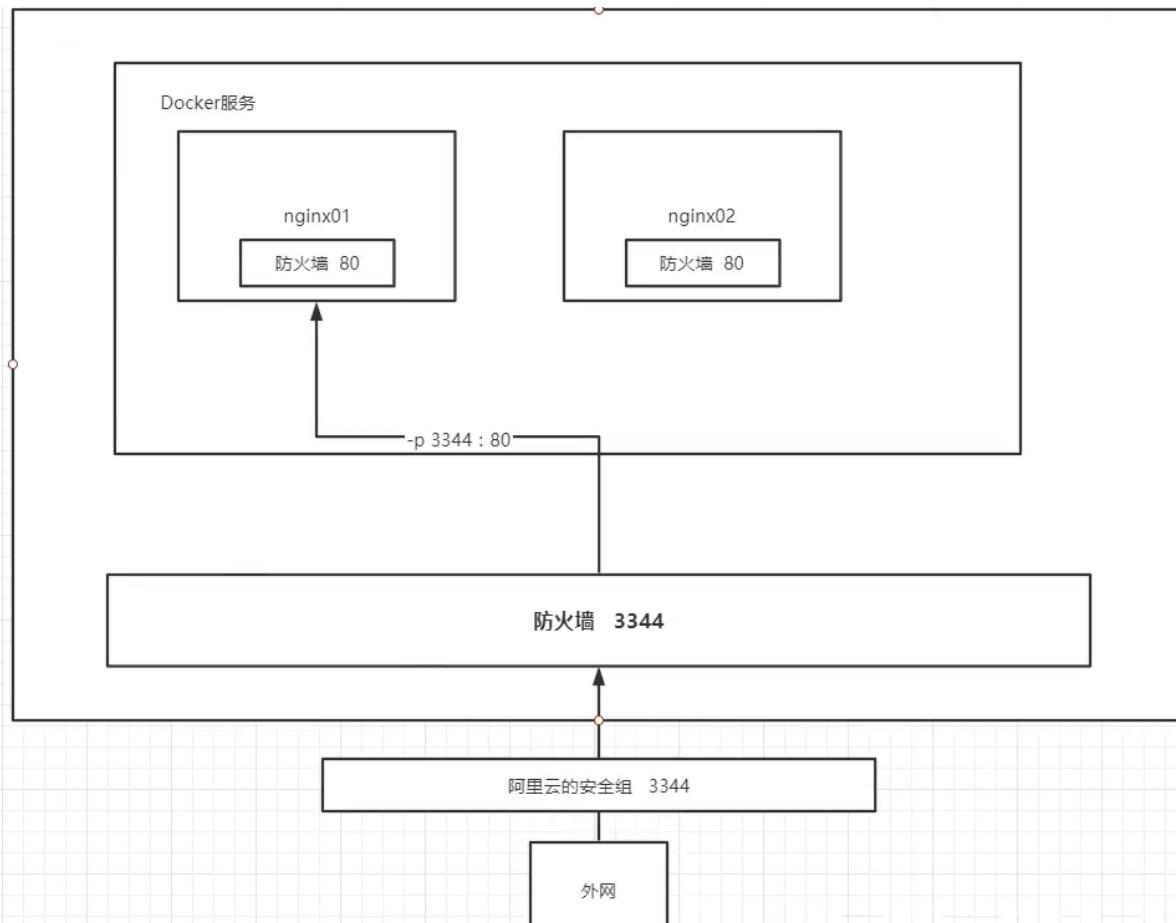
可以到dockerhub官网查看Nginx的详细版本信息：[https://hub.docker.com/\\_/nginx](https://hub.docker.com/_/nginx)

## (2) 运行测试

```
docker run -d --name nginx01 -p 3334:80 nginx
```

-d 后台运行  
--name 给容器命名  
-p 3334:80 将宿主机的端口3334映射到该容器的80端口

端口暴露的概念：



### (3)配置文件

进入容器，自定义配置文件（这里需要进入容器，相当于容器内部又是一个新的系统）

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker exec -it nginx01 /bin/bash
root@20c896637ff5:/# whereis nginx
nginx: /usr/sbin/nginx /usr/lib/nginx /etc/nginx /usr/share/nginx
root@20c896637ff5:/# cd /etc/nginx
root@20c896637ff5:/etc/nginx# ls
conf.d  fastcgi_params  koi-utf  koi-win  mime.types  modules  nginx.conf
scgi_params  uwsgi_params  win-utf
root@20c896637ff5:/# cd /etc/nginx
root@20c896637ff5:/etc/nginx# ls
conf.d  fastcgi_params  koi-utf  koi-win  mime.types  modules  nginx.conf
scgi_params  uwsgi_params  win-utf
```

启动Nginx容器的同时设置数据卷的命令：

```
docker run
--name my_nginx
-d -p 80:80
-v /data/nginx/conf/nginx.conf:/etc/nginx/nginx.conf
-v /data/nginx/log:/var/log/nginx
-v /data/nginx/html:/usr/share/nginx/html
nginx
```

参数说明：

第一个-v: 挂载nginx的主配置文件，以方便在宿主机上直接修改容器的配置文件  
第二个-v: 挂载容器内nginx的日志，容器运行起来之后，可以直接在宿主机的这个目录中查看nginx日志  
第三个-v: 挂载静态页面目录

## Tomcat部署

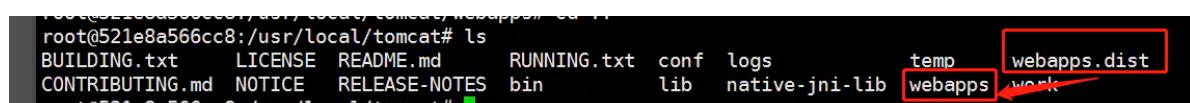
### (1) 下载并运行

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker pull tomcat
Using default tag: latest
latest: Pulling from library/tomcat
6c33745f49b4: Pull complete
ef072fc32a84: Pull complete
c0afb8e68e0b: Pull complete
d599c07d28e6: Pull complete
e8a829023b97: Pull complete
d04be46a31d1: Pull complete
db6007c69c35: Pull complete
e4ad4c894bce: Pull complete
248895fda357: Pull complete
277059b4cba2: Pull complete
Digest: sha256:57dae7dfb9b62a413cde65334c8a18893795cac70afc3be589c8336d8244655d
Status: Downloaded newer image for tomcat:latest
docker.io/library/tomcat:latest
[root@izwz99sm8v95sckz8bd2c4z ~]# docker run -d -p 3335:8080 --name tomcat01
tomcat
7136295a6082cb0f805b025a1471bde02ead4864be3e2c9dc337b1dde0a3113
```

### (2) 进入容器

1.容器中的命令是少了

2.阿里云镜像默认下载的是最小的镜像，保证最小的运行环境。所以我们需要把webapps.dist复制到webapps中



```
root@521e8a566cc8:/usr/local/tomcat# ls
BUILDING.txt      LICENSE      README.md      RUNNING.txt  conf    logs      temp      webapps.dist
CONTRIBUTING.md   NOTICE       RELEASE-NOTES bin          lib     native-jni-lib  work
root@521e8a566cc8:/usr/local/tomcat#
```

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker exec -it tomcat01 /bin/bash
root@7136295a6082:/usr/local/tomcat# ls
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  README.md  RELEASE-NOTES
RUNNING.txt   bin          conf    lib     logs    native-jni-lib  temp    webapps  webapps.dist
work

root@7136295a6082:/usr/local/tomcat# cd webapps.dist

root@7136295a6082:/usr/local/tomcat/webapps.dist# ls
ROOT  docs  examples  host-manager  manager
root@7136295a6082:/usr/local/tomcat/webapps.dist# cd ROOT

root@7136295a6082:/usr/local/tomcat/webapps.dist/ROOT# ls
RELEASE-NOTES.txt  WEB-INF  asf-logo-wide.svg  bg-button.png  bg-middle.png  bg-nav.png  bg-upper.png  favicon.ico  index.jsp  tomcat.css  tomcat.svg
```

```
root@7136295a6082:/usr/local/tomcat/webapps.dist/ROOT# cd ../../
root@7136295a6082:/usr/local/tomcat# cd webapps
root@7136295a6082:/usr/local/tomcat/webapps# ls
root@7136295a6082:/usr/local/tomcat/webapps# cp -r
/usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps/
root@7136295a6082:/usr/local/tomcat/webapps# ls
ROOT  docs  examples  host-manager  manager
root@7136295a6082:/usr/local/tomcat/webapps# exit
exit
```

## ElasticSearch部署

由于ElasticSearch占用内存特别大，所以我们可以修改ElasticSearch占用的内存大小，来避免服务器卡顿

**注意！！！** 可以添加 `-e ES_JAVA_OPTS="-Xms128m -Xmx512m"` 配置ElasticSearch的虚拟机占用的内存大小。

`docker stats` 查看资源占用情况

### 部署elasticsearch

指令：`docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" -e ES_JAVA_OPTS="-Xms128m -Xmx512m" elasticsearch:7.6.2`

```
[root@izwz99sm8v95sckz8bd2c4z ~]# docker run -d --name elasticsearch01 -p
9200:9200 -p 9300:9300 -e "discovery.type=single-node" -e ES_JAVA_OPTS="-Xms128m
-Xmx512m" elasticsearch:7.6.2
3b8cd4991814896c523ee67b84ce198e32bd82b1a62d512b198138a58ca946f1
[root@izwz99sm8v95sckz8bd2c4z ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            NAMES
STATUS              PORTS              NAMES
3b8cd4991814        elasticsearch:7.6.2   "/usr/local/bin/dock..."   10 seconds ago   elasticsearch01
Up 6 seconds        0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp
[root@izwz99sm8v95sckz8bd2c4z ~]# docker stats
```

## MySQL部署

### (1) 下载并运行

```
#拉取并运行容器
docker run -d --name mysql-5.7 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456
mysql:5.7
```

参数说明：

`-p 3306:3306`：映射容器服务的 `3306` 端口到宿主机的 `3306` 端口，外部主机可以直接通过宿主机 `ip:3306` 访问到 MySQL 的服务。  
`MYSQL_ROOT_PASSWORD=123456`：设置 MySQL 服务 默认账号 `root` 用户的密码。

## (2) 进入容器查看MySQL服务

进入容器，并通过账号root查看MySQL服务能否正常连接。

```
docker exec -it 9b3aad6819ff /bin/bash  
mysql -h localhost -u root -p
```

# 容器数据卷

## 什么是容器数据卷？

为了实现数据持久化，使容器之间可以共享数据。可以将容器内的目录，挂载到宿主机上或其他容器内，实现同步和共享的操作。即使将容器删除，挂载到本地的数据卷也不会丢失。

通俗来讲，新建了一个包含mysql的容器，这时候数据都在容器里面，我们为了保证安全性，可以把包含数据的文件夹与外部的文件夹进行数据同步。

容器数据卷技术就是一个数据共享的技术（这个技术中的数据会占用两倍的存储）

## 使用容器数据卷

### 直接使用命令

```
docker run -it -v 主机内目录:容器内目录 镜像名/id
```

将容器内目录挂载到主机内目录上，通过**docker inspect [容器名或ID]**命令查看该容器即可以看到挂载信息：

```
"Mounts": [      挂载 -v 卷  
 {  
   "Type": "bind",  
   "Source": "/home/ceshi", 主机内地址  
   "Destination": "/home", docker容器内的地址  
   "Mode": "",  
   "RW": true,  
   "Propagation": "rprivate"  
 }  
,
```

相当于他们两个文件夹实现了数据同步

### 匿名挂载

```
docker run -d -v 容器内目录 镜像名/id # 匿名挂载
```

匿名挂载后，使用**docker volume ls**命令查看所有挂载的卷：

```

Run 'docker volume COMMAND --help' for more information on a command.
[root@kuangshen home]# docker volume ls
DRIVER      VOLUME NAME
local      9f38292179faa178afcce54d80be99d4ddd68c91d2a68870bcece72d2b7ed061
local      228cf7c48b552e12a10326fedaf5f6b4596754288f54125e0eb6fd000e5b91a16
local      892f3ed0f3d54e7f7e5ae4a4aa975feb0fb3b922198092f1c25627549ef721e6
local      1140d3097e959a9936defb84d100348c935bec93ac93c386ed45bc31119b8711
local      63123e1982129d2928c9f492f42d5a336cdc1bbc0e468291009447191be2cee2
local      a5cae94fefdf69b9dbeb16681097cf3f13ce47f9cf9e8d4f8bbfe8763219816ac
local      dfd259a7777e8ae34fa2fcc8a9fc09ab7a954c9c31b8ce0022b28b2b35f18ef7
local      e39e21f7669a5c18bfc342c6e3c923b258656dac236d620bf7790dbe7826db1a
local      e65ef52004e4ab88659773b5beb3ec531417695a425f737b5d6f9fc8227a2d8b
local      eb1802549897a087bd4013aa458dec078d35f82221197dbdbf945cc51f77d49a
local      f3b8ad951d92ff1aeeb783748e2270547210130c56a9172ce61601342c745c5f
local      f66703b6089c2570d74113aca84cb774b51beb6e0749442d94fe29499f853162

```

每一个VOLUME NAME对应一个挂载的卷，由于挂载时未指定主机目录，因此无法直接找到目录。

由于没有写容器外部的目录，所以默认地址存在于`/var/lib/docker/volumes/卷名/_data`，可以通过`docker inspect`找到

## 具名挂载

```
docker run -d -v 卷名: 容器内目录 镜像名/id # 具名挂载
```

```

[root@kuangshen home]# docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx nginx
95b809564484c8ac87d65c69643e7e67447f1c77ff9a91b93edec7003692e3a9
[root@kuangshen home]# docker volume ls
DRIVER      VOLUME NAME
local      9f38292179faa178afcce54d80be99d4ddd68c91d2a68870bcece72d2b7ed061
local      228cf7c48b552e12a10326fedaf5f6b4596754288f54125e0eb6fd000e5b91a16
local      892f3ed0f3d54e7f7e5ae4a4aa975feb0fb3b922198092f1c25627549ef721e6
local      1140d3097e959a9936defb84d100348c935bec93ac93c386ed45bc31119b8711
local      63123e1982129d2928c9f492f42d5a336cdc1bbc0e468291009447191be2cee2
local      a5cae94fefdf69b9dbeb16681097cf3f13ce47f9cf9e8d4f8bbfe8763219816ac
local      dfd259a7777e8ae34fa2fcc8a9fc09ab7a954c9c31b8ce0022b28b2b35f18ef7
local      e39e21f7669a5c18bfc342c6e3c923b258656dac236d620bf7790dbe7826db1a
local      e65ef52004e4ab88659773b5beb3ec531417695a425f737b5d6f9fc8227a2d8b
local      eb1802549897a087bd4013aa458dec078d35f82221197dbdbf945cc51f77d49a
local      f3b8ad951d92ff1aeeb783748e2270547210130c56a9172ce61601342c745c5f
local      f66703b6089c2570d74113aca84cb774b51beb6e0749442d94fe29499f853162
local      juming-nginx
local      nginxconfig
[root@kuangshen home]#

```

可以发现挂载的卷：volume01，并通过`docker volume inspect 卷名`命令找到主机内目录：

```

[root@kuangshen home]# docker volume inspect juming-nginx
[
  {
    "CreatedAt": "2020-05-15T19:37:31+08:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/juming-nginx/_data",
    "Name": "juming-nginx",
    "Options": null,
    "Scope": "local"
  }
]

```

所有docker容器内的卷，在未指定主机内目录时，都在：`/var/lib/docker/volumes/卷名/_data`下，可通过具名挂载可以方便的找到卷，因此广泛使用这种方式进行挂载。

```

#通过-v 容器内路径 ,  ro rw 改变读写权限
#ro readonly 只读
#rw readwrite 可读可写

docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx:ro nginx
docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx:rw nginx
#ro 只要看到ro就说明这个路径只能通过宿主机来操作, 容器内部是无法操作的

```

## 数据卷容器

很像java中子类继承父类



```

[root@kuangshen /]# docker images
REPOSITORY      TAG        IMAGE ID      CREATED       SIZE
kuangshen/centos  1.0        5d04f189a434  24 minutes ago  237MB
mysql            5.7        e73346bdf465  33 hours ago   448MB
nginx            latest     602e111c06b6  3 weeks ago    127MB
centos           latest     470671670cac  3 months ago   237MB
[root@kuangshen /]# docker run -it --name docker01 kuangshen/centos:1.0
[root@45768e0d0196 /]# ls
bin  etc  lib  lost+found  mnt  proc  run  srv  tmp  var          volume02
dev  home  lib64  media      opt  root  sbin  sys  usr  volume01
[root@45768e0d0196 /]# ls -l
total 56
lrwxrwxrwx  1 root root    7 May 11  2019 bin -> usr/bin
drwxr-xr-x  5 root root  360 May 15 12:18 dev
drwxr-xr-x  1 root root  4096 May 15 12:18 etc
drwxr-xr-x  2 root root  4096 May 11  2019 home
lrwxrwxrwx  1 root root    7 May 11  2019 lib -> usr/lib
lrwxrwxrwx  1 root root    9 May 11  2019 lib64 -> usr/lib64
drwxr-xr-x  2 root root  4096 Jan 13 21:48 lost+found
drwxr-xr-x  2 root root  4096 May 11  2019 media
drwxr-xr-x  2 root root  4096 May 11  2019 mnt
drwxr-xr-x  2 root root  4096 May 11  2019 opt
dr-xr-xr-x  115 root root   0 May 15 12:18 proc
dr-xr-xr-x  2 root root  4096 Jan 13 21:49 root
drwxr-xr-x  11 root root  4096 Jan 13 21:49 run
lrwxrwxrwx  1 root root   8 May 11  2019 sbin -> usr/sbin
drwxr-xr-x  2 root root  4096 May 11  2019 srv
dr-xr-xr-x  13 root root   0 Mar 23 14:00 sys
drwxrwxrwt  7 root root  4096 Jan 13 21:49 tmp
drwxr-xr-x  12 root root  4096 Jan 13 21:49 usr
drwxr-xr-x  20 root root  4096 Jan 13 21:49 var
drwxr-xr-x  2 root root  4096 May 15 12:18 volume01
drwxr-xr-x  2 root root  4096 May 15 12:18 volume02

```

启动docker01

挂载到外部的两个文件

```
[root@kuangshen /]# docker run -it --name docker02 --volumes-from docker01 kuangshen/centos:1.0
[root@af993368f540 /]# ls -l
total 56
lrwxrwxrwx  1 root root    7 May 11  2019 bin -> usr/bin
drwxr-xr-x  5 root root  360 May 15 12:19 dev
drwxr-xr-x  1 root root 4096 May 15 12:19 etc
drwxr-xr-x  2 root root 4096 May 11  2019 home
lrwxrwxrwx  1 root root    7 May 11  2019 lib -> usr/lib
lrwxrwxrwx  1 root root    9 May 11  2019 lib64 -> usr/lib64
drwx----- 2 root root 4096 Jan 13 21:48 lost+found
drwxr-xr-x  2 root root 4096 May 11  2019 media
drwxr-xr-x  2 root root 4096 May 11  2019 mnt
drwxr-xr-x  2 root root 4096 May 11  2019 opt
dr-xr-xr-x 116 root root    0 May 15 12:19 proc
dr-xr-x---  2 root root 4096 Jan 13 21:49 root
drwxr-xr-x  11 root root 4096 Jan 13 21:49 run
lrwxrwxrwx  1 root root    8 May 11  2019 sbin -> usr/sbin
drwxr-xr-x  2 root root 4096 May 11  2019 srv
dr-xr-xr-x 13 root root    0 Mar 23 14:00 sys
drwxrwxrwt  7 root root 4096 Jan 13 21:49 tmp
drwxr-xr-x 12 root root 4096 Jan 13 21:49 usr
drwxr-xr-x 20 root root 4096 Jan 13 21:49 var
drwxr-xr-x  2 root root 4096 May 15 12:18 volume01
drwxr-xr-x  2 root root 4096 May 15 12:18 volume02
[root@af993368f540 /]#
```

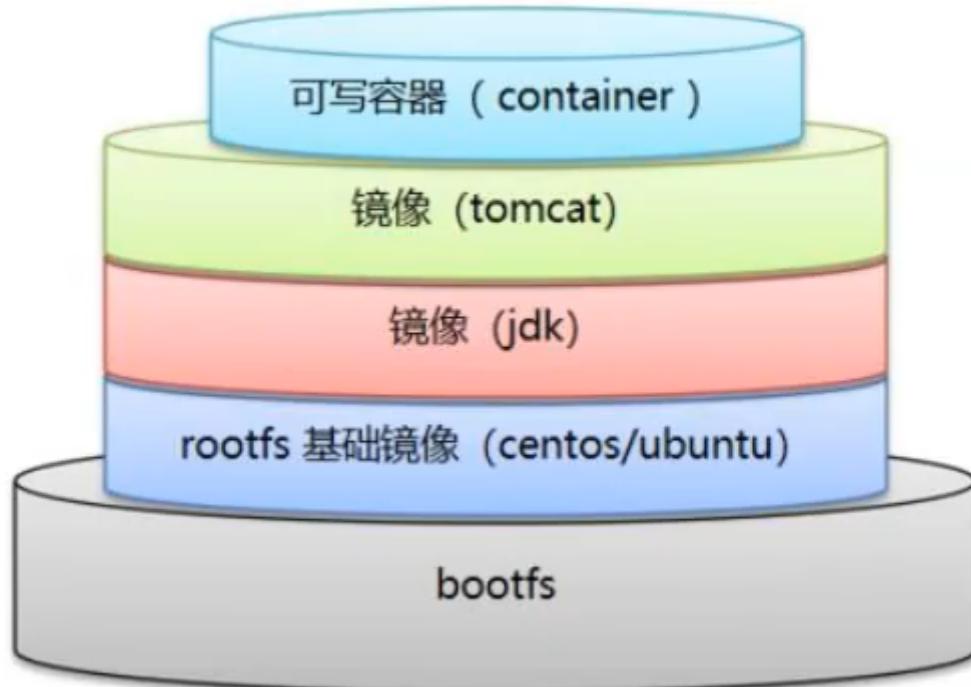
启动docker02去继承  
docker01

这里面docker01就是数据卷容器，如果docker01被删除，docker02中的数据不会被清除（因为是拷贝而不是访问）

```
docker run -it --name container02 --volumes-from container01 镜像名/id # 将两个容器进行挂载
```

## DockerFile

Dockerfile是用来构建**docker镜像**的文件，进一步解释了docker的精髓--分层



每一行指令都会创建提交一个新的镜像层，并提交

## 构建步骤：

编写一个dockerfile文件,随后运行命令:

```
docker build -f 文件路径 -t 镜像名 . # 文件名为Dockerfile时可省略且最后的.不要忽略
```

举例:

```
[root@iz1608aqb7ntn9z 20210806]# vim Dockerfile
# -----写入内容-----
FROM centos      # 来自centos
CMD /bin/bash    # 进入到/bin/bash
CMD echo Hello Dockerfile  # 输出Hello Dockerfile

# -----写入结束-----
[root@iz1608aqb7ntn9z 20210806]# docker build -f ./Dockerfile -t mydocker .
Sending build context to Docker daemon  2.56kB
Step 1/3 : FROM centos
--> 300e315adb2f
Step 2/3 : CMD /bin/bash
--> Running in 526f489adf0b
Removing intermediate container 526f489adf0b
--> 3c2af9c73098
Step 3/3 : CMD echo Hello Dockerfile
--> Running in 023af54a93e2
Removing intermediate container 023af54a93e2
--> 7753b44c9137
Successfully built 7753b44c9137
Successfully tagged mydocker:latest

[root@iz1608aqb7ntn9z 20210806]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
mydocker            latest   7753b44c9137  6 seconds ago  209MB
.....
[root@iz1608aqb7ntn9z 20210806]# docker run -it mydocker
Hello Dockerfile
```

## Dockerfile命令

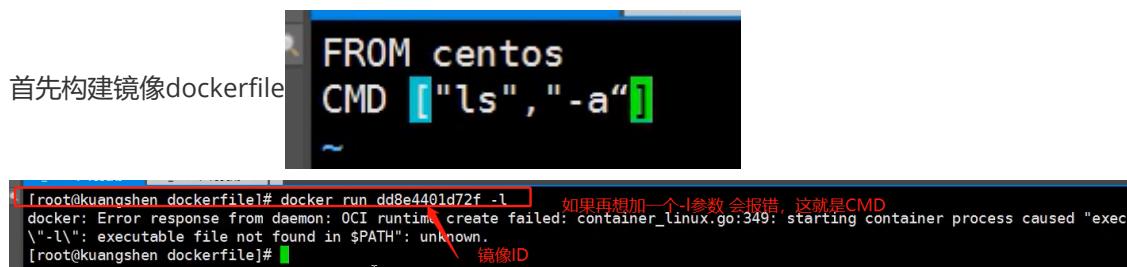
命令	效果
FROM	基础镜像: Centos/Ubuntu
MAINTAINER	镜像作者+邮箱
RUN	镜像构建的时候需要运行的命令
ADD	为镜像添加内容 (压缩包)
WORKDIR	镜像工作目录 (进入容器时的目录)
VOLUME	挂载的目录
EXPOSE	暴露端口配置
CMD/ENTRYPOINT	指定这个容器启动时要运行的命令 (CMD替代先前命令, ENTRYPOINT在先前命令后追加)
COPY	类似于ADD, 将文件拷贝到镜像中
ENV	构建时设置环境变量

## 注意

- 每个保留关键字 (指令) 都必须是大写字母
- 从上到下顺序执行
- “#” 表示注释
- 每一个指令都会创建提交一个新的镜像层并提交
- 列出本地镜像的历史 (当前这个镜像是如何被做出来的) `docker history 镜像ID`

```
[root@VM-4-3-centos ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
centos latest 5d0da3dc9764 10 months ago 231MB
[root@VM-4-3-centos ~]# docker history 5d0da3dc9764
IMAGE CREATED CREATED BY SIZE COMMENT
5d0da3dc9764 10 months ago /bin/sh -c #(nop) CMD ["/bin/bash"]
<missing> 10 months ago /bin/sh -c #(nop) LABEL org.label-schema.sc...
<missing> 10 months ago /bin/sh -c #(nop) ADD file:805cb5e15fb6e0bb0...
```

- 对于CMD和ENTRYPOINT的区别



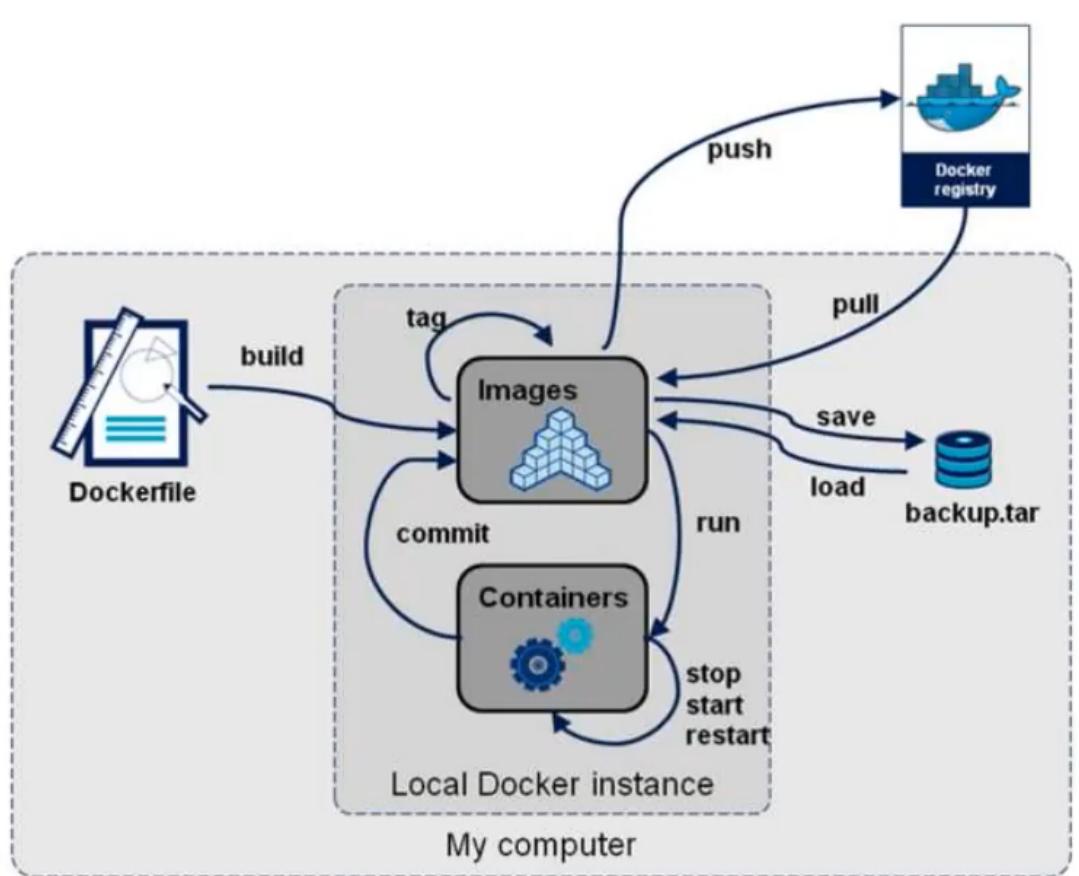
CMD的情况下 -l 替换里面的CMD命令 -l不是完整的命令，所以报错 如果改变成

```
[root@kuangshen dockerfile]# docker run dd8e4401d72f ls -al
total 56
drwxr-xr-x  1 root root 4096 May 15 13:13 .
drwxr-xr-x  1 root root 4096 May 15 13:13 ..
-rw xr-xr-x  1 root root    0 May 15 13:13 .dockerenv
lrwxrwxrwx  1 root root    7 May 11 2019 bin -> /usr/bin
drwxr-xr-x  5 root root 340 May 15 13:13 dev
drwxr-xr-x  1 root root 4096 May 15 13:13 etc
drwxr-xr-x  2 root root 4096 May 11 2019 home
lrwxrwxrwx  1 root root    7 May 11 2019 lib -> /usr/lib
lrwxrwxrwx  1 root root    9 May 11 2019 lib64 -> /usr/lib64
drwx----- 2 root root 4096 Jan 13 21:48 lost+found
drwxr-xr-x  2 root root 4096 May 11 2019 media
drwxr-xr-x  2 root root 4096 May 11 2019 mnt
drwxr-xr-x  2 root root 4096 May 11 2019 opt
dr-xr-xr-x 118 root root    0 May 15 13:13 proc
dr-xr-x---  2 root root 4096 Jan 13 21:49 root
drwxr-xr-x 11 root root 4096 Jan 13 21:49 run
lrwxrwxrwx  1 root root    8 May 11 2019 sbin -> /usr/sbin
drwxr-xr-x  2 root root 4096 May 11 2019 srv
dr-xr-xr-x 13 root root    0 Mar 23 14:00 sys
drwxrwxrwt  7 root root 4096 Jan 13 21:49 tmp
drwxr-xr-x 12 root root 4096 Jan 13 21:49 usr
drwxr-xr-x 20 root root 4096 Jan 13 21:49 var
```

则会顺利执行ls -al指令，里面的CMD命令被替换

ENTRYPOINT可以直接在后面加-l，在后面追加

小结



## Docker网络

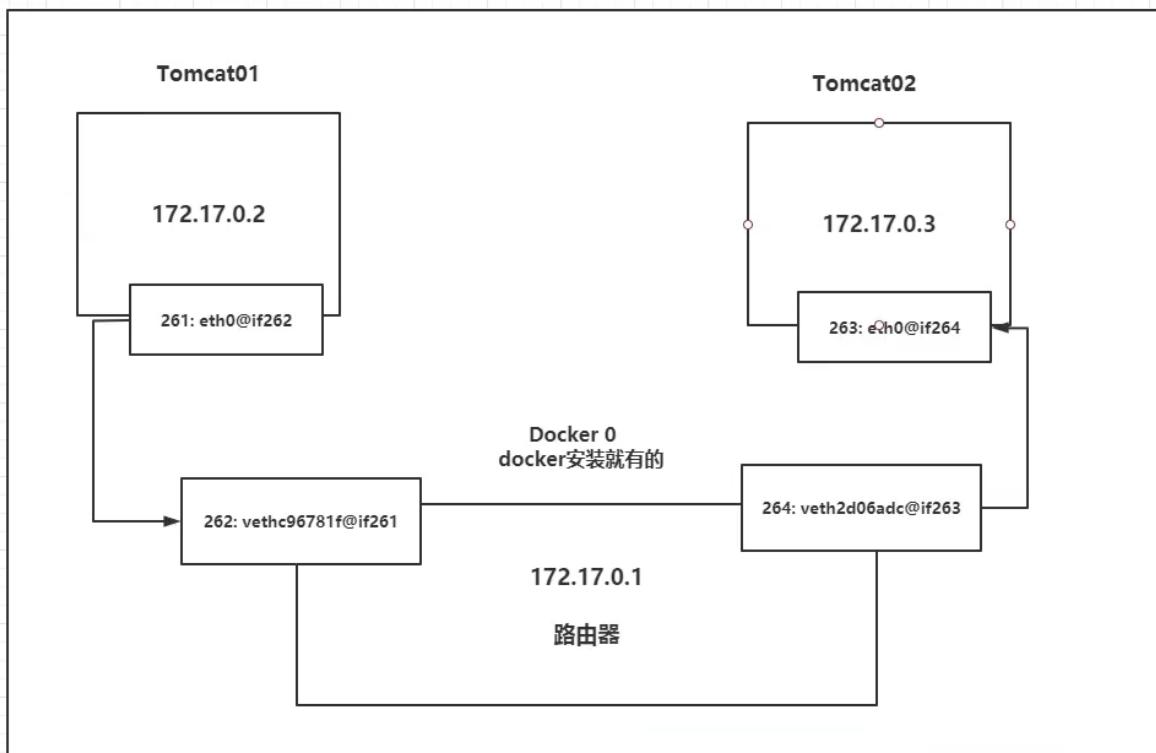
# 理解Docker

通过命令`ip addr`查看本地ip地址，我们发现除了本机回环地址和阿里云的内网地址外，还多了一个网卡：Docker0，这是Docker服务启动后自动生成的。

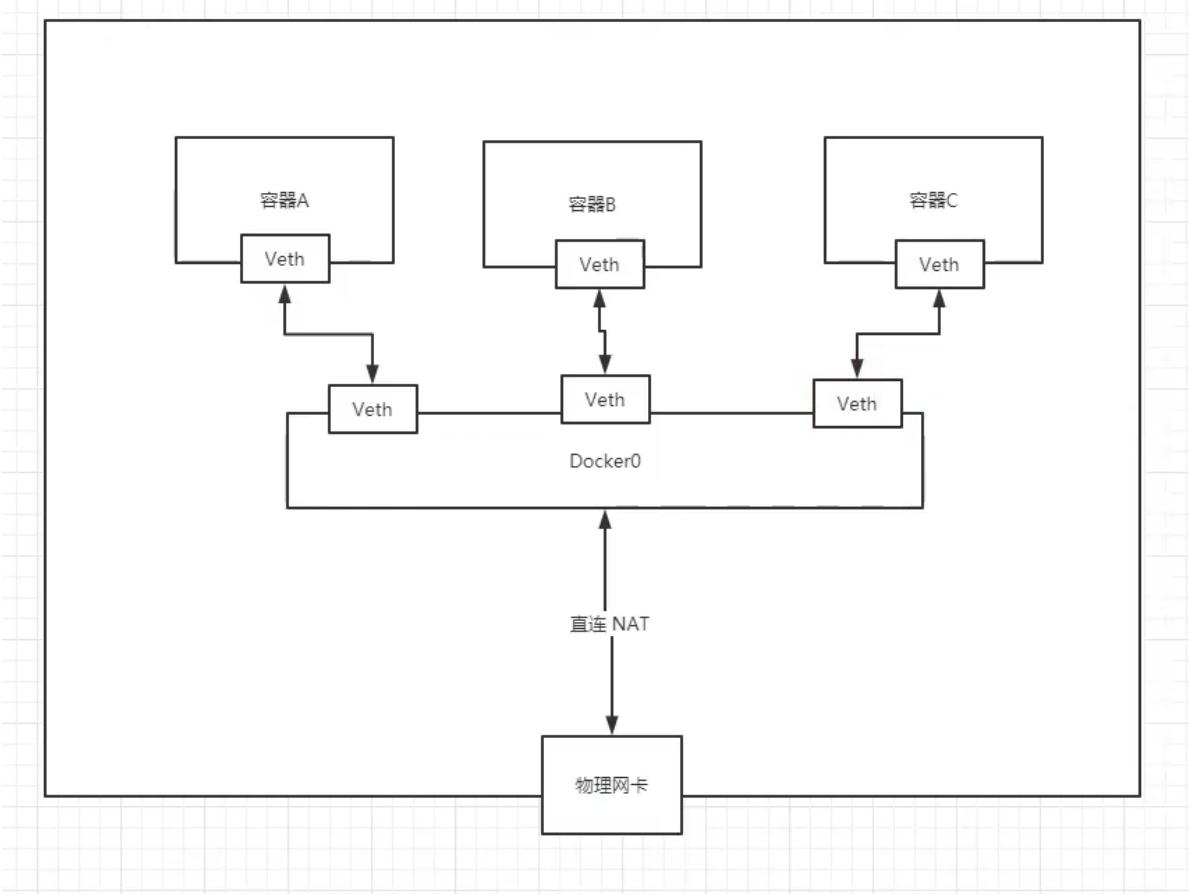
```
[root@kuangshen tomcat]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:16:3e:30:27:f4 brd ff:ff:ff:ff:ff:ff
        inet 172.17.90.138/20 brd 172.17.95.255 scope global dynamic eth0
            valid_lft 310744886sec preferred_lft 310744886sec
                阿里云内网地址
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:bb:71:07:06 brd ff:ff:ff:ff:ff:ff
        inet 172.18.0.1/16 brd 172.18.255.255 scope global docker0
            valid_lft forever preferred_lft forever
                docker
```

而如果进入一个正在后台运行的tomcat容器，同样使用`ip addr`命令，发现容器得到了一个新的网络，ip地址：`172.18.0.2`。这是Docker在容器启动时为其分配的。

```
[root@kuangshen /]# docker exec -it tomcat01 ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
261: eth0@if262: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
            valid_lft forever preferred_lft forever
[root@kuangshen /]#
```



- linux可以ping通docker容器内部，因为docker0的ip地址为**172.17.0.1**，容器为**172.17.0.2**。
- 原理：我们每启动一个docker容器，docker就会给容器分配一个默认的可用ip，我们只要安装了docker，就会有一个网卡docker0(bridge)。网卡采用桥接模式，并使用veth-pair技术（veth-pair就是一堆虚拟设备接口，成对出现，一段连着协议，一段彼此相连，充当一个桥梁。）。
- 容器和容器之间是可以互相ping通的：容器1→Docker0→容器2
- docker中的所有网络接口都是虚拟的，转发效率高。删除容器后，对应的网桥也随之删除。



## --link

若编写一个微服务并连接数据库，如果数据库ip改变，如何根据容器名而不是ip访问容器？显然，直接使用容器名是无法ping通容器内部的：

```
tomcat02
[root@kuangshen /]# docker exec -it tomcat02 ping tomcat01
ping: tomcat01: Name or service not known
```

这时我们可以在容器启动命令中加入一个选项：`-link`，使得我们可以根据容器名来访问容器。

```
docker run -d -P --link 容器名/id 镜像名/id
```

```
[root@kuangshen /]# docker run -d -P --name tomcat03 --link tomcat02 tomcat
5ca72d80ebb048d3560df1400af03130f37ece244be2a54884336aace2106884
[root@kuangshen /]# docker exec -it tomcat03 ping tomcat02
PING tomcat02 (172.18.0.3) 56(84) bytes of data.
64 bytes from tomcat02 (172.18.0.3): icmp_seq=1 ttl=64 time=0.100 ms
```

然而反向就不可以ping通，这是因为`-link`的本质是把需要连接的容器名/id写入启动容器的配置文件中，即增加了一个ip和容器名/id的映射：

```
tomcat      latest      d0331211/bb0      2 da
[root@kuangshen /]# docker exec -it tomcat03 cat /etc/hosts
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      tomcat02 312857784cd4
172.18.0.4      5ca72d80ebb0
[root@kuangshen /]#
```

```
[root@kuangshen /]# docker exec -it tomcat02 ping tomcat03
ping: tomcat03: Name or service not known
[root@kuangshen /]#
```

目前已经不建议使用这种方式。

## 自定义网络

我们使用命令：

```
docker network ls      # 查看所有的docker网络
```

```
[root@kuangshen /]# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
5a008c015cac    bridge    bridge      local
db44649a9bff    compositest_default  bridge      local
ae2b6209c2ab    host      host      local
c037f7ec7e57    none      null      local
b701476a0394    redis    bridge      local
[root@kuangshen /]#
```

docker中的网络模式有：

- bridge: 桥接 (docker默认) /
- none: 不配置网络 /
- host: 和宿主机共享网络

**docker run** 命令默认带有一个参数`-net bridge`, 此处的bridge指的就是docker0。如果我们不想使用 docker0, 那如何创建一个新的网络呢?

```
docker network create --driver 网络模式 --subnet 子网ip --gateway 网关 网络名
```

```
[root@kuangshen /]# docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 mynet
eb21272b3a35ceaba11b4aa5bbff131c3fb09c4790f0852ed4540707438db052
```

```
[root@kuangshen /]# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
5a008c015cac    bridge    bridge      local
db44649a9bff    compositest_default  bridge      local
ae2b6209c2ab    host      host      local
eb21272b3a35    mynet    bridge      local
c037f7ec7e57    none      null      local
b701476a0394    redis    bridge      local
```

用户自定义的网卡可以在容器之间提供自动的 DNS 解析，缺省的桥接网络上的容器只能通过 IP 地址互相访问，除非使用 --link 参数。在用户自定义的网卡上，容器直接可以通过名称或者别名相互解析

相当于搭了一个局域网，docker默认在用户自定义的网卡上，容器直接可以通过名称或者别名进行自动的DNS相互解析

## 网络连通

```
[root@kuangshen /]# docker network --help

Usage: docker network COMMAND

Manage networks

Commands:
  connect      Connect a container to a network
  create       Create a network
  disconnect   Disconnect a container from a network
  inspect      Display detailed information on one or more networks
  ls           List networks
  prune        Remove all unused networks
  rm           Remove one or more networks
```

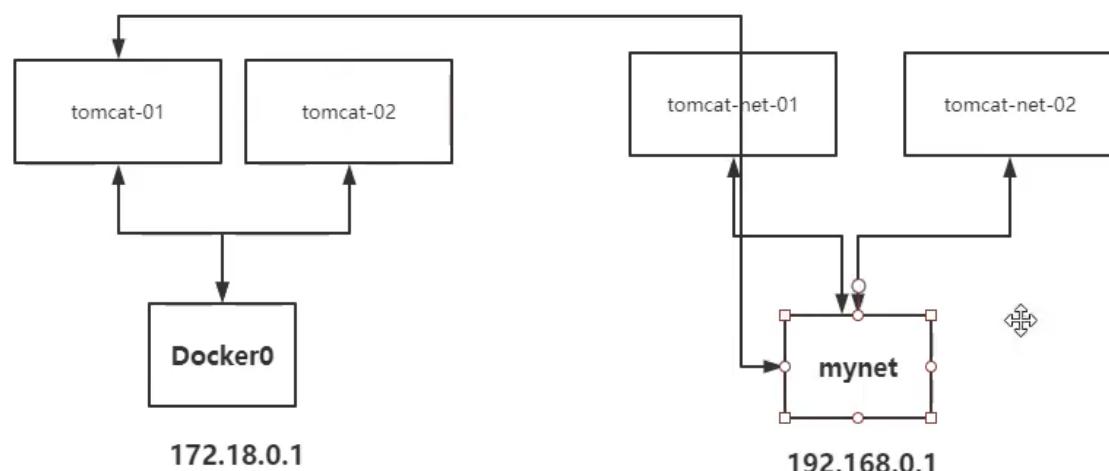
语法：

```
[root@kuangshen /]# docker network connect --help

Usage: docker network connect [OPTIONS] NETWORK CONTAINER
      Connect a container to a network

Options:
  --alias strings      Add network-scoped alias for the container
  --driver-opt strings  driver options for the network
  --ip string          IPv4 address (e.g., 172.30.100.104)
  --ip6 string         IPv6 address (e.g., 2001:db8::33)
  --link list          Add link to another container
  --link-local-ip strings Add a link-local address for the container
[root@kuangshen /]#
```

把不同网段的容器进行网络互联



对于建立在不同网络下(docker0, newnet)的两个容器tomcat01和tomcat02，他们的网段不同，因此是无法彼此ping通容器内部的：

这时我们需要通过**docker network connect**命令打通容器与网络之间的连接：

```
docker network connect 网络名 容器名/id
```

```
[root@kuangshen /]# docker network connect mynet tomcat01
```

## SpringBoot项目打包Docker镜像

1、构建SpringBoot项目  demo-0.0.1-SNAPSHOT.jar

2、打包运行，上传到服务器

```
[root@kuangshen idea]# ll
total 17228
-rw-r--r-- 1 root root 17634295 May 15 23:58 demo-0.0.1-SNAPSHOT.j
-rw-r--r-- 1 root root      120 May 15 23:58 Dockerfile
```

3、编写Dockerfile

```
FROM java:8
COPY *.jar /app.jar
CMD ["--server.port=8080"]
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

4、构建镜像

```
# 1. 复制jar和DockerFile到服务器
# 2. 构建镜像
$ docker build -t xxxxx:xx .
```

5、发布运行

以后我们使用了Docker之后，给别人交付就是一个镜像即可！

## Docker Compose

Compose 是用于定义和运行多容器 Docker 应用程序的工具。通过 Compose，您可以使用 YML 文件来配置应用程序需要的所有服务。然后，使用一个命令，就可以从 YML 文件配置中创建并启动所有服务。

### Compose 使用的三个步骤：

- 使用 Dockerfile 定义应用程序的环境。
- 使用 docker-compose.yml 定义构成应用程序的服务，这样它们可以在隔离环境中一起运行。
- 最后，执行 docker-compose up 命令来启动并运行整个应用程序。

## Compose 安装

第一步：

运行以下命令以下载 Docker Compose 的当前稳定版本：

```
sudo curl -L "https://github.com/docker/compose/releases/download/v2.2.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Docker Compose 存放在 GitHub，不太稳定。

你可以也通过执行下面的命令，高速安装 Docker Compose。(如果要下载其他版本，可以改/v2.4.1这部分)

```
curl -L https://get.daocloud.io/docker/compose/releases/download/v2.4.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

## 第二步：

将可执行权限应用于二进制文件：

```
sudo chmod +x /usr/local/bin/docker-compose
```

## 第三步：

创建软链：

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

之后就安装好啦

可以用 docker-compose --version 命令测试安装是否成功

```
[root@VM-4-3-centos ~]# docker-compose --version
docker-compose version 1.28.4, build cabd5cfb
```