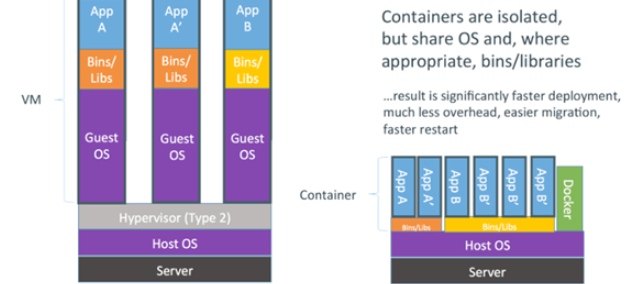
# Docker

官网是这样介绍docker的：

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications…

其实看完这句话还是不明白docker究竟是什么

我们可以把他想象成是一个用了一种新颖方式实现的超轻量虚拟机。当然在实现的原理和应用上还是和VM有巨大差别的，并且专业的叫法是应用容器（Application Container）。



比如现在想用MySQL,那就找个装好并配置好的MySQL的容器(可以认为是特殊的,轻量级的虚拟机),运行起来,那么就可以使用 MySQL了。

那么为什么不直接在操作系统中安装一个mysql,而是用容器呢?

安装MySql过程并不简单,要配置安装源,安装依赖包,对mysql进行配置…如果要在多台主机上安装,每台主机都要进行这些繁琐的操作,万一服务器挂了,这一系列操作还要再重来一遍

但有了docker,一个安装配置好的mysql容器,可以直接拿到另一台主机上启动,而不必重新安装mysql

另外,docker还有一重要的用处,就是可以保证开发,测试和生产环境的一致.

# docker 手册

中文免费手册 [Docker — 从入门到实践]  
[https://vuepress.mirror.docker-practice.com](https://vuepress.mirror.docker-practice.com/)

docker 从入门到实践，离线版

docker pull dockerpracticecn/docker\_practice

docker run -it --rm -p 4000:80 dockerpracticecn/docker\_practice

# centos7 安装docker

## 离线安装包

### 下载离线安装包

<https://download.csdn.net/download/weixin_38305440/12265961>

### 上传离线安装包

docker-install 目录上传到 /root

### 切换到docker-install目录

**cd docker-install**

### 安装

**rpm -ivh \*.rpm**

## yum 在线安装 docker

### 官方安装手册

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

### 卸载旧版

**sudo yum remove docker-ce \**

**docker-ce-client \**

**docker-client-latest \**

**docker-common \**

**docker-latest \**

**docker-latest-logrotate \**

**docker-logrotate \**

**docker-engine**

### ****安装一组工具****

**sudo yum install -y yum-utils \**

**device-mapper-persistent-data \**

**lvm2**

### ****设置yum仓库地址****

**sudo yum-config-manager \**

**--add-repo \**

**https://download.docker.com/linux/centos/docker-ce.repo**

**sudo yum-config-manager \**

**--add-repo \**

**http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo**

### ****更新yum缓存****

**sudo yum makecache fast**

### ****安装新版 docker****

**sudo yum install -y docker-ce docker-ce-cli containerd.io**

## 启动docker系统服务

### ****启动 docker****

**sudo systemctl start docker**

### ****设置 docker 开机启动****

**sudo systemctl enable docker**

## 镜像加速

由于国内网络问题，需要配置加速器来加速。

修改配置文件 /etc/docker/daemon.json

vim /etc/docker/daemon.json

**添加以下内容:**

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

**之后重新启动服务。**

**# 重新加载docker配置**

**sudo systemctl daemon-reload**

**#重启docker服务**

**sudo systemctl restart docker**

### ****查看镜像配置****

**docker info**

### ****运行 hello-world 镜像，验证 docker****

sudo docker run hello-world

# 基本概念

## 镜像

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由一组文件系统组成，或者说，由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

## 容器

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的 类 和 实例 一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的 命名空间。因此容器可以拥有自己的 root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户 ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。

前面讲过镜像使用的是分层存储，容器也是如此。每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层，我们可以称这个为容器运行时读写而准备的存储层为容器存储层。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照 Docker 最佳实践的要求，容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用 数据卷（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。

数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此，使用数据卷后，容器删除或者重新运行之后，数据却不会丢失。

# docker 镜像操作

## 下载 CentOS 镜像

docker pull centos:7

## 查看centos7镜

docker images

或

docker image ls

## 运行 centos7

docker run -it xxxx bash

xxxx - 镜像名, 或 image id 的前几位

-it 这是两个参数，一个是 -i：交互式操作，一个是 -t 终端。我们这里打算进入 bash 执行一些命令并查看返回结果，因此我们需要交互式终端。

bash 放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 bash。

## 删除镜像

501 镜像 id 前几位，一般三位以上，足够区分即可

docker image rm 501

删除指定仓库的镜像

docker image rm centos

## 镜像导出

docker save mysql:5.7 node:8 | gzip > app.tar.gz

## 镜像导入

docker load < apps.tar.gz

docker load -i apps.tar.gz

# 容器操作

## 启动容器

docker run -it centos:7 bash

当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

检查本地是否存在指定的镜像，不存在就从公有仓库下载

利用镜像创建并启动一个容器

分配一个文件系统，并在只读的镜像层外面挂载一层可读写层

从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去

从地址池配置一个 ip 地址给容器

执行用户指定的应用程序

执行完毕后容器被终止

## 后台运行

docker run -dit centos:7

-d 后台运行容器

容器是否会长久运行，是和 docker run 指定的命令有关，和 -d 参数无关。

## 查看后台运行的容器输出结果

docker container logs 802

## 查看容器

docker container ls -a

# 或

docker ps –a

-a all, 全部

## 终止容器

docker container stop 802

## 重新启动容器

docker container restart 802

## 进入容器

在使用 -d 参数时，容器启动后会进入后台。

某些时候需要进入容器进行操作，可以使用 docker exec 命令

docker exec -it 802 bash

## 删除容器

docker container rm 802

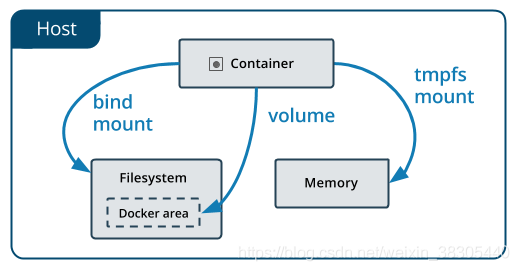
如果删除运行中的容器，需要添加 -f 参数

## 清理所有终止状态容器

注意:这里的container不能省略

docker container prune

# 数据管理



在容器中管理数据主要有两种方式：

数据卷（Volumes）

挂载主机目录 (Bind mounts)

## 数据卷

数据卷 是一个可供一个或多个容器使用的特殊目录

数据卷 可以在容器之间共享和重用

对 数据卷 的修改会立马生效

对 数据卷 的更新，不会影响镜像

数据卷 默认会一直存在，即使容器被删除

### 创建数据卷

docker volume create my-vol

### 查看所有数据卷

docker volume ls

### 查看指定 数据卷 的信息

docker volume inspect my-vol

查询的结果：

[

{

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/my-vol/\_data",

"Name": "my-vol",

"Options": {},

"Scope": "local"

}

]

### 启动挂载数据卷的容器

docker run -it --mount source=my-vol,target=/webapp centos:7 bash

# 或者：

docker run -it -v my-vol:/webapp centos:7 bash

-v my-vol:/webapp 把数据卷 my-vol 挂载到容器的 /webapp 目录

### 删除数据卷

删除指定的数据卷，如果数据卷被容器使用则无法删除

docker volume rm my-vol

清理无主数据卷

docker volume prune

## 挂载主机目录

docker run -it --mount type=bind,source=/usr/app,target=/opt/app centos:7 bash

# 或

docker run -it -v /usr/app:/opt/app centos:7 bash

-v 如果本地目录不存在 Docker 会自动为你创建一个文件夹

--mount 参数时如果本地目录不存在，Docker 会报错

### 查看挂载目录信息

docker inspect 91a

显示结果：

...

"Mounts": [

{

"Type": "bind",

"Source": "/usr/app",

"Destination": "/opt/app",

"Mode": "",

"RW": true,

"Propagation": "rprivate"

}

],

...

# 网络

## 自动分配映射端口

docker run -d -P tomcat

# 查看容器信息

docker container ls -a

显示结果：

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

d03480b1a781 tomcat "catalina.sh run" 3 minutes ago Up 3 minutes 0.0.0.0:32768->8080/tcp trusting\_gagarin

## 映射指定端口

docker run -d -p 8080:8080 tomcat

8080:8080 本机端口:容器端口

## 映射多个端口

docker run -d \

-p 5000:5000 \

-p 80:8080 tomcat

## 映射指定端口（指定网卡）

docker run -d -p 192.168.64.150:8080:8080 tomcat

## 自动分配映射端口（指定网卡）

docker run -d -p 192.168.64.150::8080 tomcat

## 查看端口配置

docker port 8af

# 容器互联

## 新建网络

docker network create -d bridge my-net

-d driver,网络类型，默认 bridge，也可以是 overlay（Swarm mode）

## 列出网络

docker network ls

## 查看网络信息

docker inspect 67d

## 连接容器

docker run -it --name app1 --network my-net centos:7

新开终端执行：

docker run -it --name app2 --network my-net centos:7

在两个终端中分别执行：

ping app1

ping app2

显示如下：

[root@35569c623c4c /]# ping app1

PING app1 (172.18.0.2) 56(84) bytes of data.

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=1 ttl=64 time=0.577 ms

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=2 ttl=64 time=0.061 ms

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=3 ttl=64 time=0.066 ms

......

多个容器互联，推荐使用 Docker Compose

# Dockerfile

## 准备

centos:7镜像

jdk压缩包 jdk-8u212-linux-x64.tar.gz

tomcat7压缩包 apache-tomcat-7.0.96.tar.gz

## Dockerfile文件

#以centos7为基础,安装oracle jdk8和tomcat7

FROM centos:7

#ADD命令将压缩包传入镜像中的指定目录,并同时解压缩

ADD jdk-8u212-linux-x64.tar.gz /opt/

ADD apache-tomcat-7.0.96.tar.gz /usr/

#为了方便,把文件夹名称改得简单一点

RUN mv /usr/apache-tomcat-7.0.96 /usr/tomcat

#设置环境变量

ENV JAVA\_HOME=/opt/jdk1.8.0\_212 \

CATALINA\_HOME=/usr/tomcat \

PATH=$PATH:/opt/jdk1.8.0\_212/bin:/usr/tomcat/bin

#暴露容器的8080端口

EXPOSE 8080

#设置启动容器时自动运行tomcat

ENTRYPOINT /usr/tomcat/bin/startup.sh && tail -F /usr/tomcat/logs/catalina.out

FROM centos:7

ADD jdk-8u212-linux-x64.tar.gz /opt/

ADD apache-tomcat-7.0.96.tar.gz /usr/

RUN mv /usr/apache-tomcat-7.0.96 /usr/tomcat

ENV JAVA\_HOME=/opt/jdk1.8.0\_212 \

CATALINA\_HOME=/usr/tomcat \

PATH=$PATH:/opt/jdk1.8.0\_212/bin:/usr/tomcat/bin

EXPOSE 8080

ENTRYPOINT /usr/tomcat/bin/startup.sh && tail -F /usr/tomcat/logs/catalina.out

## 使用 Dockerfile 构建镜像

docker build -t tomcat:7 .

注意末尾的点,表示构建过程中从当前目录寻找文件

### 启动容器

### 准备存储目录

webapps目录,例如 /opt/webapps

logs目录,例如 /var/lib/tomcat-logs

mkdir /opt/webapps

mkdir /opt/webapps/ROOT

mkdir /var/lib/tomcat-logs

# 添加欢迎页

cat <<EOF > /opt/webapps/ROOT/index.html

<h1>Hello Docker!!!</h1>

EOF

### 启动容器,挂载目录

docker run \

-d \

-p 8080:8080 \

--name tomcat7 \

-v /opt/webapps:/usr/tomcat/webapps \

-v /var/lib/tomcat-logs:/usr/tomcat/logs \

tomcat:7

# Docker案例

## 关闭防火墙

# 关闭防火墙

systemctl stop firewalld.service

# 禁止防火墙开机启动

systemctl disable firewalld.service

## 启动或重启docker

# 启动docker

systemctl start docker

# 重启docker

systemctl restart docker

## redis

### 加载镜像

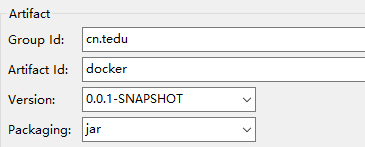
docker load < redis-docker-image.gz

### 启动容器

docker run -d --name redis7000 -p 7000:6379 redis

### jedis 连接测试

#### 新建测试项目



#### pom.xml

添加 redis 和 junit 依赖

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>tedu.cn</groupId>

<artifactId>docker</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>docker</name>

<dependencies>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

</dependencies>

</project>

#### 编写测试类

**package** docker;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.junit.Test;

**import** redis.clients.jedis.Jedis;

**import** redis.clients.jedis.JedisPoolConfig;

**import** redis.clients.jedis.JedisShardInfo;

**import** redis.clients.jedis.ShardedJedis;

**import** redis.clients.jedis.ShardedJedisPool;

**public** **class** Test1 {

@Test

**public** **void** test1() {

Jedis j = **new** Jedis("192.168.64.150", 7000);

j.set("key1", "value1");

String v = j.get("key1");

System.***out***.println(v);

j.close();

}

}

#### 在容器中查看数据

# 进入容器

docker exec -it redis7000 bash

# 运行redis客户端工具

redis-cli

127.0.0.1:6379> keys \*

1) "key1"

127.0.0.1:6379> get key1

"value1"

### 启动多个 redis 容器

启动三个redis容器,将端口分别映射到7000,7001和7002端口

# 如果7000已经启动,不必重复启动

docker run -d --name redis7000 -p 7000:6379 redis

docker run -d --name redis7001 -p 7001:6379 redis

docker run -d --name redis7002 -p 7002:6379 redis

# 查看容器

docker ps –a

[root@localhost ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

756f5227dd64 redis "docker-entrypoint.s…" 6 seconds ago Up 4 seconds 0.0.0.0:7001->6379/tcp redis7001

bc87e35664c8 redis "docker-entrypoint.s…" 13 seconds ago Up 11 seconds 0.0.0.0:7002->6379/tcp redis7002

c3700e3e4c73 redis "docker-entrypoint.s…" 2 hours ago Up 2 hours 0.0.0.0:7000->6379/tcp redis70001

### jedis 分片测试

编写测试方法

@Test

**public** **void** test2() {

JedisPoolConfig cfg = **new** JedisPoolConfig();

cfg.setMaxTotal(500);

cfg.setMaxIdle(20);

List<JedisShardInfo> shards = **new** ArrayList<JedisShardInfo>();

shards.add(**new** JedisShardInfo("192.168.64.150", 7000));

shards.add(**new** JedisShardInfo("192.168.64.150", 7001));

shards.add(**new** JedisShardInfo("192.168.64.150", 7002));

ShardedJedisPool pool = **new** ShardedJedisPool(cfg, shards);

ShardedJedis j = pool.getResource();

**for** (**int** i = 0; i < 100; i++) {

j.set("key"+i, "value"+i);

}

pool.close();

}

### 在容器中查看数据

分别进入三个redis容器,执行 keys \*查看结果

docker exec -it redis7000 redis-cli

docker exec -it redis7001 redis-cli

docker exec -it redis7002 redis-cli

## 容器互联

### 创建docker虚拟网络

docker network rm my-net #删除之前创建的虚拟网卡

docker network create --subnet=172.18.0.0/24 dockernet

ifconfig

## mysql

### 加载 mariadb 镜像

docker load < mariadb-docker-image.gz

### 创建数据卷

docker volume create mysql-data

### 启动 mariadb 容器,并挂载数据卷

docker run -d --name mysql --net dockernet --ip 172.18.0.11 -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=root -p 3306:3306 mariadb

docker ps -a

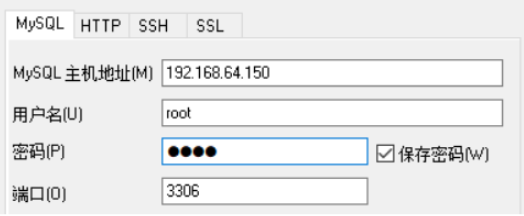
docker inspect mysql

### 进入容器,登录mysql测试

docker exec -it mysql bash

mysql -uroot -p

### 连接数据库,导入数据





选择 资料/docker/EasyMall/easymall.sql

## tomcat

### 加载 tomcat 镜像

docker load < tomcat7-docker-image.gz

docker image ls

### 启动 tomcat 容器

docker run -d --name tomcat --net dockernet --ip 172.18.0.12 -v /opt/webapps:/usr/tomcat/webapps -p 8080:8080 tomcat:7

### 访问测试

192.168.64.150:8080