# Docker容器技术

## 需求场景

Docker 提供轻量的虚拟化，**通过docker镜像能创建出多个容器，**可以把镜像快速部署到无数台服务器上。

Docker是开发人员的系统管理员使用容器开发，部署和运行应用程序的平台。使用Linux容器部署应用程序称为容器化。容器不是新的，但他们勇于轻松部署应用程序。

## Docker与虚拟机对比

每个虚拟器需要安装一个操作系统

虚拟机：资源全部隔离，cpu、内存、磁盘相互独立，一台宿主机中可以安装的虚拟机数量是一定的，有限的，移植性不强（文件很大，拷贝传输困难）。

Docker：部分共享部分隔离，共享cpu，内存，磁盘甚至操作系统，极为轻量,如图-126 所示。

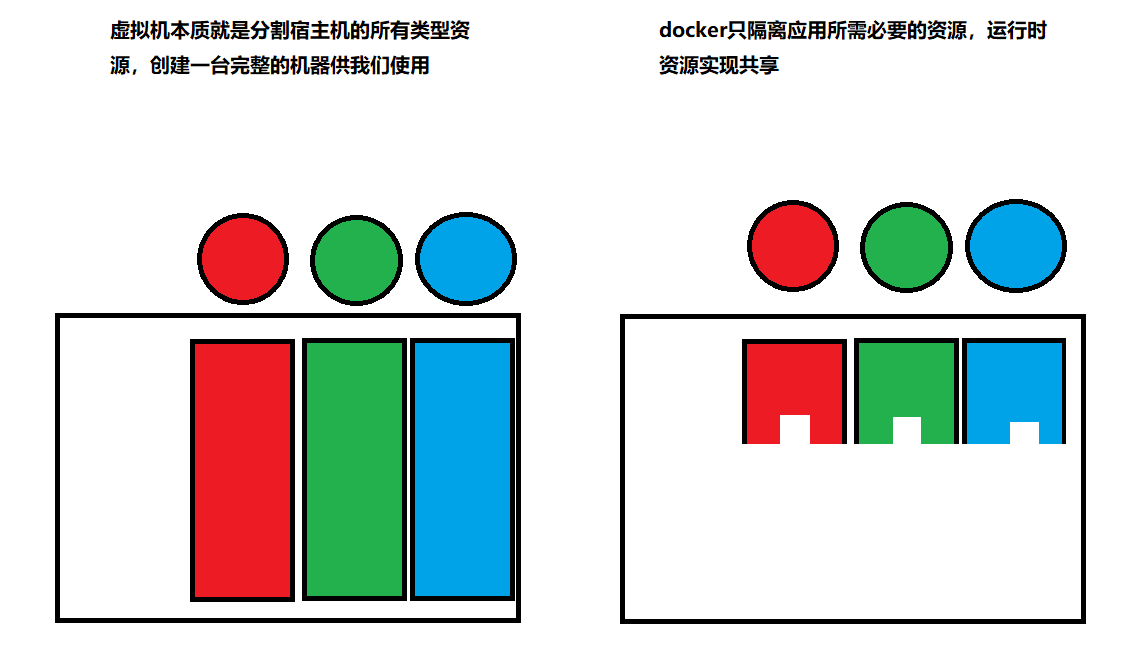


图- 126

## Docker五大要素

1. 沙箱：隔离，将应用之间的必要资源隔离开，防止互相影响，应用之间互相独立。
2. 镜像：模板，centos+tomcat+mysql+redis,镜像可以拆分，传输用dockerfile
3. 容器：镜像的实例，镜像只读，容器可写，容器中可以保存应用产生的零时文件
4. 数据卷：挂载到容器上，用于保存必要数据。比如容器中数据库的数据，但是不建议用，因为重启容器需要挂载同步数据，效率较慢，解决方式：数据库拆分
5. 仓库：与maven仓库概念一致，应用都会保存在仓库中，创建容器是根据镜像的规定进行拉取，可以共享。

## Docker的版本及要求

Docker分为docker-ce（社区版-开源）和docker-ee（企业级-收费）

Docker基于linux3.8及以上版本64bit内核开发，所以在使用docker前需要确认linux内核版本，一般centos7以上。

查看linux版本命令：

uname -a

执行结果如图-127 所示.

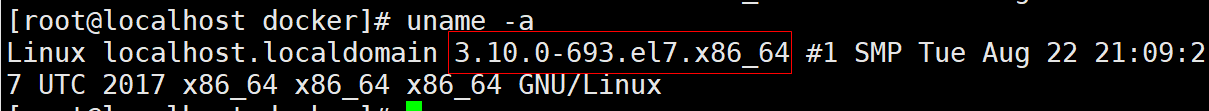


图- 127

## 虚拟机的安装

安装的是VMware虚拟机

## 安装docker

如何快速进入某个多层路径?

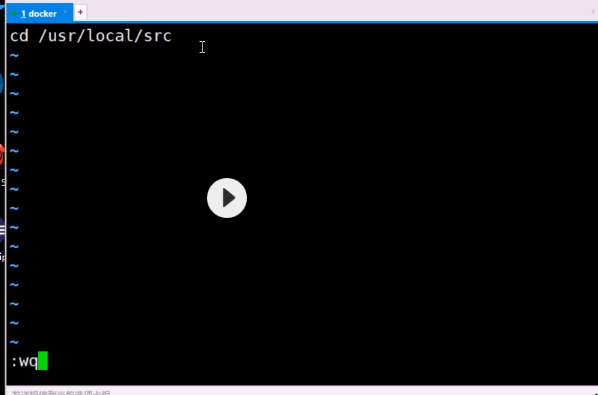
执行以下命令:

vim go 回车

进入以下页面,输入要进入的多层路径,

如:cd /usr/local/src

然后按Esc , 输入 :wq保存退出.



返回界面之后,可执行:

. go 回车.,即可进入src文件夹下

### 验证docker是否安装好

执行docker version 命令查看docker的版本，**虚拟机中的docker已经装好了，不要执行卸载docker和安装docker的命令。**

[root@localhost ~]# docker version

Client:

Version: 17.12.1-ce

API version: 1.35

Go version: go1.9.4

Git commit: 7390fc6

Built: Tue Feb 27 22:15:20 2018

OS/Arch: linux/amd64

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

### 卸载老版本的 docker 及其相关依赖

sudo yum remove docker docker-common container-selinux docker-selinux docker-engine

### 安装 yum-utils，它提供了 yum-config-manager，可用来管理yum源

sudo yum install -y yum-utils

### 添加yum源

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

### 更新索引

sudo yum makecache fast

### 安装 docker-ce

sudo yum install docker-ce

## 启动 docker

//查看docker状态

systemctl status docker

Active: inactive (dead)

//启动docker

systemctl start docker

//停止docker

service docker stop

//重启docker

service docker restart

### 查看镜像和容器

//查看装好的镜像

docker images

命令执行完后结果如图-132 所示。

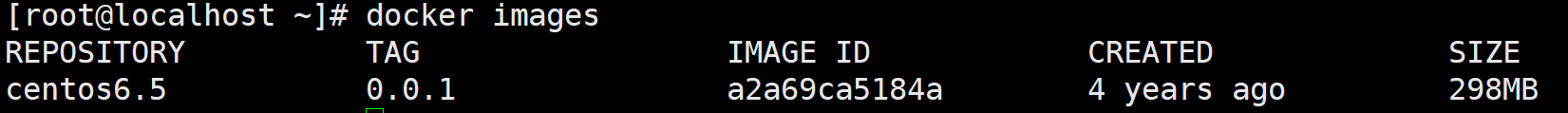


图- 132

//查看容器

docker ps -a

命令执行完后结果如图-133 所示。

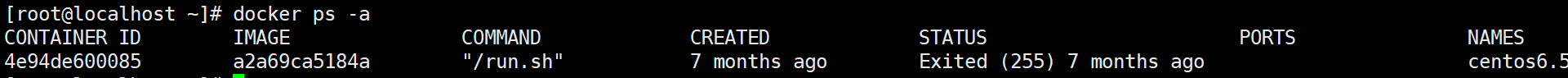


图- 133

## docker命令

|  |  |
| --- | --- |
| 命令 | 用法 |
| yum -y install docker-ce | 下载最新版的docker |
| service docker start | 启动Docker服务 |
| service docker stop | 停止Docker服务 |
| service docker restart | 重新启动Docker服务 |
| docker version | 查看Docker的版本号 |
| docker pull 镜像地址:版本 | 从镜像仓库中下载 |
| docker save a2a69ca5184a > jt-centOS6.tar | 根据镜像id导出镜像 |
| docker save -o redis-3.2.8.tar redis:3.2.8 | 根据镜像名称导出镜像 |
| docker load -i docker-centos-6.5.tar | 指定jar包导入镜像文件 |
| docker rmi a2a69ca5184a | 根据Id号删除镜像文件 |
| docker rmi -f a2a69ca5184a | 强制删除镜像文件 删除镜像前需要先关闭容器 |
| docker images | 查询所有镜像文件 |
| docker inspect index.alauda.cn/tutum/centos:6.5 | 查看镜像文件细节信息 |
| docker tag 旧镜像名称和端口 redis-ali:0.0.1 | 修改镜像的名称 |
| docker build -t 镜像名称:版本号 | 根据dockerfile来创建镜像文件 |
| docker run -d --name 容器名 镜像名:版本号 | 根据镜像名称启动容器 |
| docker run -d --name 容器名(自定) 镜像id号 | 根据镜像id启动容器 |
| docker run -d -p 虚拟机端口:镜像端口 --name 容器名 镜像名:版本号 | 启动容器,并指定暴露端口 |
| docker ps | 查看活动的docker容器进程 |
| Docker ps -a/-all | 查看全部的容器 |
| docker exec -it 容器id bash | 进入指定的容器 |
| docker stop 容器Id号 | 停止指定容器 |
| docker start 容器Id号 | 启动创建好的容器 |
| docker stop $(docker ps -q) & docker rm $(docker ps -aq) | 关闭和删除所有的容器 |
| docker rm 容器Id | 删除指定的容器 |

## Redis案例

### 加载镜像

#### 上传文件

使用fz进入/usr/local/src创建redis文件夹

上传redis-3.2.8.tar到/usr/local/src/redis

#### 加载镜像

cd /usr/local/src/redis

docker load -i redis-3.2.8.tar

#### 查看镜像

docker images

执行结果如图-134 所示

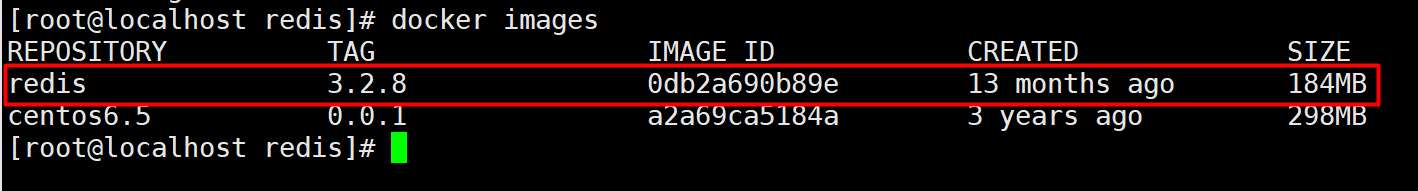


图- 134

### 创建容器(创建镜像redis的实例)

docker run -d --name redis7000 -p 7000:6379 redis:3.2.8

参数说明：

-d，则containter将会运行在后台模式(Detached mode)

--name 实例名称

-p 对外程序访问端口7000，宿主机映射的redis端口6379

最后的redis为镜像的名称

访问过程如图-135 所示.

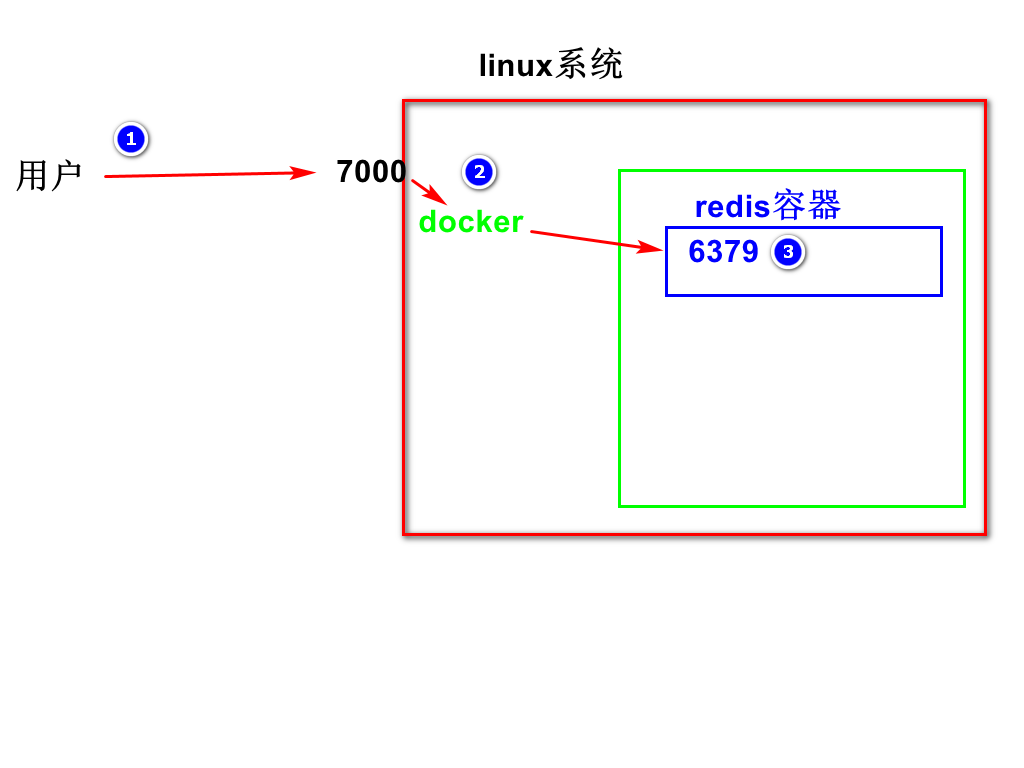


图- 135

查看容器

docker ps –a

显示结果如图-136 所示

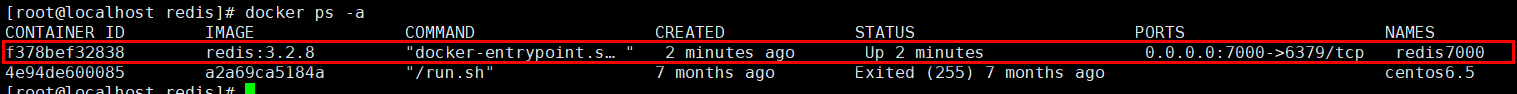


图- 136

### 测试单个节点

创建maven quick start项目，项目名jt-redis。

继承jt-dubbo-parent

pom.xml中把junit版本改成4.12

创建测试类 JedisTest，拷贝下面的方法

@Test //完成单实例链接,修改代码中ip地址

public void jedis(){

Jedis jedis = new Jedis("192.168.163.30", 7000);

//jedis.auth("123456");

jedis.set("name", "tony"); //调用redis命令set

String s = jedis.get("name");

System.out.println(s);

jedis.close();

}

### 查看服务器上的数据

进入到redis7000容器中，进入到容器中执行命令

[root@localhost redis]# docker exec -it redis7000 bash

root@9cc7930f4c69:/data# redis-cli

127.0.0.1:6379> keys \*

1) "name"

127.0.0.1:6379> get name

"tony"

//退出redis客户端

127.0.0.1:6379> exit

//退出容器

root@9cc7930f4c69:/data# exit

exit

[root@localhost redis]#

### 创建多个redis容器(redis分片)

访问过程如图- 137所示

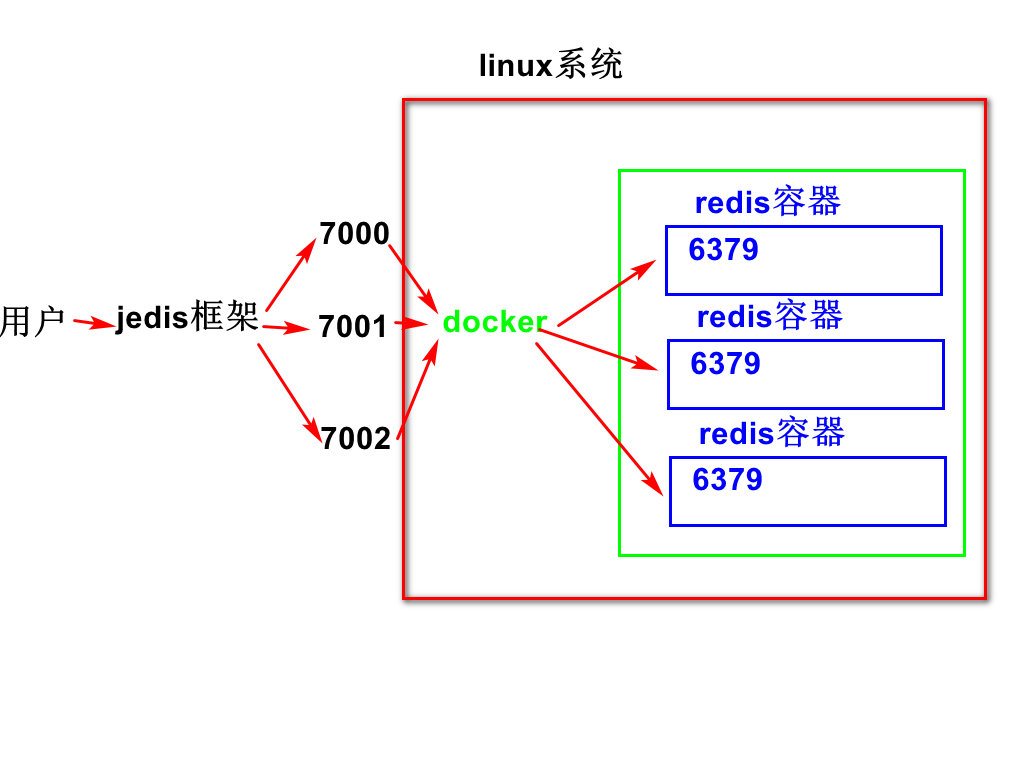


图- 137

docker run -d --name redis7001 -p 7001:6379 redis:3.2.8

docker run -d --name redis7002 -p 7002:6379 redis:3.2.8

### 查看容器

docker ps –a

显示的结果如图-138 所示

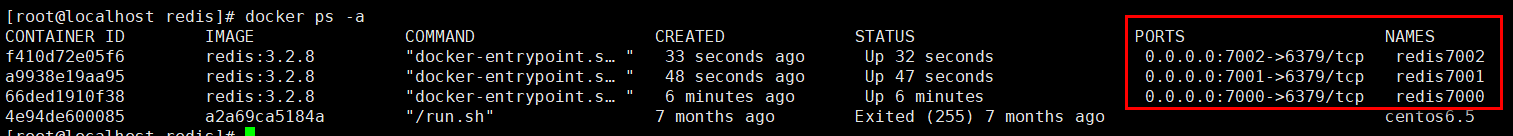


图- 138

### 测试分片

拷贝代码

//redis测试分片

@Test

public void test02(){

//2.创建分片的连接池

JedisPoolConfig poolConfig = new JedisPoolConfig();

poolConfig.setMaxTotal(500);

poolConfig.setMaxIdle(20);

//3.准备redis的分片

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

shards.add(new JedisShardInfo("192.168.65.110", 7000));

shards.add(new JedisShardInfo("192.168.65.110", 7001));

shards.add(new JedisShardInfo("192.168.65.110", 7002));

//1.创建分片的对象

ShardedJedisPool jedisPool =

new ShardedJedisPool(poolConfig, shards);

//获取jedis对象

ShardedJedis shardedJedis = jedisPool.getResource();

//5.redis的存取值操作

for (int i = 0; i < 9; i++) {

shardedJedis.set("n"+i,"我是分片操作"+i);

}

}

### 进入3个容器内部查看数据

//进入到第一个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7000 bash

root@4fb42d95757f:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n8"

2) "n0"

3) "n6"

4) "n3"

127.0.0.1:6379> exit

root@4fb42d95757f:/data# exit

exit

//进入到第二个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7001 bash

root@1418ed384483:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n2"

127.0.0.1:6379> exit

root@1418ed384483:/data# exit

exit

//进入到第三个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7002 bash

root@58de528a1c05:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n5"

2) "n1"

3) "n7"

4) "n4"

127.0.0.1:6379> exit

root@58de528a1c05:/data# exit

### 删除容器

#查看所有容器

docker ps -a

#停止实例(停止容器)

docker stop redis7000

//运行代码会报错

#开启实例

docker start redis7000

#删除容器,先要停止容器

docker stop redis7000

docker rm redis7000

docker ps -a

docker run -d --name redis7000 -p 7000:6379 redis:3.2.8

## 创建网络

不指定ip地址tomcat无法访问mysql，如图-139 所示

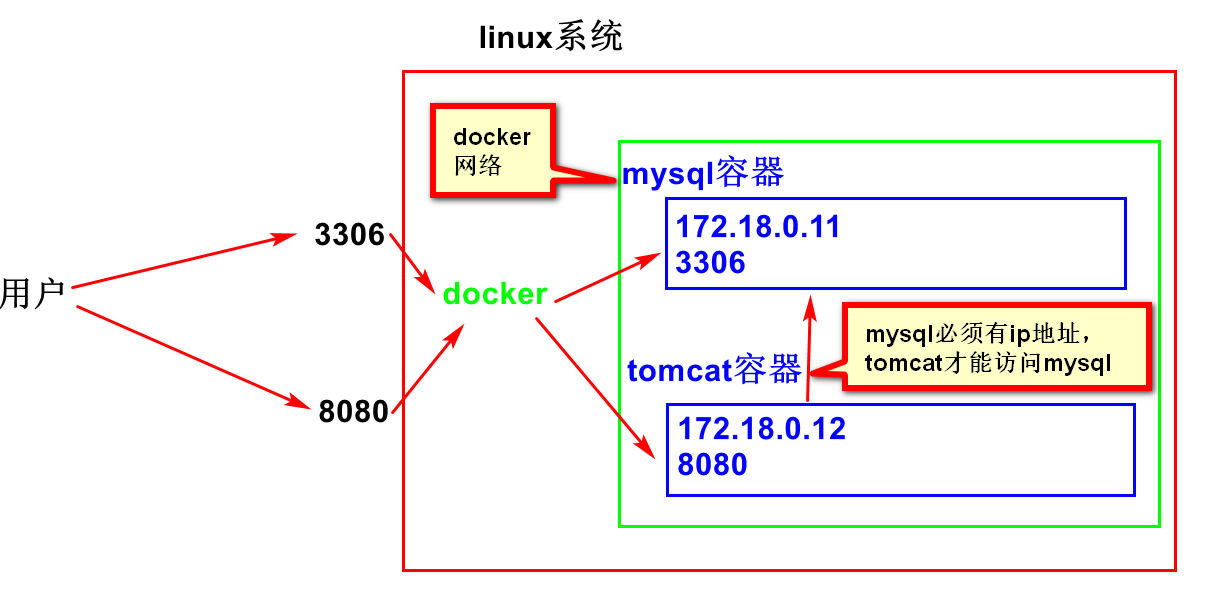


图- 139

### 查看ip地址

启动docker

systemctl start docker

[root@localhost ~]# ifconfig

docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.216.201 netmask 255.255.255.0 broadcast 192.168.216.255

### 创建网络

执行docker network create --subnet=172.18.0.0/24 dockernet创建一个网络

docker network create --subnet=172.18.0.0/24 dockernet

a5cd262aeff63cdeefbab3f768ccc542a635c658b51e4e95307b03adb05a69d0

//执行ifconfig，查看ip信息，发现多了一个172.18.0.1 IP

[root@localhost ~]# ifconfig

br-a5cd262aeff6: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

inet 172.18.0.1 netmask 255.255.255.0 broadcast 172.18.0.255

docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.216.201 netmask 255.255.255.0 broadcast 192.168.216.255

## 安装mysql容器

### 上传文件

上传mysql-image.tar到/usr/local/src文件夹中。

确认上传完成后，再执行下面的命令，否则加载mysql镜像失败

### 加载mysql镜像

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

mysql-image.tar redis

//加载镜像

[root@localhost src]# docker load -i mysql-image.tar

//查看镜像，刚加载的镜像的repository,tag为none

[root@localhost src]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 44a8e1a5c0b2 16 months ago 407MB

//设置镜像的名称

[root@localhost src]# docker tag 44a8e1a5c0b2 docker.io/mysql:5.7

//再次查看镜像

[root@localhost src]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

mysql 5.7 44a8e1a5c0b2 16 months ago 407MB

### 创建mysql容器

//启动mysql docker容器，设置root密码，设置默认编码，指定端口映射。

[root@localhost src]# docker run --net dockernet --ip 172.18.0.11 --name=mysql-container -it -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root mysql:5.7 --character-set-server=utf8

[root@localhost src]# docker ps -a

//查看mysql的ip地址

[root@localhost src]# docker inspect mysql-container

[

"Networks": {

"dockernet": {

"IPAMConfig": {

"IPv4Address": "172.18.0.11"

},

]

### 导入数据库

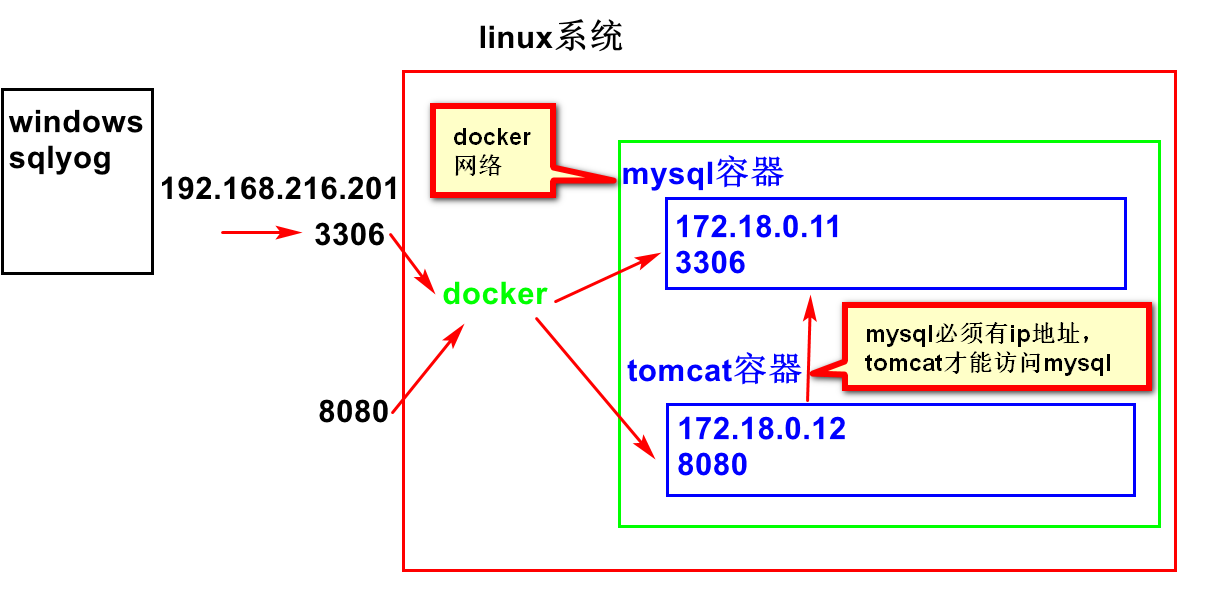
执行过程如图-140 所示

图- 140

在windows启动sqlyog客户端，导入数据,Ip地址是虚拟机的ip

关闭以前的连接，创建新的连接，配置信息如图-141 所示

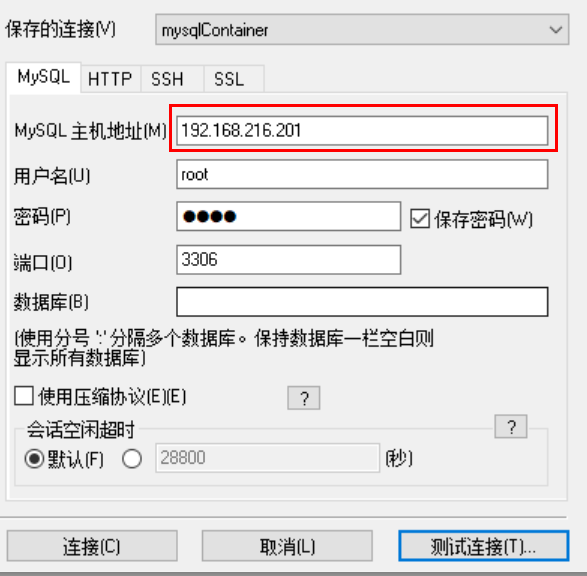


图- 141

连接成功后，只显示4个数据库，如图- 142所示

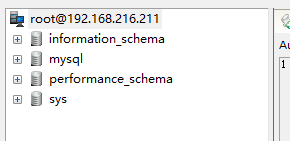


图- 142

导入数据库，如图-143 所示。

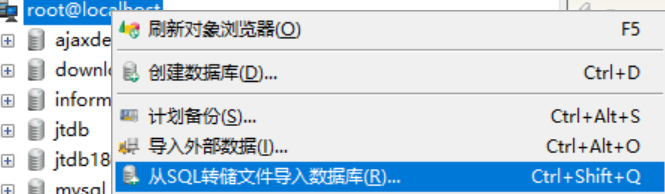


图- 143

执行easymall.sql

刷新数据库，如图-144 所示。



图- 144

刷新后能看到easymall数据库，如图-145 所示。

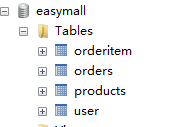


图- 145

### 如查连不上mysql,报10060或10061,解决办法如下

先关机poweroff,再开机，xshell重新打开连接

systemctl stop firewalld.service

systemctl disable firewalld.service

systemctl start docker

docker start mysql-container

## 安装tomcat容器

### 服务器结构图，如图-146 所示

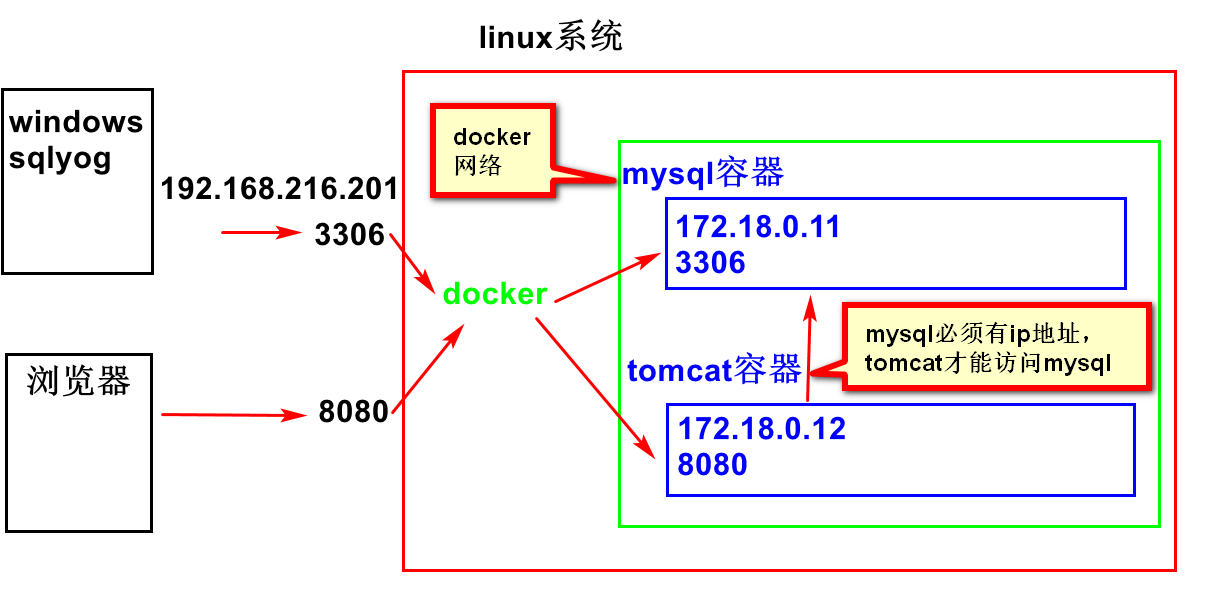


图- 146

### 上传镜像

上传tomcat7-jre8-image.tar到/usr/local/src文件夹中。

### 安装镜像

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

mysql-image.tar redis tomcat7-jre8-image.tar

[root@localhost src]# docker load -i tomcat7-jre8-image.tar

[root@localhost src]# docker images

[root@localhost src]# docker tag 510e45f8f9f7 docker.io/tomcat:7.0

[root@localhost src]# docker images

### 创建容器

docker run -d -p 8080:8080 --net dockernet --ip 172.18.0.12 --name easyMall-tomcat tomcat:7.0

docker ps –a

windows浏览器中访问<http://虚拟机ip:8080>，能看到tomcat首页。

## 部署easymall

### 修改数据库链接地址

* 1. 在资源管理器中设置打开xml的软件

创建1.xml,双击1.xml,测试 一下是不是用你喜欢的编辑软件打开的。如果不是，按照下图设置打开xml的编辑软件，如图- 147所示。



图- 147

* 1. 将ROOT.war改成ROOT.rar,用winrar直接打开，如图-148 所示。

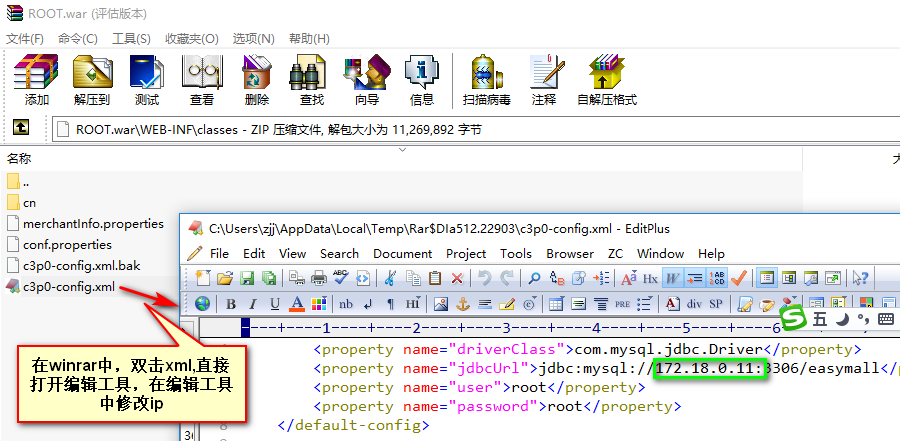


图- 148

将ROOT文件中的\WEB-INF\classes\c3p0-config.xml文件修改,修改数据库ip地址为mysql容器的ip地址172.18.0.11。保存修改，

<property name="jdbcUrl">

jdbc:mysql://172.18.0.11:3306/easymall

</property>

3.使用winrar再次打开ROOT.rar,查看ip地址是否修改过来了。

4.将ROOT.rar改成ROOT.war

5.上传ROOT.war到/usr/local/src文件夹中。

### 查看tomcat容器文件夹目录

docker ps -a

[root@localhost src]# docker exec -it easyMall-tomcat bash

root@c620df99d5e5:/usr/local/tomcat# cd /usr/local/tomcat

root@c620df99d5e5:/usr/local/tomcat# ls

LICENSE RELEASE-NOTES bin include logs temp work

NOTICE RUNNING.txt conf lib native-jni-lib webapps

root@c620df99d5e5:/usr/local/tomcat# cd webapps

root@c620df99d5e5:/usr/local/tomcat/webapps# ls

ROOT docs examples host-manager manager

root@c620df99d5e5:/usr/local/tomcat/webapps# exit

exit

[root@localhost src]#

### 上传root.war到tomcat容器中

[root@localhost src]# cd /usr/local/src

[root@localhost src]# ls

[root@localhost src]# docker cp ROOT.war easyMall-tomcat:/usr/local/tomcat/webapps/

浏览器中访问<http://虚拟机ip:8080/>，如图-149 所示



图- 149

## Dockerfile制作镜像

Dockerfile是docker应用的核心，通过dockerfile能制作出新的镜像，如图- 150所示。

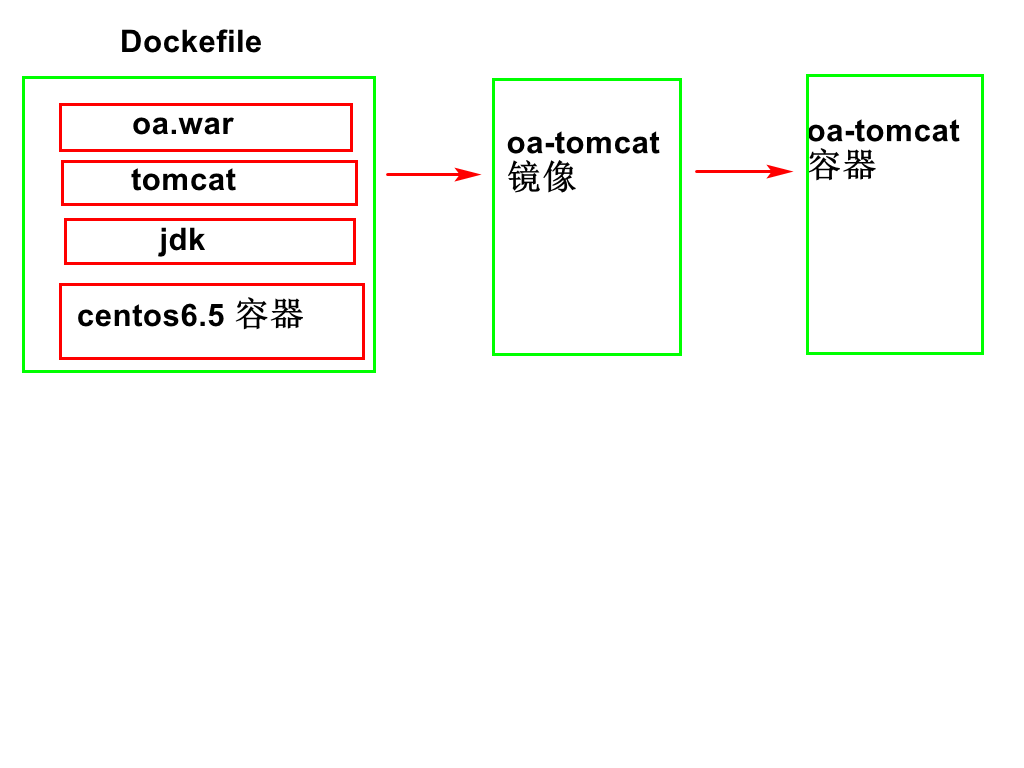


图- 150

### 关键字及含义

|  |  |  |
| --- | --- | --- |
| 序号 | 关键字 | 说明 |
|  | FROM | 指定基础镜像的来源 |
|  | MAINTAINER | 作者 |
|  | ADD | 复制文件，会自动解压 |
|  | WORKDIR | 设置当前工作目录 cd |
|  | VOLUME | 设置数据卷，挂载主机目录 |
|  | EXPOSE | 指定对外暴漏的端口 |
|  | RUN | 执行命令 sh |
|  | CMD | 执行命令 exec，一个Dockerfile只能一个 |
|  | COPY | 复制文件 |
|  | ENTRYPOINT | docker run时参数可以覆盖，指定参数值 |

### 打包生成war

使用eclipse创建出一个maven webapp项目,项目名为oa,修改index.jsp中的内容，打包出war,到target文件夹中找到oa.war。

### 启动CentOS6.5容器

1.启动centos6.5容器

docker ps –a

docker start centos6.5

2.查看容器/usr/local/src文件夹中的内容，文件夹中没有内容。

[root@localhost ~]# docker exec -it centos6.5 bash

bash-4.1# cd /usr/local/src

bash-4.1# ls

bash-4.1# exit

### 生成镜像

1. 上传jdk-8u51-linux-x64.tar.gz到虚拟机的\usr\local\src。
2. 上传apache-tomcat-7.0.55.tar.gz到虚拟机的\usr\local\src
3. 上传oa.war（项目）到虚拟机的\usr\local\src
4. 执行ls命令，查看三个文件是否上传到虚拟机中。
5. 打开记事本，新建一个文件，拷贝下面的内容，保存到Dockerfile.txt文件中，文件名中D是大写的，保存时文件格式必须为utf-8。将文件Dockerfile.txt改名成Dockerfile,去掉.txt扩展名。

#添加contos6.5文件

FROM a2a69ca5184a

#添加JDK1.8 centos+jdk

ADD jdk-8u51-linux-x64.tar.gz /usr/local/src

ENV JAVA\_HOME=/usr/local/src/jdk1.8.0\_51

ENV PATH=$JAVA\_HOME/bin:$PATH

ENV CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

#centOS6.5+JDK1.8+tomcat7

ADD apache-tomcat-7.0.55.tar.gz /usr/local/src

ENV CATALINA\_HOME /usr/local/src/apache-tomcat-7.0.55

ENV PATH=$PATH:$CATALINA\_HOME/bin

#添加oa.war包文件

WORKDIR $CATALINA\_HOME/webapps

RUN mkdir oa

COPY oa.war $CATALINA\_HOME/webapps/oa/oa.war

WORKDIR $CATALINA\_HOME/webapps/oa

RUN jar xvf oa.war

RUN rm oa.war

#对外暴露的端口号

EXPOSE 8080

CMD ["/usr/local/src/apache-tomcat-7.0.55/bin/catalina.sh","run"]

1. 上传Dockerfile到宿主机的/usr/local/src中
2. 制作新的镜像文件

在宿主机上执行docker build命令

镜像名称是oa-tomcat,版本是0.0.1

cd /usr/local/src

// ./代表当前目录

docker build -t oa-tomcat:0.0.1 ./

制作成功会显示“Successfully tagged oa-tomcat:0.0.1”

1. docker images 查看镜像，如图- 151所示



图- 151

## 使用自己制作的镜像创建容器

1.创建容器

docker run -d -p 9080:8080 --net dockernet --ip 172.18.0.13 --name oa-tomcat oa-tomcat:0.0.1

2.访问tomcat和oa

访问<http://虚拟机ip:9080>能看到tomcat首页

访问http://虚拟机ip:9080/oa/index.jsp能看到oa首页

1. 查看容器内容文件结构

[root@localhost src]# docker ps -a

[root@localhost src]# docker exec -it oa-tomcat bash

bash-4.1# cd /usr/local/src

bash-4.1# ls

apache-tomcat-7.0.55 jdk1.8.0\_51

bash-4.1# cd apache-tomcat-7.0.55/

bash-4.1# ls

LICENSE RELEASE-NOTES bin lib temp work

NOTICE RUNNING.txt conf logs webapps

bash-4.1# cd webapps

bash-4.1# ls

ROOT docs examples host-manager manager oa

bash-4.1# cd oa

bash-4.1# ls

META-INF WEB-INF index.jsp

bash-4.1# cd ../

bash-4.1# cd ../

bash-4.1# pwd

/usr/local/src/apache-tomcat-7.0.55

bash-4.1# cd conf

bash-4.1# ls

server.xml

bash-4.1# exit

1. 把容器中的文件拷贝出来

[root@localhost src]# pwd

/usr/local/src

//.代表当前目录

[root@localhost src]# docker cp oa-tomcat:/usr/local/src/apache-tomcat-7.0.55/conf/server.xml .

[root@localhost src]# ls

server.xml

1. 修改server.xml

把server.xml拷贝到windows中。

在124行host标签中添加context标签

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true">

<Context path="" docBase="/usr/local/src/apache-tomcat-7.0.55/webapps/oa" debug="0" reloadable="true" crossContext="true"/>

1. 拷贝server.xml到容器中

把windows中的server.xml拷贝到虚拟机的/usr/local/src中。

[root@localhost src]# pwd

/usr/local/src

[root@localhost src]# docker cp server.xml oa-tomcat:/usr/local/src/apache-tomcat-7.0.55/conf/

1. 重启容器

[root@localhost src]# docker ps -a

[root@localhost src]# docker restart oa-tomcat

1. 浏览器中访问

<http://192.168.216.201:9080/index.jsp能看到oa>首页

## 生成镜像文件

生成镜像文件tar的作用如图-152 所示.

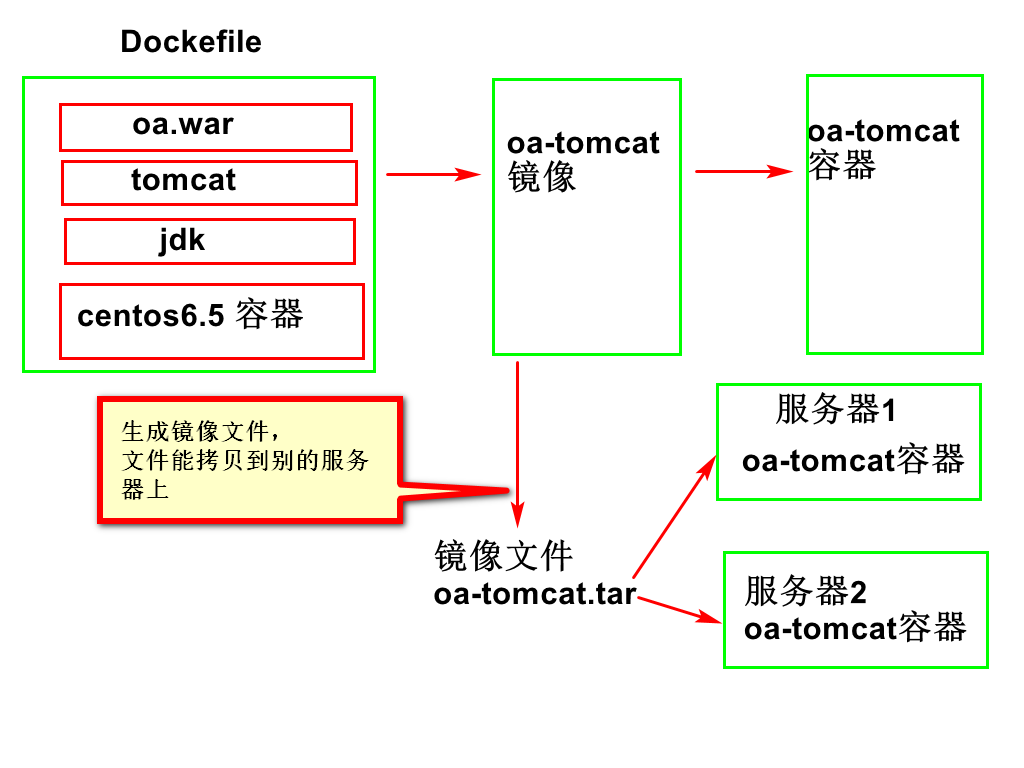


图- 152

**以下命令生成镜像文件**

[root@localhost src]# docker images

[root@localhost src]# docker save -o oa-tomcat-0.0.1.tar oa-tomcat:0.0.1

[root@localhost src]# ls

oa-tomcat-0.0.1.tar

## 使用制作的镜像文件创建容器

[root@localhost ~]# docker ps -a

[root@localhost ~]# docker stop oa-tomcat

//移出容器

[root@localhost ~]# docker rm oa-tomcat

//移出镜像

[root@localhost ~]# docker rmi oa-tomcat:0.0.1

Untagged: oa-tomcat:0.0.1

Deleted: sha256:1274a0931b333171acc17751da76407b935999ad6ee0cb6078c503828c15a3e6

//没有oa-tomcat容器

[root@localhost src]# docker images

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

oa-tomcat-0.0.1.tar

[root@localhost src]# docker load -i oa-tomcat-0.0.1.tar

[root@localhost src]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

oa-tomcat 0.0.1 1274a0931b33 15 hours ago 653MB

[root@localhost src]# docker run -d -p 9180:8080 --net dockernet --ip 172.18.0.19 --name oa-tomcat2 oa-tomcat:0.0.1

b617901309f6db4e4e49a60be22f8e929b1b3b4d0c24e3a55588ff61224fbe61

[root@localhost src]# docker ps –a

浏览器访问

[http://192.168.216.201:9180](http://192.168.216.201:9180/oa/index.jsp)

http://192.168.216.201:9180/oa/index.jsp

## 总结和拓展

Docker容器一般用于部署重复相同的服务，如：redis集群，tomcat集群Hadoop集群等。适用于快速高效集群扩展。其中Dockerfile是其核心，通过编写dockerfile来实现部署方式的快速传输（本质就是传输文本文件），各个节点通过拉取仓库中的镜像文件进行容器的快速创建。整个过程全自动，省去了运维成本。

Devops:开发测试一体化,人们发现dockerfiles能够实现所有机器环境及应用实现完全一致,而这正好能解决一个我们日常开发中遇到的一个非常麻烦的问题,:开发和测试生产环境的不一致导致开发人员完成任务后,到测试环节却出现非bug问题,开发和测试调好后,也不能保证到生产环境没有问题,因为代码一样但是环境不一样会导致很多不必要的麻烦.而docker中的dockerfiles通过统一调取一份镜像,从环境到代码甚至部署过程都是一致的,所以就可以解决开发测试遇到的环境问题.而docker的开发初衷就是开发测试运维一体化.