**实战Docker虚拟化技术**

## Docker入门简介

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的[Linux](http://baike.baidu.com/view/1634.htm)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）。几乎没有性能开销,可以很容易地在机器和数据中心中运行。最重要的是,他们不依赖于任何语言、框架或包括系统。

“Docker”应该是2014年最火爆的技术之一，如果没有听说过，那么你就out了，2015年将开启新的跨越。

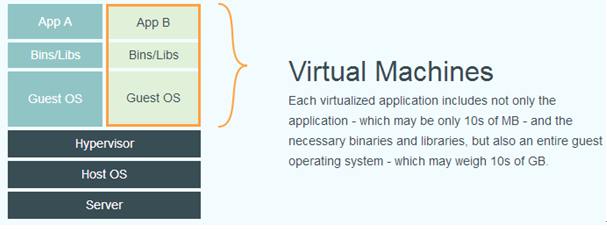
Docker 自开源后受到广泛的关注和讨论，以至于 dotCloud 公司后来都改名为 Docker Inc。Redhat 已经在其 RHEL6.5 中集中支持 Docker；Google 也在其 PaaS 产品中广泛应用。

Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。 Docker 的基础是 Linux 容器（LXC）等技术。

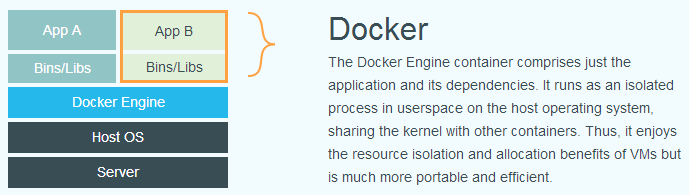
在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。

下面对比了Docker 和传统虚拟化（KVM、XEN等）方式的不同之处，容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统方式则是在硬件的基础上，虚拟出自己的系统，再在系统上部署相关的APP应用。

下图为传统虚拟化方案：



如下为Docker虚拟化方案：



Docker虚拟化有三个概念需要理解，分别镜像、容器、仓库。

1. 镜像：docker的镜像其实就是模板，跟我们常见的ISO镜像类似，是一个样板。
2. 容器：使用镜像常见的应用或者系统，我们称之为一个容器。
3. 仓库：仓库是存放镜像的地方，分为公开仓库（Public）和私有仓库（Private）两种形式。

## Docker虚拟化特点

## 跟传统VM比较具有如下优点：

## 操作启动快

## 运行时的性能可以获取极大提升，管理操作（启动，停止，开始，重启等等） 都是以秒或毫秒为单位的。

## 轻量级虚拟化

## 你会拥有足够的“操作系统”，仅需添加或减小镜像即可。在一台服务器上可以布署100~1000个Containers容器。但是传统虚拟化，你虚拟10-20个虚拟机就不错了。

## 开源免费

## 开源的，免费的，低成本的。由现代Linux内核支持并驱动。注\* 轻量的Container必定可以在一个物理机上开启更多“容器”，注定比VMs要便宜。

## 前景及云支持

## 正在越来越受欢迎，包括各大主流公司都在推动docker的快速发展，性能有很大的优势。

## 跟传统VM比较具有如下缺点：

## 目前知道的人比较少

## 相关的技术资料欠缺

## Go语言还没完全成熟

## Docker安装配置

我们这里主要讲解如何在Centos6.x系列服务器安装，默认docker只有在centos6.5以上机器才能使用yum直接安装，如果其他版本需要安装centos扩展源epel。

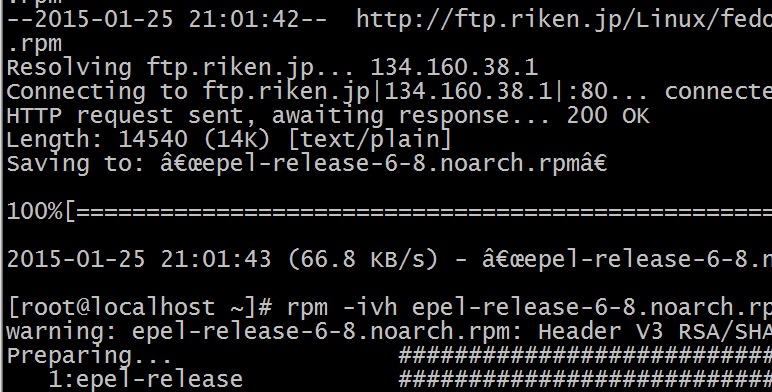
docker官方文档说要求Linux kernel至少3.8以上，一般为centos6.5或者Ubuntu系统，那centos6.5如何来安装呢？

在Centos6.x系列安装docker软件，首先要关闭selinux，然后需要安装相应的epel源，如下：

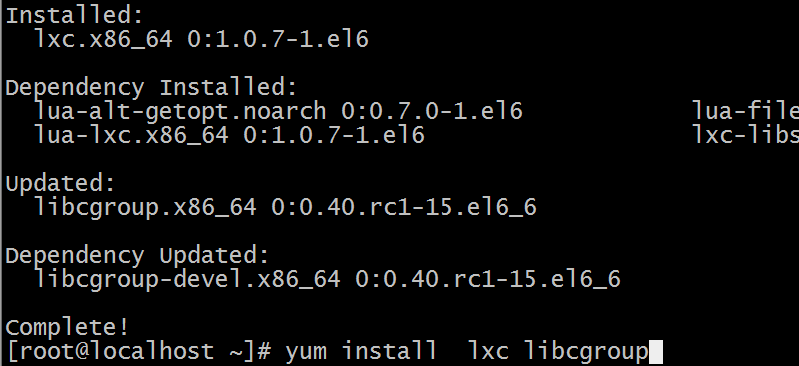
sed -i '/SELINUX/s/enforcing/disabled/g' /etc/selinux/config

wget <http://ftp.riken.jp/Linux/fedora/epel/6/x86_64/epel-release-6-8.noarch.rpm>

rpm -ivh epel-release-6-8.noarch.rpm



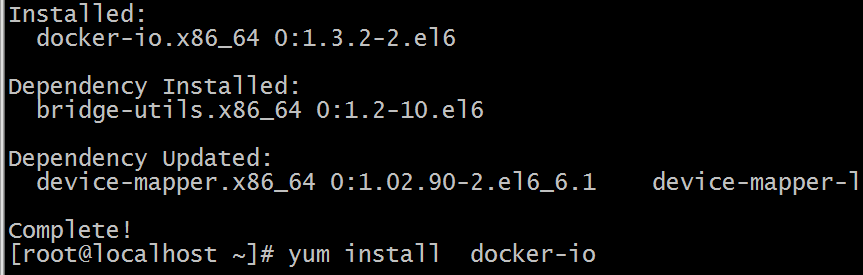
yum install lxc libcgroup device-mapper-event-libs



然后安装docker：

#yum install docker-io

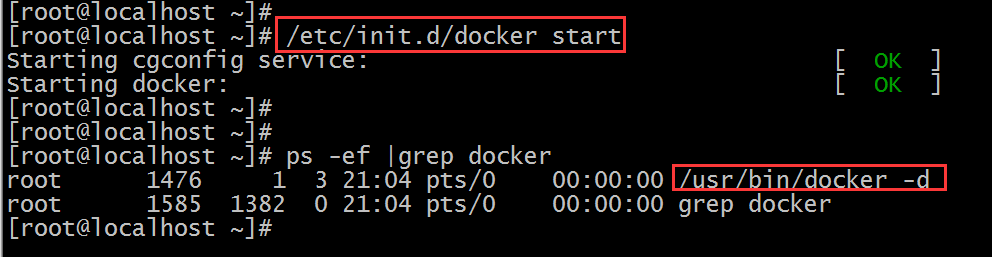
Yum install device-mapper\* -y



安装完后：

启动docker进程：/etc/init.d/docker start

查看docker进程：ps -ef |grep docker



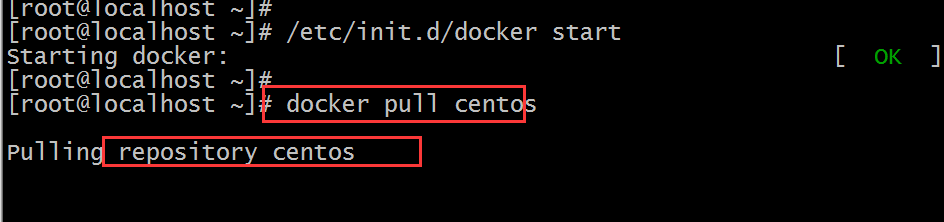
Docker简单使用：

要使用docker虚拟化，首先我们需要去下载一个镜像，然后使用docker命令启动，为了好让大家理解，我这里直接利用docker启动一个新的centos系统哦。

去公共仓库下载centos镜像，下载的速度取决于网速哦。

[http://182.92.188.163/centos6.5.tar](http://182.92.188.163/centos7.tar)

## 如果大家的网络比较慢，可以从这里下载好，然后导入镜像也可以：



## Docker常用命令讲解

## docker version #查看版本

## docker search centos#搜索可用docker镜像

## docker images 查看当前docker所有镜像

## docker pull centos #下载镜像

## cat centos.tar | docker import - centos6 #Docker导入镜像

## docker export id > cenos6.tar #Docker导出镜像

## docker run centos echo "hello word"#在docker容器中运行hello world!

## docker run centos yum install ntpdate#在容器中安装ntpdate的程序

## docker ps -l 命令获得最后一个容器的id，docker ps -a查看所有的容器。

## 运行docker commit 提交刚修改的容器，例如：

## docker commit 2313132 centos:v1

## docker run -i -t centos /bin/bash 在容器里启动一个/bin/bash shell环境，可以登录进入操作，其中-t 表示打开一个终端的意思，-i表示可以交互输入。

## docker run -d centos:v1 /bin/bash ,-d表示在后台启动，以daemon方式启动。

## docker stop id 关闭容器

## docker start id 启动某个容器

## docker rm id 删除容器，docker rmi images删除镜像

## docker run -d -p 80:80 -p 8022:22 centos:v2，解析：-p指定容器启动后docker上运行的端口映射及容器里运行的端口，80:80，第一个80表示docker系统上的80，第二个80表示docker虚拟机里面的端口。用户默认访问本机80端口，自动映射到容器里面的80端口。

## docker exec -it id /bin/bash

## Docker独立IP及容器互联

## 内置bridge（nat）

## 缺点：

## 需要配套服务注册/发现，否则宿主上端口分配困难，容易冲突。

## 由于每个容器暴露的端口都不一致，造成前端路由层nginx配置（proxy\_pass）里无法使用dns的方式。

## 端口映射要在容器启动时就指定好，后期无法变更。

## 测试发现nat不支持websocket。

## 自建桥接网络

## 优点：

## 每个容器都有独立ip，对外提供服务，如nginx+php，nginx+resin，都可以使用默认的80端口

## 由于容器暴露端口都可以使用80端口，因此前端路由层nginx配置（proxy\_pass）里可以使用dns的方式。

## 无需为了后期端口映射添加而烦恼

## 桥接支持websocket

## 当我们把docker镜像下载下来之后，启动一个容器，登录容器我们会发现容器的ip如下图所示：

## 停止服务

## /etc/init.d/docker stop

## 关掉docker0

## ifconfig docker0 down

## 删除docker

## brctl  delbr  docker0

## 增加网桥br0

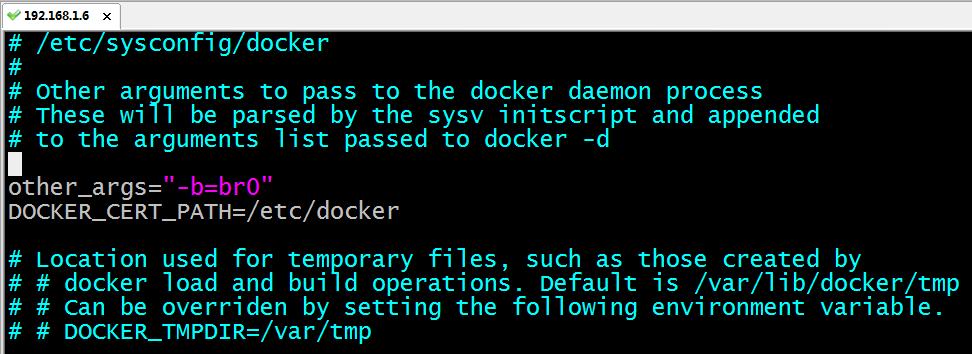
## yum install bridge-utils

## CentOS6.5下的配置：

## vim  /etc/sysconfig/docker

## other\_args="-b=br0"

## 如下图：



## 配置bridge桥接网络：

## 在/etc/sysconfig/network-scripts/下，修改ifcfg-eth0网卡配置，同时增加ifcfg-br0桥接网卡配置如下：

## vi ifcfg-eth0内容如下：

## DEVICE=eth0

## BOOTPROTO=none

## ONBOOT=yes

## TYPE=Ethernet

## BOOTPROTO=static

## BRIDGE=br0

## vi ifcfg-br0内容如下：

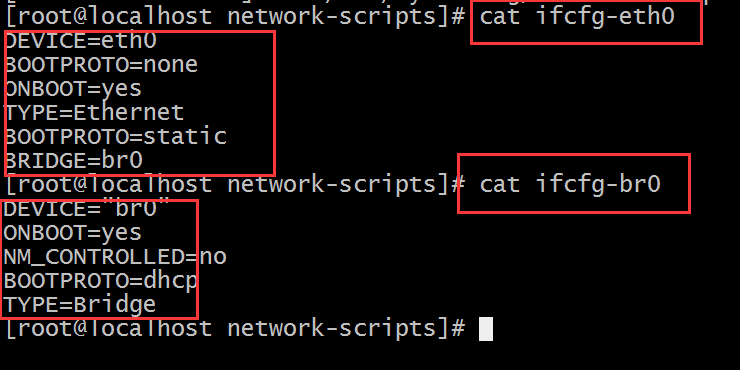
## DEVICE="br0"

## ONBOOT=yes

## NM\_CONTROLLED=no

## BOOTPROTO=dhcp

## TYPE=Bridge

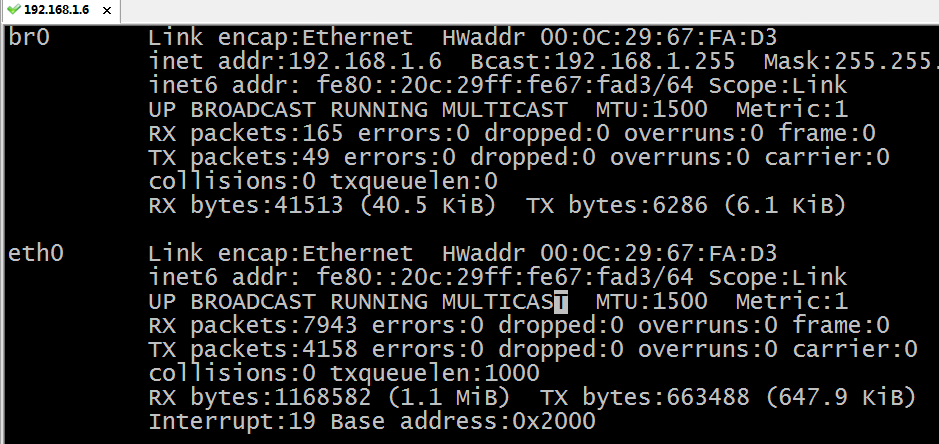


## 启动docker服务

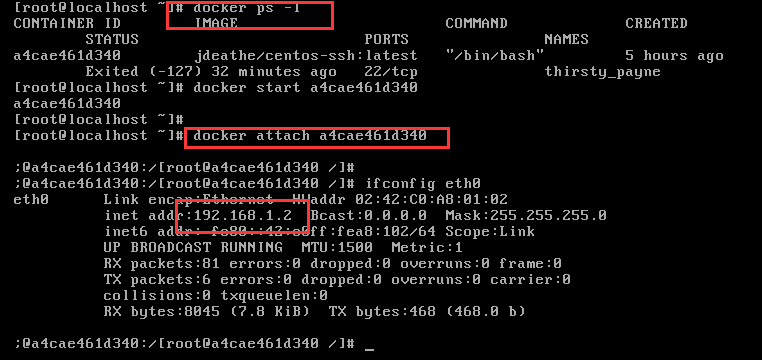
## /etc/init.d/docker start

## 查看服务器网卡信息如下：

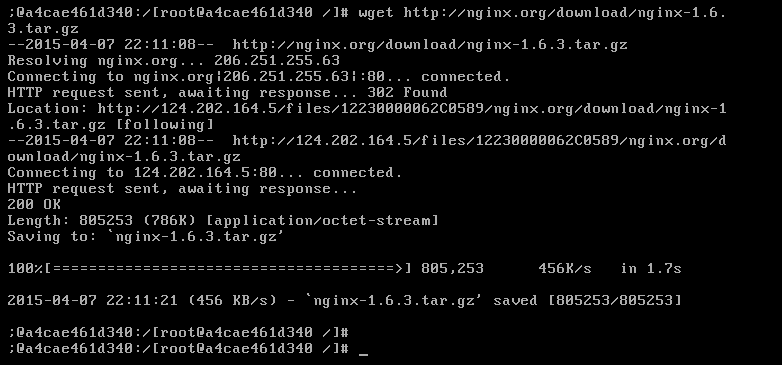
## 启动一个新的docker客户端，查看客户端ip如下：



## 进入已经启动过的容器命令：docker attach 容器ID 即可。



## 从外网下载nginx包：



## CentOS7下Docker桥接网络配置

## docker默认提供了一个隔离的内网环境，启动时会建立一个docker0的虚拟网卡，每个容器都是连接到docker0网卡上的。而docker0的ip段为172.17.0.1，如果想让容器与宿主机同一网段的其他机器访问，就必须在启动docker的时候将某个端口映射到宿主机的端口。这样如果大家会发现很麻烦，而且在企业里面也没这么使用的，比较弱。

## 大家应该知道KVM的桥接网络非常方便，其实docker也比较方便，至少不是自带的桥接而已，上次课程我们讲解了docker容器在centos6.5下的实现方法，今天我们来讲解centos7下如果快速实现docker容器桥接网络，并为容器分配外网IP。

## 物理机的IP信息

## IP：192.168.1.88/24 网关192.168.1.1

## 停止docker服务

## service docker stop

## 删除docker0网卡

## ip link set dev docker0 down

## brctl delbr docker0

## 新建桥接网卡br0

## brctl addbr br0

## ip link set dev br0 up

## ip addr add 192.168.1.88/24 dev br0

## #为br0分配物理网络中的ip地址

## ip addr del 192.168.1.109/24 dev eth0 #将宿主机网卡的IP清空

## brctl addif br0 eth0 #将宿主机网卡挂到br0上

## ip route del default #删除原路由

## ip route add default via 192.168.1.1 dev br0 #为br0设置路由

## 设置docker服务启动参数

## vim /etc/sysconfig/docker-network，其他系统可能在/etc/sysconfig/docker下哦。

## 改成如下行即可：

## DOCKER\_NETWORK\_OPTIONS="-b=br0"

## 启动docker服务

## service docker start

## 安装pipework

## git clone https://github.com/jpetazzo/pipework

## cp ~/pipework/pipework /usr/local/bin/

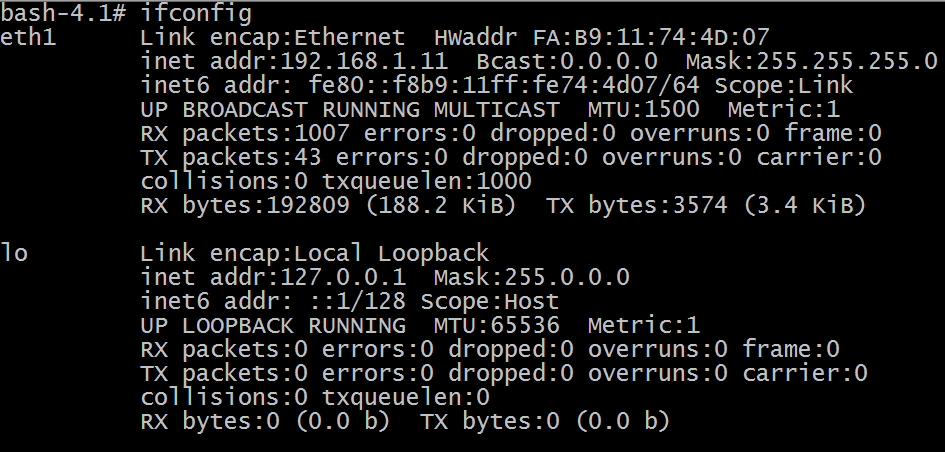
## 启动容器并设置网络

## docker run -itd --net=none --name=lamp2 centos7 /bin/bash

## pipework br0 lamp2 192.168.1.11/24@192.168.1.88

## 进入容器查看ip

## docker attach lamp2



## Docker构建Tomcat WEB服务器

Tomcat是由Apache软件基金会下属的[Jakarta项目](http://zh.wikipedia.org/wiki/Jakartaé¡¹ç®)开发的一个[Servlet](http://zh.wikipedia.org/wiki/Servlet)容器，按照[Sun Microsystems](http://zh.wikipedia.org/wiki/Sun_Microsystems)提供的技术规范，实现了对[Servlet](http://zh.wikipedia.org/wiki/Servlet)和[JavaServer Page](http://zh.wikipedia.org/wiki/JavaServer_Page)（[JSP](http://zh.wikipedia.org/wiki/JSP)）的支持，Tomcat本身也是一个HTTP服务器，可以单独使用，apache是一个以C语言编写的HTTP服务器。Tomcat主要用来解析JSP语言。目前最新版本为8.0。

### Tomcat安装配置

安装tomcat之前需要安装jdk (Java Development Kit) 是 Java 语言的软件开发工具包(SDK)），这里选择jdk-6u18-linux-x64-rpm.bin，bin文件安装跟sh文件方法一样，sh ./jdk-6u18-linux-x64-rpm.bin，回车即可，默认安装到/usr/java/jdk1.6.0\_18目录下。

配置java环境变量，vi /etc/profile 添加如下语句：

export JAVA\_HOME=/usr/java/jdk1.6.0\_18

export CLASSPATH=$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib

export PATH=$JAVA\_HOME/bin:$JAVA\_HOME/jre/bin:$PATH:$HOMR/bin

source /etc/profile //使环境变量立刻生效。

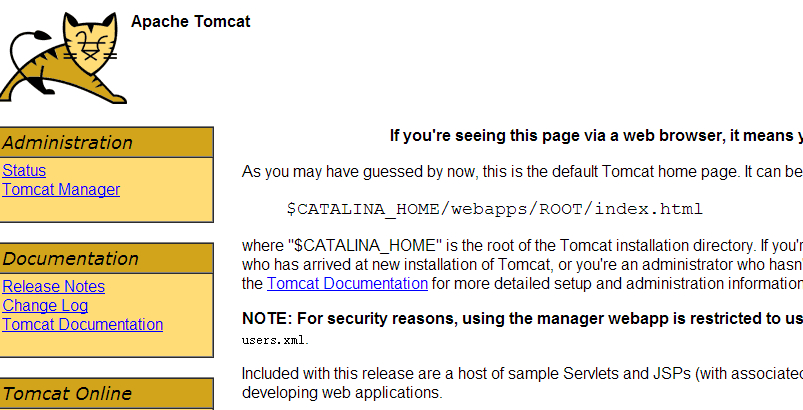
java -version //查看java版本，显示版本为1.6.0\_18，证明安装成功。

在官网下载tomcat相应版本，这里下载的版本为apache-tomcat-6.0.30.tar.gz，下载完后解压：

**tar -xzf apache-tomcat-6.0.30.tar.gz ;mv apache-tomcat-6.0.30 /usr/local/tomcat**

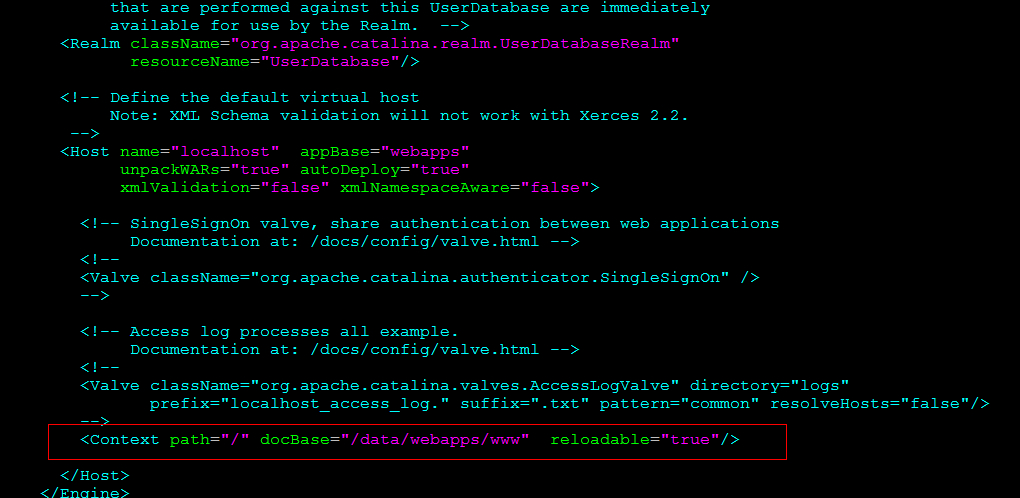
启动tomcat，命令为：/usr/local/tomcat\_test/bin/startup.sh

查看ps -ef |grep tomcat 进程及端口是否存在，通过页面访问可以看到tomcat默认测试页面：



这个画面是默认网站，怎么来创建一个自己的网站页面呢，定义自己的发布目录，方法如下：在server.xml配置文件末尾加入如下行：（附截图）

**<Context path="/" docBase="/data/webapps/www" reloadable="true"/>**



在/data/webapps/www目录下，创建自己的jsp代码，重启tomcat即可访问。

### Tomcat性能优化

线上环境使用默认tomcat配置文件，性能很一般，为了满足大量用户的访问，需要对tomcat进行参数性能优化，具体优化的地方如下：

* Linux内核的优化
* 服务器资源配置的优化
* Tomcat参数优化
* 配置负载集群优化

这里着重讲解tomcat参数的优化：server.xml文件,关闭DNS查询、配置最大并发等参数。

maxThreads：tomcat起动的最大线程数，即同时处理的任务个数，默认值为200

acceptCount：当tomcat起动的线程数达到最大时，接受排队的请求个数，默认值为100

当然这些值都不是越大越好，需要根据实际情况来设定。可以基于测试的基础上来不断的调优分析。Server.xml里面优化如下：

<Connector port="8080"

protocol="org.apache.coyote.http11.Http11NioProtocol"

connectionTimeout="20000"

redirectPort="8443"

maxThreads="5000"

minSpareThreads="20"

acceptCount="1000"

disableUploadTimeout="true"

enableLookups="false"

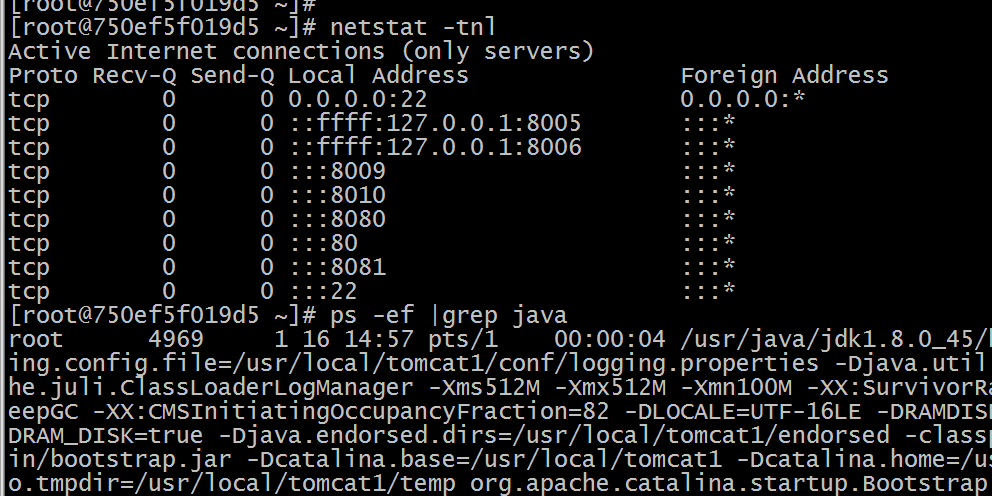
URIEncoding="UTF-8" />

Catalina.sh JVM参数优化，添加如下内容：

CATALINA\_OPTS="$CATALINA\_OPTS –Xms4000M –Xmx4000M –Xmn1000M -XX:SurvivorRatio=4 -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=82 -DLOCALE=UTF-16LE -DRAMDISK=/ -DUSE\_RAM\_DISK=ture -DRAM\_DISK=true"

配置多个tomcat实例，方法也很简单，只需要在服务器上cp多个tomcat，然后修改三个端口和发布目录即可，然后分别启动即可。

为了提升整个网站的性能，还需要在tomcat前面架设nginx web反向代理服务器，用以提高用户高速访问。



## 使用Dockerfile进行docker容器管理

## 我们可以远程登录Linux服务器，那首先需要去设置密码，如果出现如下错误：

## /usr/share/cracklib/pw\_dict.pwd: No such file or directory  PWOpen: No such file or directory

## rpm -e cracklib-dicts --nodeps ;rpm -e pam --nodeps

## yum install pam cracklib-dicts -y

## [root@5255b18871ae /]# echo '123456' | passwd --stdin root

## Changing password for user root.

## passwd: all authentication tokens updated successfully.

## 我们如果现在启动sshd，sshd会报错：

## Could not load host key: /etc/ssh/ssh\_host\_rsa\_key

## Could not load host key: /etc/ssh/ssh\_host\_ecdsa\_key

## 所以我们先生成/etc/ssh/ssh\_host\_rsa\_key和/etc/ssh/ssh\_host\_ecdsa\_key：

## [root@5255b18871ae /]# ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh\_host\_rsa\_key -N ''

## [root@5255b18871ae /]# ssh-keygen -q -t ecdsa -f /etc/ssh/ssh\_host\_ecdsa\_key -N ''

## 然后查一下容器的IP，以daemon方式启动sshd：

## [root@5255b18871ae /]# /usr/sbin/sshd -D

## 我们看到容器IP为192.168.1.3，从外部远程ssh到这个容器：

## 发现容器立即关闭了连接，原因是容器的ssh使用了pam\_loginuid.so模块，我们把它关掉：

## [root@5255b18871ae /]# sed -i '/^session\s\+required\s\+pam\_loginuid.so/s/^/#/' /etc/pam.d/sshd

## 上述命令的意思是：在/etc/pam.d/sshd文件里注释掉"session required pam\_loginuid.so"这一行。

## 然后重新启动sshd：

## [root@5255b18871ae /]# /usr/sbin/sshd -D

## 再次尝试远程ssh登入：

## [root@localhost ~]# ssh root@192.168.1.3

## root@192.168.1.3's password:

## Last login: Tue Dec 2 03:00:07 2014 from 192.168.1.3

## [root@5255b18871ae ~]#

## 登录成功！

## Dockerfile是一个镜像的表示，可以通过Dockerfile来描述构建镜像的步骤，并自动构建一个容器。

## 所有的 Dockerfile 命令格式都是:

## INSTRUCTION arguments

## 虽然指令忽略大小写，但是建议使用大写。

## FROM 命令

## FROM <image>

## 或

## FROM <image>:<tag>

## 这个设置基本的镜像，为后续的命令使用，所以应该作为Dockerfile的第一条指令。

## 比如:

## FROM centos:v1

## 如果没有指定 tag ，则默认tag是latest，如果都没有则会报错。

## RUN 命令

## RUN命令会在上面FROM指定的镜像里执行任何命令，然后提交(commit)结果，提交的镜像会在后面继续用到。

## 两种格式:

## RUN <command> (the command is run in a shell - `/bin/sh -c`)

## 或:

## RUN ["executable", "param1", "param2" ... ] (exec form)

## RUN命令等价于:

## docker run image command

## docker commit container\_id

## 编写Dockerfile

## 根据上面的操作步骤，在docker服务器端创建Dockerfile文件，内容如下：

## # 设置基本的镜像，后续命令都以这个镜像为基础

## FROM centos:v1

## # 作者信息

## MAINTAINER JFEDU.NET

## # RUN命令会在上面指定的镜像里执行任何命令

## RUN yum install passwd openssl openssh-server -y

## RUN echo '123456' | passwd --stdin root

## RUN ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh\_host\_rsa\_key -N ''

## RUN ssh-keygen -q -t ecdsa -f /etc/ssh/ssh\_host\_ecdsa\_key -N ''

## RUN sed -i '/^session\s\+required\s\+pam\_loginuid.so/s/^/#/' /etc/pam.d/sshd

## RUN mkdir -p /root/.ssh && chown root.root /root && chmod 700 /root/.ssh

## # 暴露ssh端口22

## EXPOSE 22

## # 设定运行镜像时的默认命令：输出ip，并以daemon方式启动sshd

## CMD ip addr ls eth0 | awk '{print $2}' | egrep -o '([0-9]+\.){3}[0-9]+';/usr/sbin/sshd -D

## 根据Dockerfile来创建镜像

## 用docker build根据Dockerfile创建镜像(centos:ssh)：

## docker build -t centos:ssh - < Dockerfile

## DockerFile企业案例演示：

## 开启SSH 22端口，并远程连接服务器，dockerfile内容如下：

## # 设置基本的镜像，后续命令都以这个镜像为基础

## FROM centos\_lamp:v1

## # 作者信息

## MAINTAINER JFEDU.NET

## # RUN命令会在上面指定的镜像里执行任何命令

## RUN yum install passwd openssl openssh-server -y

## RUN echo '123456' | passwd --stdin root

## RUN sed -i '/^session\s\+required\s\+pam\_loginuid.so/s/^/#/' /etc/pam.d/sshd

## RUN mkdir -p /root/.ssh && chown root.root /root && chmod 700 /root/.ssh

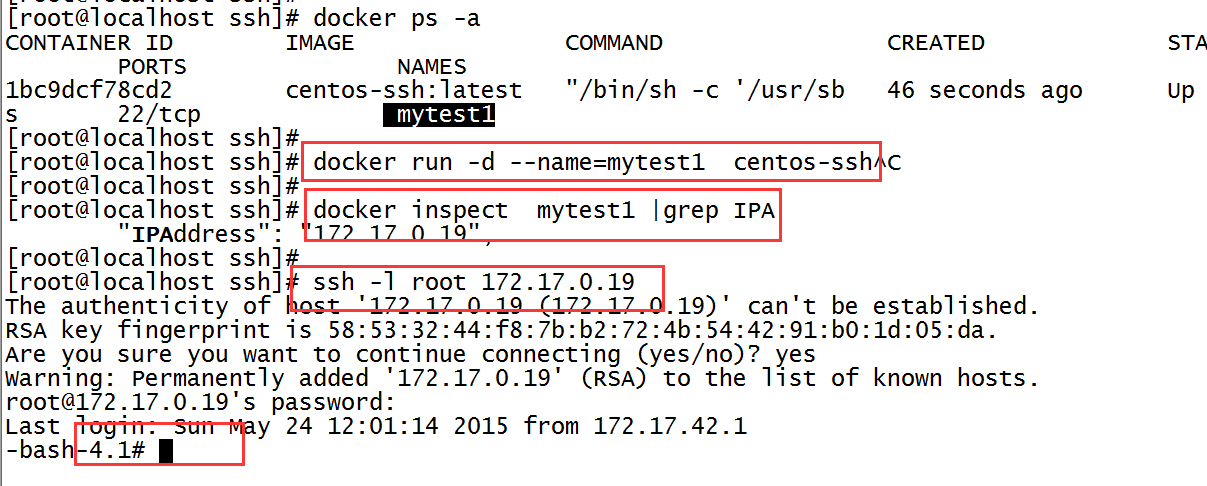
## RUN mkdir /var/run/sshd

## #暴露ssh端口22

## EXPOSE 22

## #设定运行以daemon方式启动sshd

## CMD /usr/sbin/sshd -D



## 开启Apache 80端口，并远程连接服务器，dockerfile内容如下：

## # 设置基本的镜像，后续命令都以这个镜像为基础

## FROM centos\_lamp:v1

## # 作者信息

## MAINTAINER JFEDU.NET

## # RUN命令会在上面指定的镜像里执行任何命令

## RUN yum install pcre-devel -y

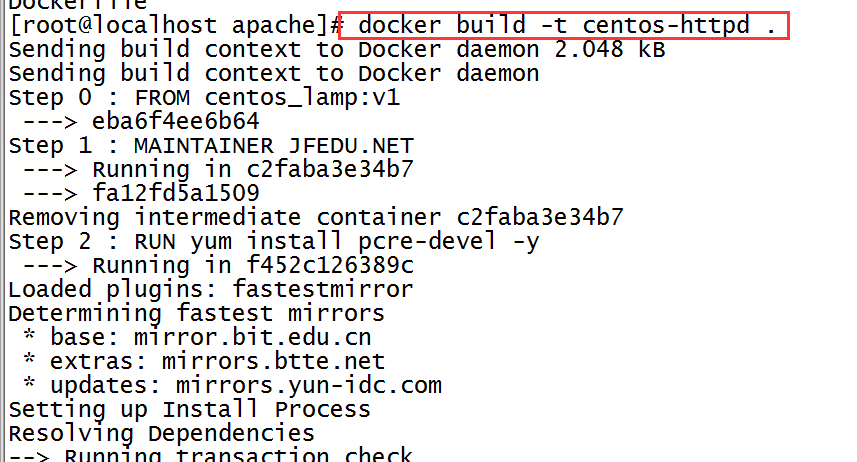
## RUN yum install httpd httpd-devel -y

## #暴露ssh端口80

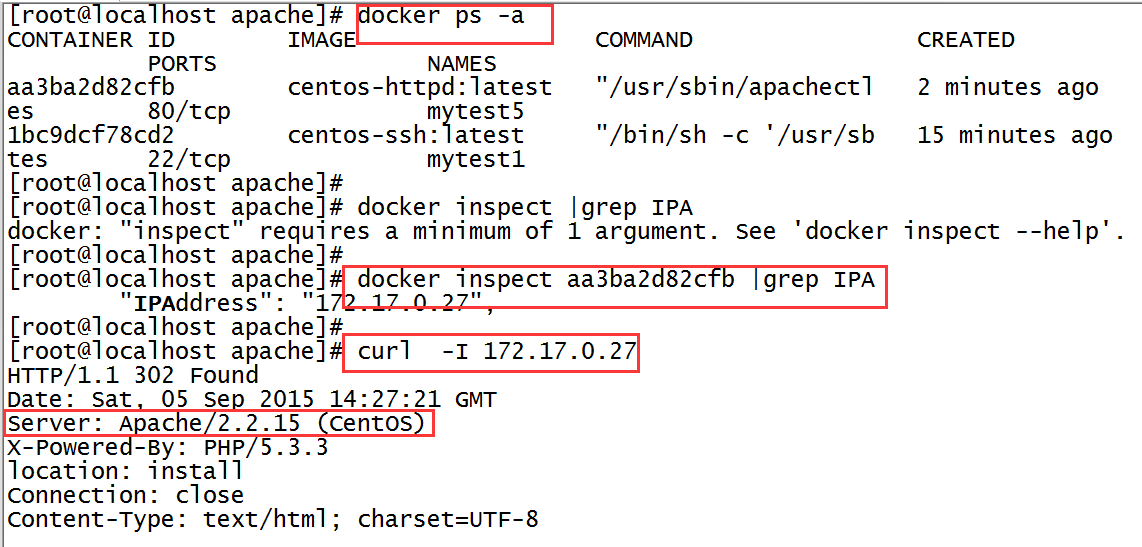
## EXPOSE 80

## #启动httpd

## CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]



## 最终截图如下：



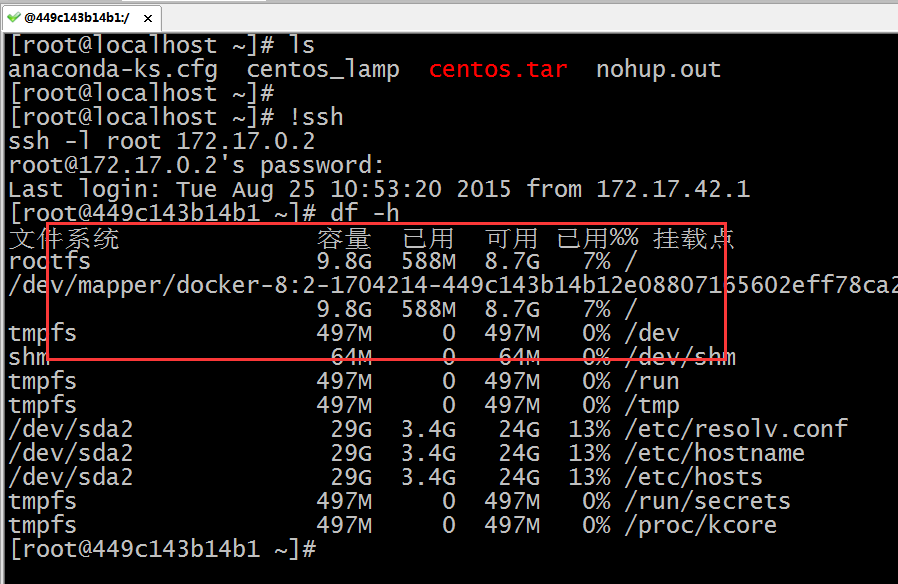
## Docker备份及磁盘扩容

docker容器默认的空间是10G,如果想指定默认容器的大小（在启动容器的时候指定），可以在docker配置文件里通过dm.basesize参数指定，例如：

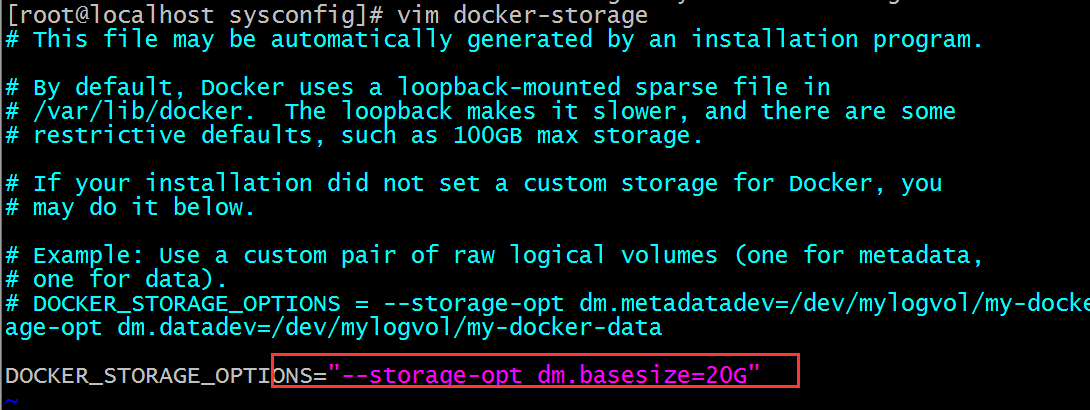
docker -d --storage-opt dm.basesize=20G

是指定默认的大小为20G，具体参数可以参考<https://github.com/docker/docker/tree/master/daemon/graphdriver/devmapper>

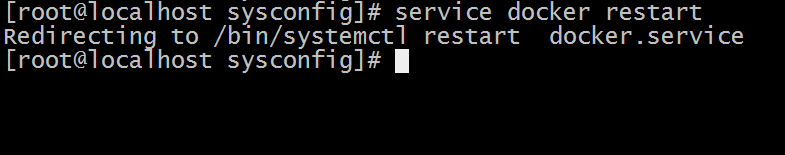
以上方法只适用于新容器生成，并且修改后需要重启docker，无法做到动态给正在运行容器指定大小。（如下图为默认容器大小）



在文件加入如下语句即可：vi /etc/sysconfig/docker-storage



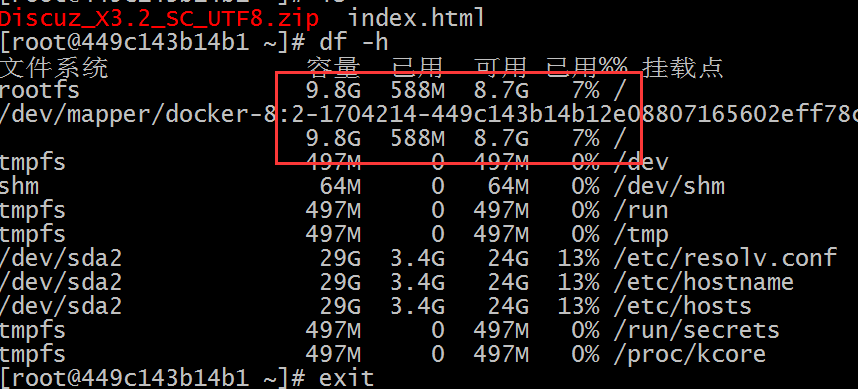
然后重启docker即可;



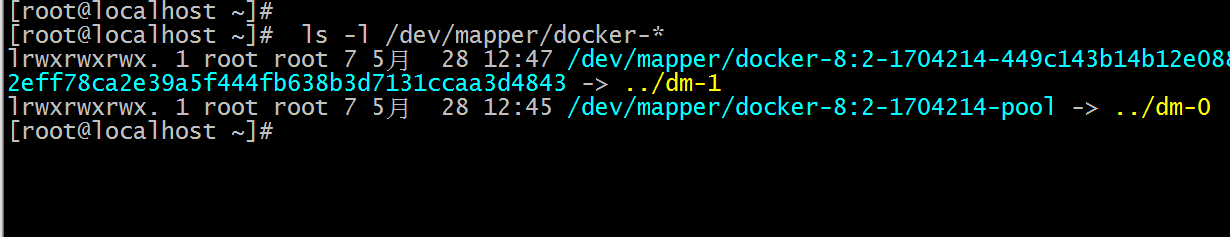
然后重新创建即可。

第二种扩容方法：

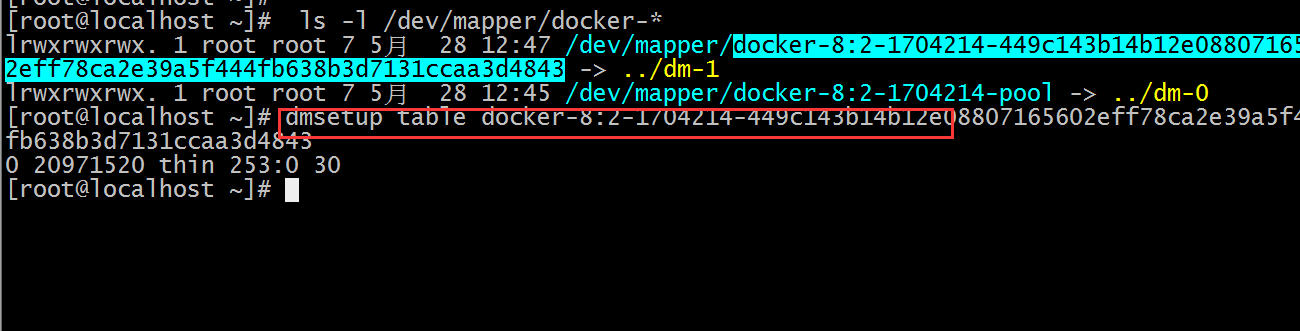
1. 查看原容器的磁盘空间大小



1. 查看mapper设备：



1. 查看卷信息表：



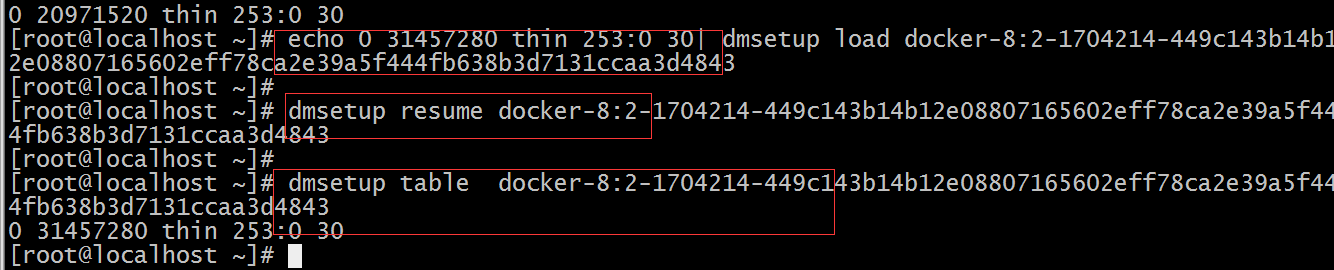
1. 根据要扩展的大小，计算需要多少扇区：

**第二个数字是设备的大小，表示有多少个512－bytes 的扇区. 这个值略高于 10GB 的大小。**

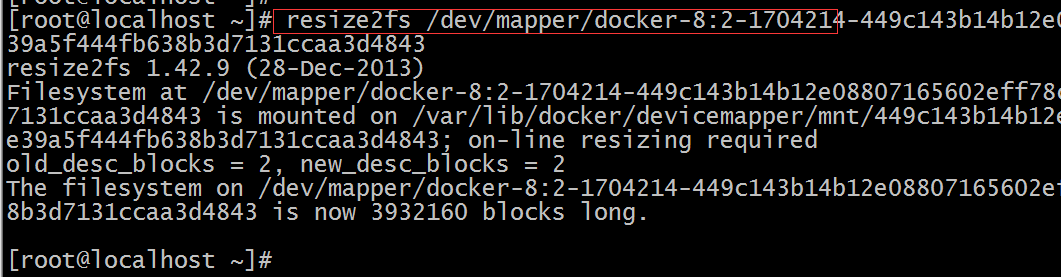
**我们来计算一下一个 15GB 的卷需要多少扇区，**

**$ echo $((15\*1024\*1024\*1024/512)) 31457280**

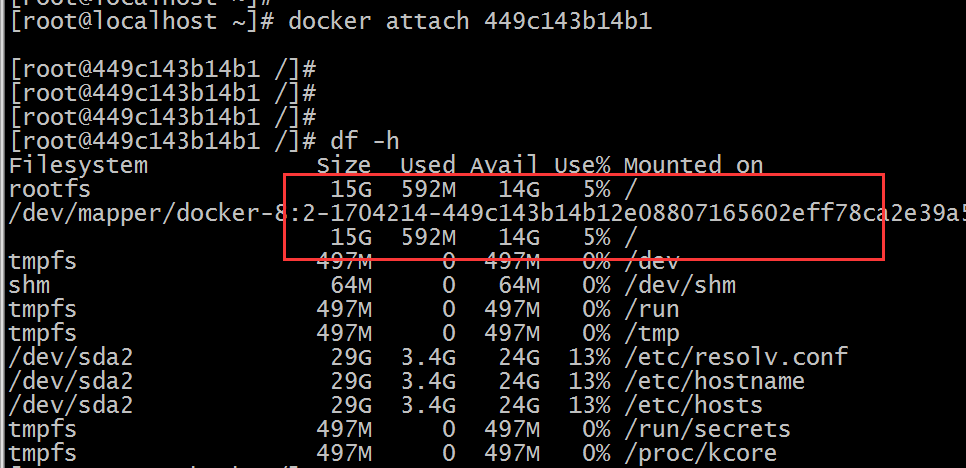
1. 修改卷信息表--激活--并且验证（红色3个部分）



## 修改文件系统大小：



## 最后验证磁盘大小：



## 成功扩容，当然了以上步骤也可以写成脚本，然后使用脚本批量扩容分区大小。

## Docker构建Mysql数据库服务器

## 在docker虚拟化中，如何来构建我们的MYSQL数据库服务器呢？答案很简单，我们可以是dockerfile来生成mysql镜像并启动运行即可。

## FROM centos:v1

## RUN groupadd -r mysql && useradd -r -g mysql mysql

## RUN install -y gcc zlib-devel gd-devel

## ENV MYSQL\_MAJOR 5.6

## ENV MYSQL\_VERSION 5.6.20

## RUN

## && curl -SL "http://dev.mysql.com/get/Downloads/MySQL-$MYSQL\_MAJOR/mysql-$MYSQL\_VERSION-linux-glibc2.5-x86\_64.tar.gz" -o mysql.tar.gz \

## && curl -SL "http://mysql.he.net/Downloads/MySQL-$MYSQL\_MAJOR/mysql-$MYSQL\_VERSION-linux-glibc2.5-x86\_64.tar.gz.asc" -o mysql.tar.gz.asc \

## && mkdir /usr/local/mysql \

## && tar -xzf mysql.tar.gz -C /usr/local/mysql \

## && rm mysql.tar.gz\* \

## ENV PATH $PATH:/usr/local/mysql/bin:/usr/local/mysql/scripts

## WORKDIR /usr/local/mysql

## VOLUME /var/lib/mysql

## EXPOSE 3306

## CMD ["mysqld", "--datadir=/var/lib/mysql", "--user=mysql"]

## Docker构建Nginx高性能web服务器

nginx [engine x]是[Igor Sysoev](http://sysoev.ru/en/)编写的一个HTTP和反向代理服务器，另外它也可以作为邮件代理服务器。 它已经在众多流量很大的俄罗斯网站上使用了很长时间，这些网站包括[Yandex](http://www.yandex.ru/)、[Mail.Ru](http://www.mail.ru/)、[VKontakte](http://www.vkontakte.ru/)，以及[Rambler](http://www.rambler.ru/)。

据Netcraft统计，在2012年8月份，[世界上最繁忙的网站中有11.48%](http://news.netcraft.com/archives/2012/08/02/august-2012-web-server-survey.html)使用Nginx作为其服务器或者代理服务器。目前互联网主流公司360、百度、新浪、腾讯、阿里等都在使用nginx作为自己的web服务器。

Nginx由内核和模块组成，其中，内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个location block（location是Nginx配置中的一个指令，用于URL匹配），而在这个location中所配置的每个指令将会启动不同的模块去完成相应的工作。

Nginx相对于Apache优点：

1. 高并发响应性能非常好，官方Nginx处理静态文件并发5w/s
2. 反向代理性能非常好。（可用于负载均衡）
3. 内存和cpu占用率低。（为Apache的1/5-1/10）
4. 功能较Apache少（常用功能均有）
5. 对php可使用cgi方式和fastcgi方式。

### Nginx WEB安装

首先需要安装pcre库，然后再安装Nginx：

#安装pcre支持rewrite库,也可以安装源码，注\*安装源码时，指定pcre路径为解压

源码的路径，而不是编译后的路径，否则会报错

（make[1]: \*\*\* [/usr/local/pcre/Makefile] Error 127 错误）

yum install pcre-devel pcre -y

#下载Nginx源码包

cd /usr/src ;wget -c <http://nginx.org/download/nginx-1.4.2.tar.gz>

#解压Nginx源码包

tar -xzf nginx-1.4.2.tar.gz

#进入解压目录，然后sed修改Nginx版本信息为WS

cd nginx-1.4.2 ; sed -i -e 's/1.4.2//g' -e 's/nginx\//WS/g' -e

's/"NGINX"/"WS"/g' src/core/nginx.h

#预编译Nginx

useradd www ;./configure --user=www --group=www --prefix=/usr/local/nginx --with-

http\_stub\_status\_module --with-http\_ssl\_module

#.configure预编译成功后，执行make命令进行编译

make

#make执行成功后，执行make install 正式安装

make install

#自此Nginx安装完毕

/usr/local/nginx/sbin/nginx -t 检查nginx配置文件是否正确，返回OK即正确。

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t

nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok

nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful

[root@localhost ~]#

然后启动nginx，/usr/local/nginx/sbin/nginx 回车即可。查看进程是否已启动：

[root@localhost ~]# ps -ef |grep nginx

nobody 5381 30285 0 May16 ? 00:04:31 nginx: worker process

root 30285 1 0 2014 ? 00:00:00 nginx: master process /usr/local/nginx/sbin/nginx

root 32260 32220 0 12:34 pts/0 00:00:00 grep nginx

[root@localhost ~]#

然后我们可以通过docker 的ip去访问即可。

## Docker构建LAMP架构配置

LNAMP(Linux+Nginx+Apache+Mysql+PHP)架构受到很多IT企业的青睐,取代了原来认为很好的LNMP(Linux+Nginx+Mysql+PHP)架构,那我们说LNAMP到底有什么优点呢,还得从Nginx和apache的优缺点说起。

Nginx处理静态文件能力很强,Apache处理动态文件很强而且很稳定,把二者综合在一块,性能提升很多倍。可能很多Linux SA在从事LNMP运维中，会发现PHP（FastCGI）模式会出现一些502错误的现象，这是因为Nginx+PHP（FastCGI）组合不稳定的原因造成的。

YUM 安装LAMP方式：

**yum install httpd httpd-tools mysql mysql-server mysql-devel php php-devel php-mysql -y**

* 源码安装LNAMP之Nginx

yum install prce-devel -y ;cd /usr/src ;wget <http://nginx.org/download/nginx-1.6.0.tar.gz> ;cd nginx-1.6.0 ;./configure -prefix=/usr/local/nginx && make &&make install

* 源码安装LNAMP之Apache

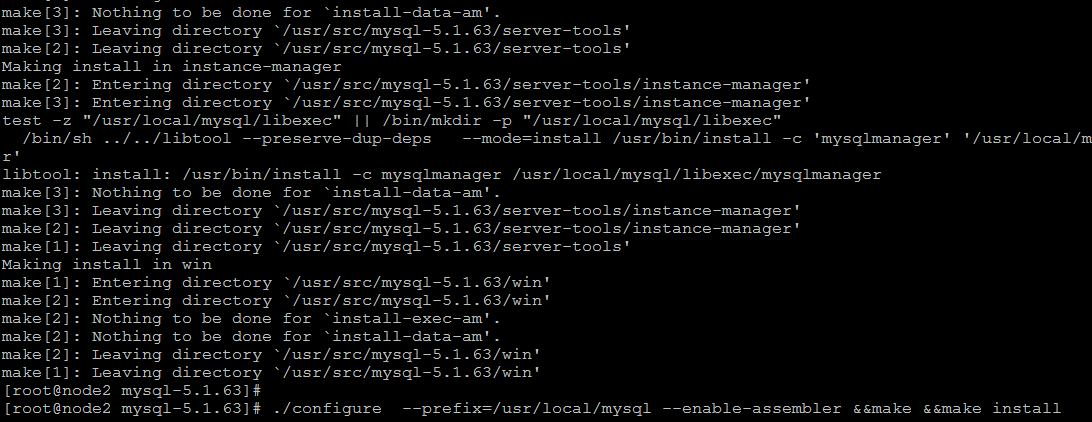
yum install apr-devel apr-util-devel –y；

cd /usr/src ; wget <http://mirror.bit.edu.cn/apache/httpd/httpd-2.2.27.tar.gz> ;tar xzf httpd-2.2.27.tar.gz ;cd httpd-2.2.27 ;./configure --prefix=/usr/local/apache --enable-so --enable-rewrite &&make &&make install

* 源码安装LNAMP之MySQL

cd /usr/src ;wget <http://downloads.mysql.com/archives/mysql-5.1/mysql-5.1.63.tar.gz> ;tar xzf mysql-5.1.63.tar.gz ;cd mysql-5.1.63 ;./configure --prefix=/usr/local/mysql --enable-assembler &&make &&make install

配置Mysql服务为系统服务：



cp /usr/local/mysql/share/mysql/my-medium.cnf /etc/my.cnf

cp /usr/local/mysql/share/mysql/mysql.server /etc/rc.d/init.d/mysqld

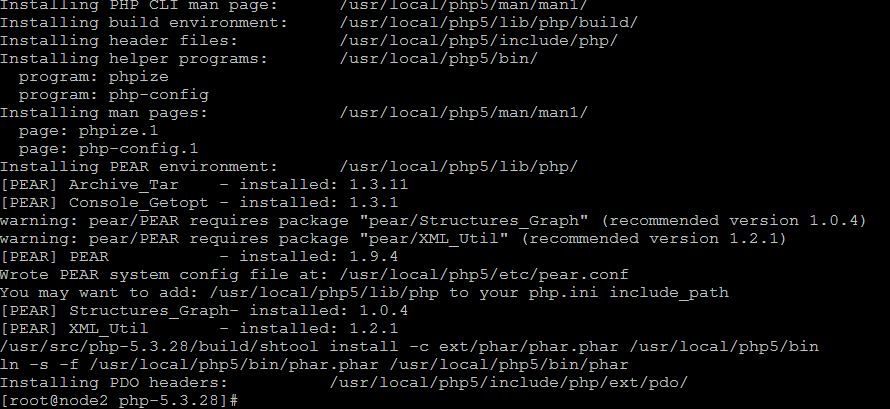
chkconfig --add mysqld   
chkconfig --level 345 mysqld on

cd /usr/local/mysql

useradd mysql  
chown -R  mysql.mysql /usr/local/mysql   
/usr/local/mysql/bin/mysql\_install\_db --user=mysql    
chown -R mysql  var   
/usr/local/mysql/bin/mysqld\_safe --user=mysql &

* 源码安装LNAMP之PHP

cd /usr/src ;wget <http://mirrors.sohu.com/php/php-5.3.28.tar.bz2> ;tar jxf php-5.3.28.tar.bz2 ;cd php-5.3.28 ;./configure --prefix=/usr/local/php5 --with-config-file-path=/usr/local/php/etc --with-apxs2=/usr/local/apache/bin/apxs --with-mysql=/usr/local/mysql/



* 源码安装Apache+PHP整合

整合apache+php环境，修改httpd.conf配置文件，然后加入如下语句：

LoadModule     php5\_module modules/libphp5.so （默认已存在）

AddType application/x-httpd-php .php

DirectoryIndex  index.php index.html (把index.php加入index.html之前)

然后在/usr/local/apache/htdocs目录下创建index.php测试页面，执行如下命令：

cat >>/usr/local/apache/htdocs/index.php <<EOF

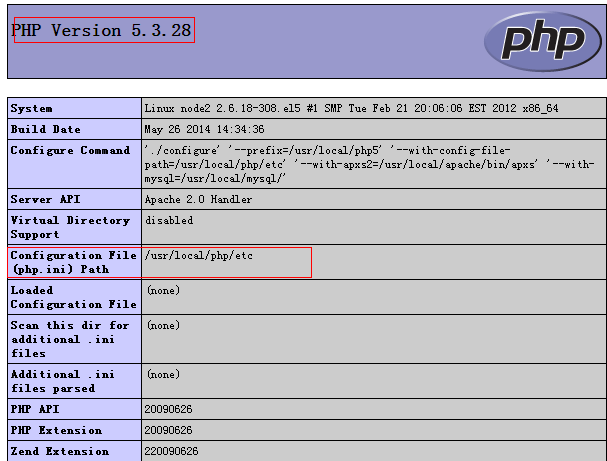
<?php

phpinfo();

?>

EOF

重新启动apache服务，通过IP访问界面如下图，即代表LAMP环境搭建成功。



* 源码安装DISCUZ论坛

下载discuz源码包文件，然后解压：

cd /usr/src ;wget <http://download.comsenz.com/DiscuzX/3.1/Discuz_X3.1_SC_UTF8.zip>

解压discuz程序包：unzip Discuz\_X3.1\_SC\_UTF8.zip -d /usr/local/apache/htdocs/

重命名程序文件：cd /usr/local/apache/htdocs/ ;mv upload/\* .

赋予discuz目录完全访问权限：cd /usr/local/apache/htdocs/ ;chmod 777 -R data/ uc\_server/ config/ uc\_client/

然后访问IP安装discuz论坛，如下图，选择“我同意”



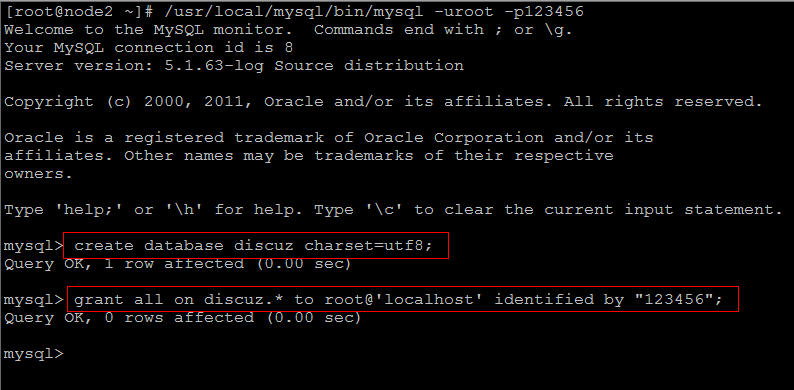
进入如下界面，数据库安装，如果不存在则需要新建数据库并授权。



数据库创建及授权命令如下：

create database discuz charset=utf8;

grant all on discuz.\* to root@'localhost' identified by "123456";



点击下一步，直至安装完成，进入等待已久的论坛画面：



## 自此LAMP环境整合并搭建成功，那如何使用Nginx来整合LAMP呢？

## Docker自动化部署脚本案例

## 自动安装并配置docker虚拟化及桥接网络，同时使用pipework这个软件来配置我们的容器IP。能够实现容器的一个简单的管理。

## #!/bin/bash

## #auto install docker and Create VM

## #by wugk 2016-01-05

## #Define PATH Varablies

## IPADDR=`ifconfig |grep "Bcast"|awk '{print $2}'|cut -d: -f2|grep "192.168"|head -1`

## GATEWAY=`route -n|grep "UG"|awk '{print $2}'|grep "192.168"|head -1`

## DOCKER\_IPADDR=$1

## IPADDR\_NET=`ifconfig |grep "Bcast"|awk '{print $2}'|cut -d: -f2|grep "192.168"|head -1|awk -F. '{print $1"."$2"."$3".""xxx"}'`

## NETWORK=(

## HWADDR=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $1}'`

## IPADDR=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $2}'`

## NETMASK=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $3}'`

## GATEWAY=`route -n|grep "UG"|awk '{print $2}'`

## )

## if [ -z "$1" -o -z "$2" -o -z "$3" -o -z "$4" ];then

## echo -e "\033[32m---------------------------------\033[0m"

## echo -e "\033[32mPlease exec $0 IPADDR CPU(C) MEM(G) DISK(G),example $0 $IPADDR\_NET 16 32 50\033[0m"

## exit 0

## fi

## CPU=`expr $2 - 1`

## if [ ! -e /usr/bin/bc ];then

## yum install bc -y >>/dev/null 2>&1

## fi

## MEM\_F=`echo $3 \\* 1024|bc`

## MEM=`printf "%.0f\n" $MEM\_F`

## DISK=$4

## USER=$5

## REMARK=$6

## ping $DOCKER\_IPADDR -c 1 >>/dev/null 2>&1

## if [ $? -eq 0 ];then

## echo -e "\033[32m---------------------------------\033[0m"

## echo -e "\033[32mThe IP address to be used,Please change other IP,exit.\033[0m"

## exit 0

## fi

## if [ ! -e /etc/init.d/docker ];then

## rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86\_64/epel-release-6-8.noarch.rpm

## yum install docker-io -y

## yum install device-mapper\* -y

## /etc/init.d/docker start

## if [ $? -ne 0 ];then

## echo "Docker install error ,please check."

## exit

## fi

## fi

## cd /etc/sysconfig/network-scripts/

## mkdir -p /data/backup/`date +%Y%m%d-%H%M`

## yes|cp ifcfg-eth\* /data/backup/`date +%Y%m%d-%H%M`/

## if

## [ -e /etc/sysconfig/network-scripts/ifcfg-br0 ];then

## echo

## else

## cat >ifcfg-eth0 <<EOF

## DEVICE=eth0

## BOOTPROTO=none

## ${NETWORK[0]}

## NM\_CONTROLLED=no

## ONBOOT=yes

## TYPE=Ethernet

## BRIDGE="br0"

## ${NETWORK[1]}

## ${NETWORK[2]}

## ${NETWORK[3]}

## USERCTL=no

## EOF

## cat >ifcfg-br0 <<EOF

## DEVICE="br0"

## BOOTPROTO=none

## ${NETWORK[0]}

## IPV6INIT=no

## NM\_CONTROLLED=no

## ONBOOT=yes

## TYPE="Bridge"

## ${NETWORK[1]}

## ${NETWORK[2]}

## ${NETWORK[3]}

## USERCTL=no

## EOF

## /etc/init.d/network restart

## fi

## echo 'Your can restart Ethernet Service: /etc/init.d/network restart !'

## echo '---------------------------------------------------------'

## cd -

## #######create docker container

## service docker status >>/dev/null

## if [ $? -ne 0 ];then

## /etc/init.d/docker restart

## fi

## NAME="Docker$$\_`echo $DOCKER\_IPADDR|awk -F"." '{print $(NF-1)"\_"$NF}'`"

## IMAGES=`docker images|grep -v "REPOSITORY"|grep -v "none"|head -1|awk '{print $1}'`

## CID=$(docker run -itd --cpuset-cpus=0-$CPU -m ${MEM}m --net=none --name=$NAME $IMAGES /bin/bash)

## if [ -z $IMAGES ];then

## echo "Plesae Download Docker Centos Images,you can to be use docker search centos,and docker pull centos6.5-ssh,exit 0"

## exit 0

## fi

## if [ ! -f /usr/local/bin/pipework ];then

## yum install wget unzip zip -y

## wget https://github.com/jpetazzo/pipework/archive/master.zip

## unzip master

## cp pipework-master/pipework /usr/local/bin/

## chmod +x /usr/local/bin/pipework

## rm -rf master

## fi

## ip netns >>/dev/null

## if [ $? -ne 0 ];then

## rpm -e iproute --nodeps

## rpm -ivh https://repos.fedorapeople.org/openstack/EOL/openstack-grizzly/epel-6/iproute-2.6.32-130.el6ost.netns.2.x86\_64.rpm

## fi

## pipework br0 $NAME $DOCKER\_IPADDR/24@$IPADDR

## docker ps -a |grep "$NAME"

## DEV=$(basename $(echo /dev/mapper/docker-\*-$CID))

## dmsetup table $DEV | sed "s/0 [0-9]\* thin/0 $((${DISK}\*1024\*1024\*1024/512)) thin/" | dmsetup load $DEV

## dmsetup resume $DEV

## resize2fs /dev/mapper/$DEV

## docker start $CID

## docker logs $CID

## LIST="docker\_vmlist.csv"

## if [ ! -e $LIST ];then

## echo "编号,容器ID,容器名称,CPU,内存,硬盘,容器IP,宿主机IP,使用人,备注" >$LIST

## fi

## ###################

## NUM=`cat docker\_vmlist.csv |grep -v CPU|tail -1|awk -F, '{print $1}'`

## if [[ $NUM -eq "" ]];then

## NUM="1"

## else

## NUM=`expr $NUM + 1`

## fi

## ##################

## echo -e "\033[32mCreate virtual client Successfully.\n$NUM `echo $CID|cut -b 1-12` $NAME $2C ${MEM}M ${DISK}G $DOCKER\_IPADDR $IPADDR $USER $REMARK\033[0m"

## if [ -z $USER ];then

## USER="NULL"

## REMARK="NULL"

## fi

## echo $NUM,`echo $CID|cut -b 1-12`,$NAME,${2}C,${MEM}M,${DISK}G,$DOCKER\_IPADDR,$IPADDR,$USER,$REMARK >>$LIST

## rm -rf docker\_vmlist\_\*

## iconv -c -f utf-8 -t gb2312 docker\_vmlist.csv -o docker\_vmlist\_`date +%H%M`.csv

## Docker应用常见故障解决汇总

## 在我们日常使用docker的时候，经常会遇到很多问题，今天我们来跟大家一起学习，常见使用docker的时候会遇到什么问题，如何去避免和解决呢？

## 报错信息如下：