# 1、docker

## 1.1 docker的简介

Docker是一个开源的容器引擎， 它有助于更快地交付应用。 Docker可将应用程序和基础设施层隔离， 并且能将基础设施当作程序一样进行管理。 使用 Docker可更快地打包、 测试以及部署应用程序，并可以缩短从编写到部署运行代码的周期。

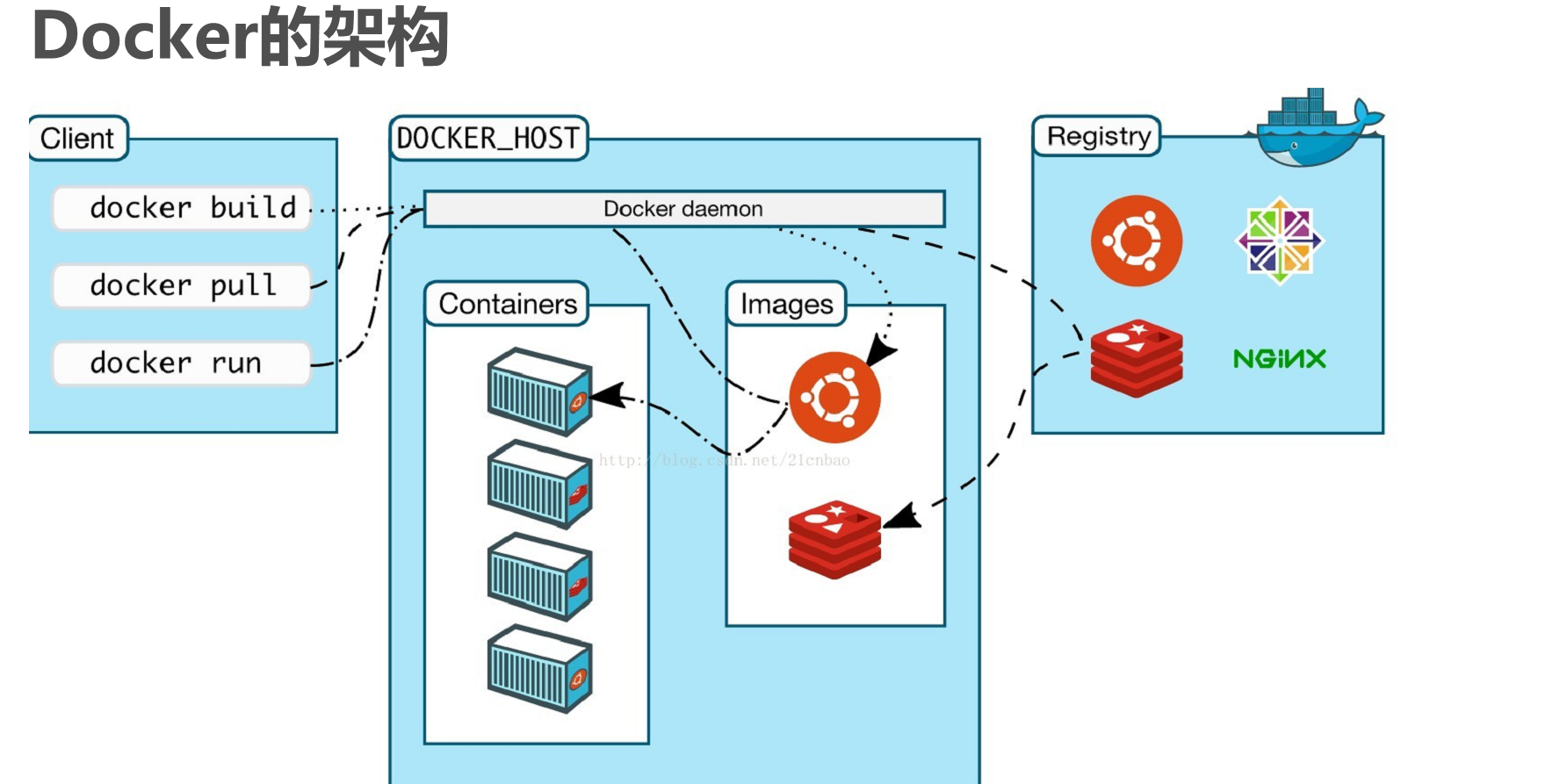
### 1.1 docker 的优点

1、简化程序  
Docker 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 机器上， 便可以实现虚拟化。 Docker改变了虚拟化的方式， 使开发者可以直接将自己的成果放入Docker中进行管理。 方便快捷已经是 Docker的最大优势， 过去需要用数天乃至数周的任务，在Docker容器的处理下，只需要数秒就能完成。

2、避免选择恐惧症  
如果你有选择恐惧症， 还是资深患者。 Docker 帮你 打包你的纠结！ 比如 Docker 镜像； Docker 镜像中包含了运行环境和配置， 所以 Docker 可以简化部署多种应用实例工作。 比如 Web 应用、 后台应用、 数据库应用、 大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、节省开支  
一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件， Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

## 1.2 docker的架构



### Docker daemon（ Docker守护进程）

Docker daemon是一个运行在宿主机（ DOCKER-HOST） 的后台进程。 可通过 Docker客户端与之通信。Client（ Docker客户端）Docker客户端是 Docker的用户界面， 它可以接受用户命令和配置标识，并与 Docker daemon通信。图中，docker build等都是 Docker的相关命令。

### Images（ Docker镜像）

Docker镜像是一个只读模板， 它包含创建 Docker容器的说明。 它和系统安装光盘有点像， 使用系统安装光盘可以安装系统，同理，使用Docker镜像可以运行 Docker镜像中的程序。

### Container（容器）

容器是镜像的可运行实例。 镜像和容器的关系有点类似于面向对象中， 类和对象的关系。 可通过 Docker API或者 CLI命令来启停、 移动、 删除容器。

### Registry

Docker Registry是一个集中存储与分发镜像的服务。 构建完 Docker镜像后， 就可在当前宿主机上运行。 但如果想要在其他机器上运行这个镜像，就需要手动复制。此时可借助 Docker Registry来避免镜像的手动复制。一个 Docker Registry可包含多个 Docker仓库，每个仓库可包含多个镜像标签，每个标签对应一个 Docker镜像。 这跟Maven的仓库有点类似， 如果把 Docker Registry比作 Maven仓库的话，那么 Docker仓库就可理解为某jar包的路径，而镜像标签则可理解为jar包的版本号。

## 1.3 docker的安装与常用命令

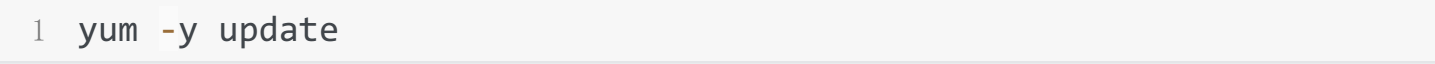
### 3.1 docker安装

Docker 是一个开源的商业产品，有两个版本：社区版（Community Edition，缩写为 CE）和企业版（Enterprise Edition，缩写为 EE）。企业版包含了一些收费服务，个人开发者一般用不到。 下面的介绍都针对社区版。  
Docker CE 的安装请参考官方文档， 我们这里以CentOS为例：  
1、Docker 要求 CentOS 系统的内核版本高于 3.1 0  
通过 uname -r 命令查看你当前的内核版本



2、使用 root 权限登录 Centos。升级所有包同时也升级软件和系统内内核

yum -y upgrade 只升级所有的包，不升级软件与系统内核



3、卸载旧版本(如果安装过旧版本的话)

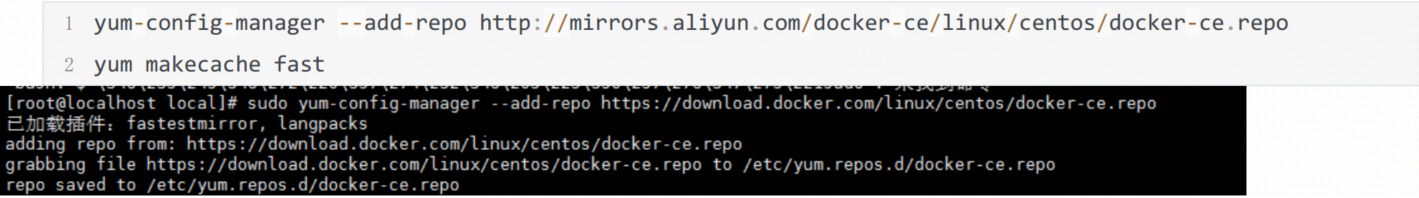


4、安装需要的软件包， yum-util 提供yum-config-manager功能， 另外两个是devicemapper驱动依赖的



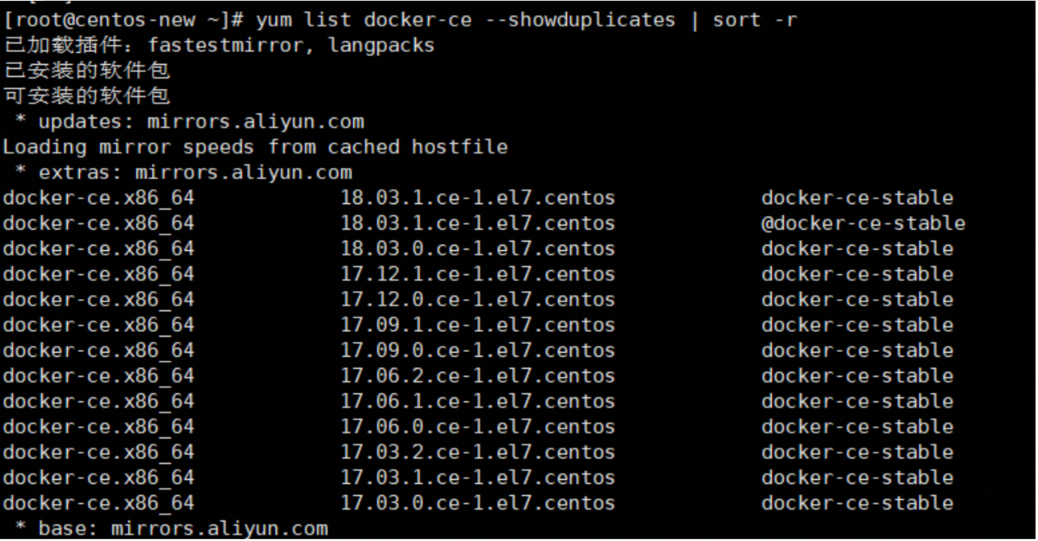
5、设置yum源，并更新yum的包索引

yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo



6、可以查看所有仓库中所有docker版本，并选择特定版本安装

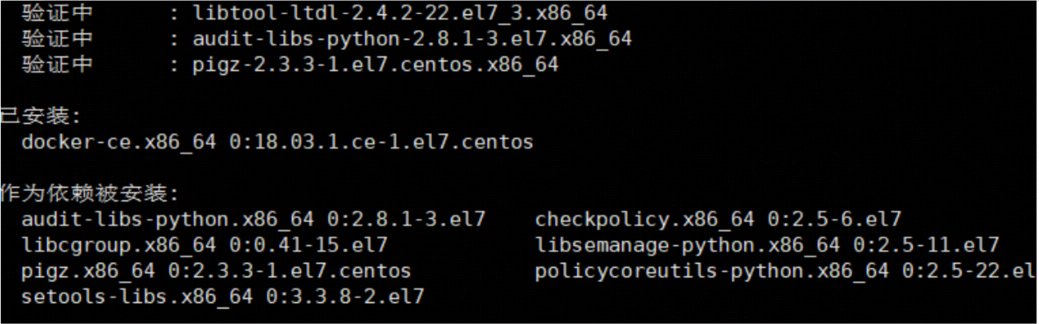
yum list docker -ce --showduplicates | sort -r



7、安装docker





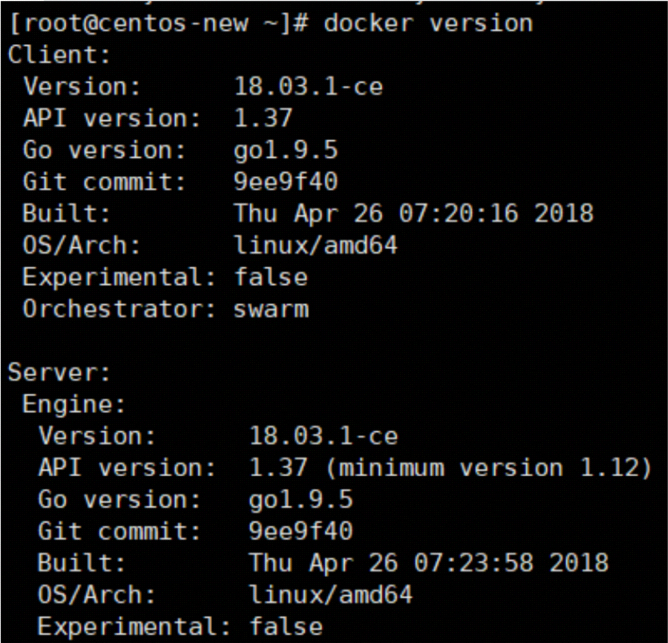


8、启动并加入开机启动



9、验证安装是否成功(有client和service两部分表示docker安装启动都成功了)





1 0、卸载docker



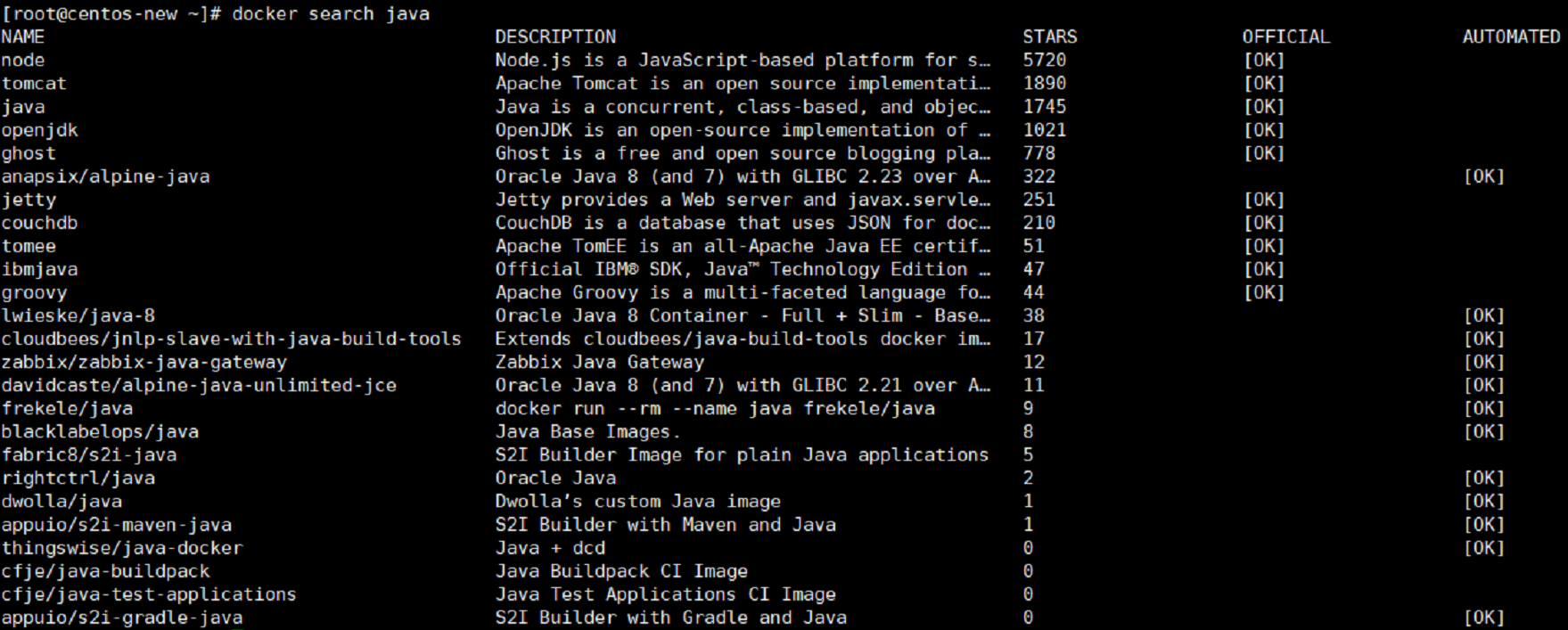
### 3.2 docker的常用命令

#### 2.1 镜像相关命令

**搜索镜像**：

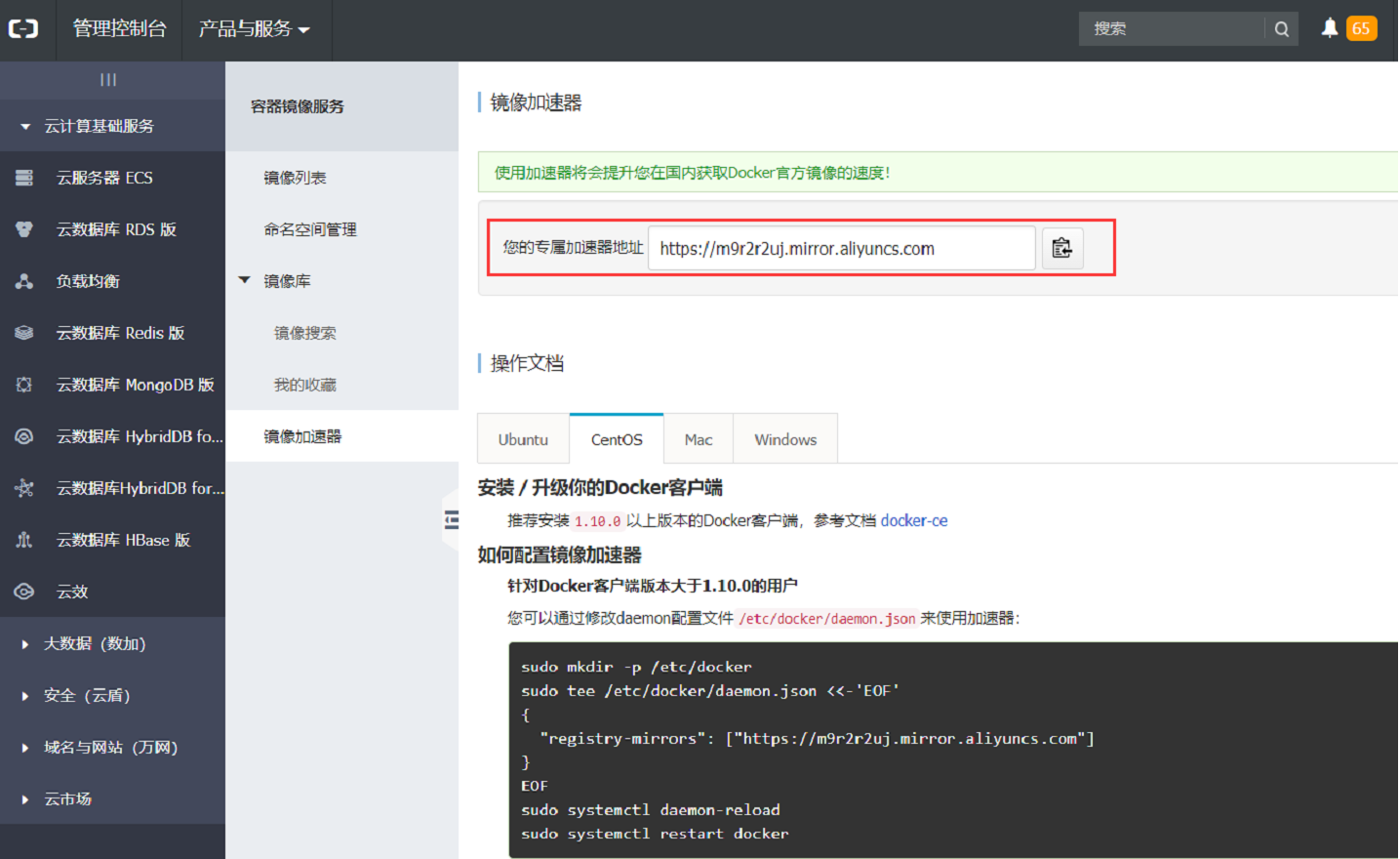
可使用 docker search命令搜索存放在 Docker Hub中的镜像。 执行该命令后， Docker就会在Docker Hub中搜索含有 java这个关键词的镜像仓库。





以上列表包含五列， 含义如下：  
- NAME:镜像仓库名称。  
- DESCRIPTION:镜像仓库描述。  
- STARS： 镜像仓库收藏数， 表示该镜像仓库的受欢迎程度， 类似于 GitHub的 stars0  
- OFFICAL:表示是否为官方仓库， 该列标记为[0K]的镜像均由各软件的官方项目 组创建和维护。  
- AUTOMATED： 表示是否是自动构建的镜像仓库。  
注意： 如果执行搜索命令报错， 需要配置镜像加速器

Error response from daemon: Get https://index.docker.io/v1 /search?q=java: read tcp 52.200.1 32.201 :443: i/o timeout  
我们可以借助阿里云的镜像加速器， 登录阿里云(https://cr.console.aliyun.com/#/accelerator)  
可以看到镜像加速地址如下图：

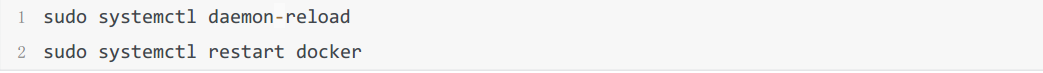




查看有没有 daemon.json。这是docker默认的配置文件。  
如果没有新建，如果有，则修改。



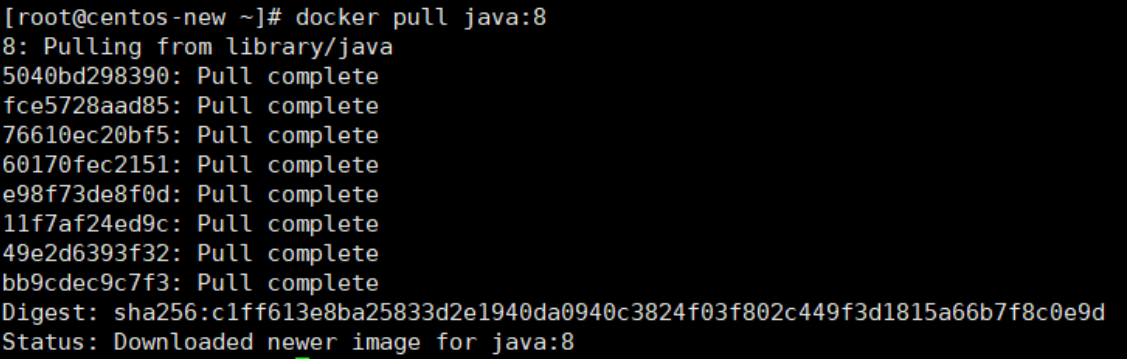
保存退出。  
重启docker服务



**下载镜像：**

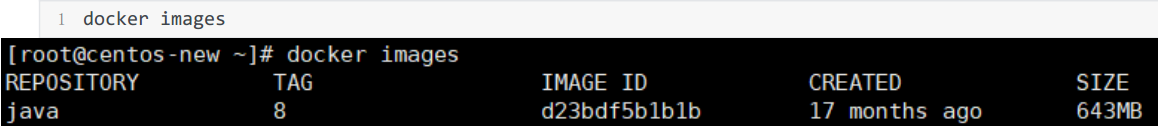
使用命令docker pull命令即可从 Docker Registry上下载镜像， 执行该命令后， Docker会从 Docker Hub中的 java仓库下载最  
新版本的 Java镜像。 如果要下载指定版本则在java后面加冒号指定版本， 例如： docker pull java:8





**列出镜像**

使用 docker images命令即可列出已下载的镜像



以上列表含义如下  
- REPOSITORY： 镜像所属仓库名称。  
- TAG:镜像标签。 默认是 latest,表示最新。  
- IMAGE ID： 镜像 ID， 表示镜像唯一标识。  
- CREATED： 镜像创建时间。  
- SIZE: 镜像大小

**删除本地镜像**

使用 docker rmi命令即可删除指定镜像， 强制删除加 –f

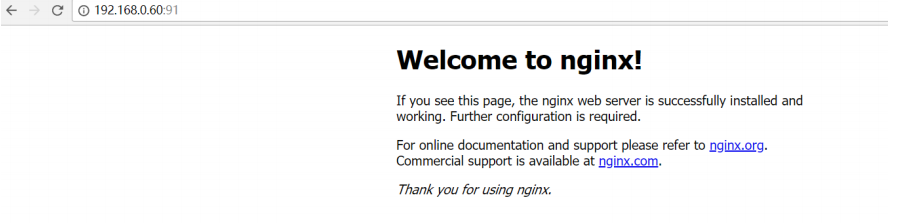


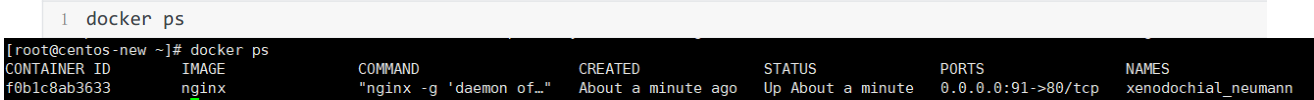
删除所有镜像



#### 2.2 容器相关命令

1、 新建并启动容器  
使用以下docker run命令即可新建并启动一个容器，该命令是最常用的命令，它有很多选项，下面将列举一些常用的选项。  
-d选项：表示后台运行  
-P选项：随机端口映射  
-p选项：指定端口映射，有以下四种格式。  
-- ip:hostPort:containerPort  
-- ip::containerPort  
-- hostPort:containerPort  
-- containerPort  
--net选项： 指定网络模式， 该选项有以下可选参数：  
--net=bridge:默认选项， 表示连接到默认的网桥。生产基本使用此方式，容器较安全  
--net=host:容器使用宿主机的网络。  
--net=container:NAME-or-ID： 告诉 Docker让新建的容器使用已有容器的网络配置。  
--net=none： 不配置该容器的网络， 用户可自定义网络配置。  
这样就能启动一个 Nginx容器。在本例中，为docker run添加了两个参数，含义如下：  
-d 后台运行  
-p 宿主机端口:容器端口 #开放容器端口到宿主机端口  
访问 http://Docker宿主机 IP:91 /， 将会看到nginx的主界面如下：

  
需要注意的是， 使用 docker run命令创建容器时， 会先检查本地是否存在指定镜像。 如果本地不存在该名称的镜像， Docker就会自动从 Docker Hub下载镜像并启动一个 Docker容器

2、列出容器  
用 docker ps命令即可列出运行中的容器  
  
如需列出所有容器（包括已停止的容器），可使用-a参数。该列表包含了7列，含义如下  
- CONTAINER\_ID：表示容器 ID。  
- IMAGE:表示镜像名称。  
- COMMAND：表示启动容器时运行的命令。  
- CREATED：表示容器的创建时间。  
- STATUS： 表示容器运行的状态。 UP表示运行中， Exited表示已停止。  
- PORTS:表示容器对外的端口号。  
- NAMES:表示容器名称。 该名称默认由 Docker自动生成， 也可使用 docker run命令的--name选项自行指定。

3、停止容器  
使用 docker stop命令，即可停止容器  
其中f0b1 c8ab3633是容器 ID,当然也可使用 docker stop容器名称来停止指定容器

4、强制停止容器  
可使用 docker kill命令发送 SIGKILL信号来强制停止容器  


5、启动已停止的容器  
使用docker run命令， 即可新建并启动一个容器。 对于已停止的容器， 可使用 docker start命令来启动  
6、 查看容器所有信息  
7、 查看容器日志  
8、 查看容器里的进程  
9、容器与宿主机相互复制文件，操作均在宿主机内执行  
 从容器里面拷文件到宿主机：  
从宿主机拷文件到容器里面：  


10、进入容器  
使用docker exec命令用于进入一个正在运行的docker容器。 如果docker run命令运行容器的时候， 没有使用-it参数， 就要用这个命令进入容器。 一旦进入了容器， 就可以在容器的 Shell 执行命令了，退出容器直接使用exit或者Ctrl+P+Q  
11 、容器内安装vim、ping、ifconfig等指令  
1 apt‐get update  
2 apt‐get install vim #安装vim  
3 apt‐get install iputils‐ping #安装ping  
4 apt‐get install net‐tools #安装ifconfig

12、删除容器  
使用 docker rm命令即可删除指定容器  
该命令只能删除已停止的容器， 如需删除正在运行的容器， 可使用-f参数  
强制删除所有容器  


#### 2.3 构建docker镜像

Dockerfile是一个文本文件，其中包含了若干条指令， 指令描述了构建镜像的细节  
先来编写一个最简单的Dockerfile，以前文下载的Nginx镜像为例， 来编写一个Dockerfile修改该Nginx镜像的首页  
1 、新建一个空文件夹docker-demo，在里面再新建文件夹app， 在app目录下新建一个名为Dockerfile的文件，在里面增加如下内容



该Dockerfile非常简单，其中的FROM、RUN都是Dockerfile的指令。FROM指令用于指定基础镜像，RUN指令用于执行命令。  
2、 在Dockerfile所在路径执行以下命令构建镜像：

如果文件名不为Dockerfile，则构建时需要指定具体的Dockerfile文件

docker build -f/etc/docker/docker-demo/dockerfileNg -t nginx:kevin .



其中，-t指定镜像名字，命令最后的点（.）表示Dockerfile文件所在路径  
3、执行以下命令，即可使用该镜像启动一个 Docker容器



4、 访问 http://Docker宿主机IP:92/，可看到下图所示界面

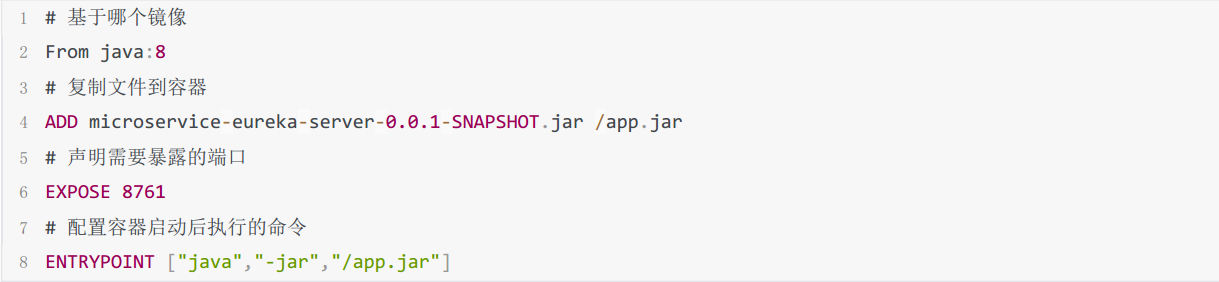






注意： RUN命令在 image 文件的构建阶段执行， 执行结果都会打包进入 image 文件； CMD命令则是在容器启动后执行。 另外， 一个 Dockerfile 可以包含多个RUN命令， 但是只能有一个CMD命令。  
注意， 指定了CMD命令以后， docker container run命令就不能附加命令了（比如前面的/bin/bash） ， 否则它会覆盖CMD命令。  
使用Dockerfile构建微服务镜像  
以项目 05-ms-eureka-server为例， 将该微服务的可运行jar包构建成docker镜像  
1、将jar包上传linux服务器/usr/local/docker-app/docker-demo/app/eureka目 录， 在jar包所在目录创建名为Dockerfile的文件

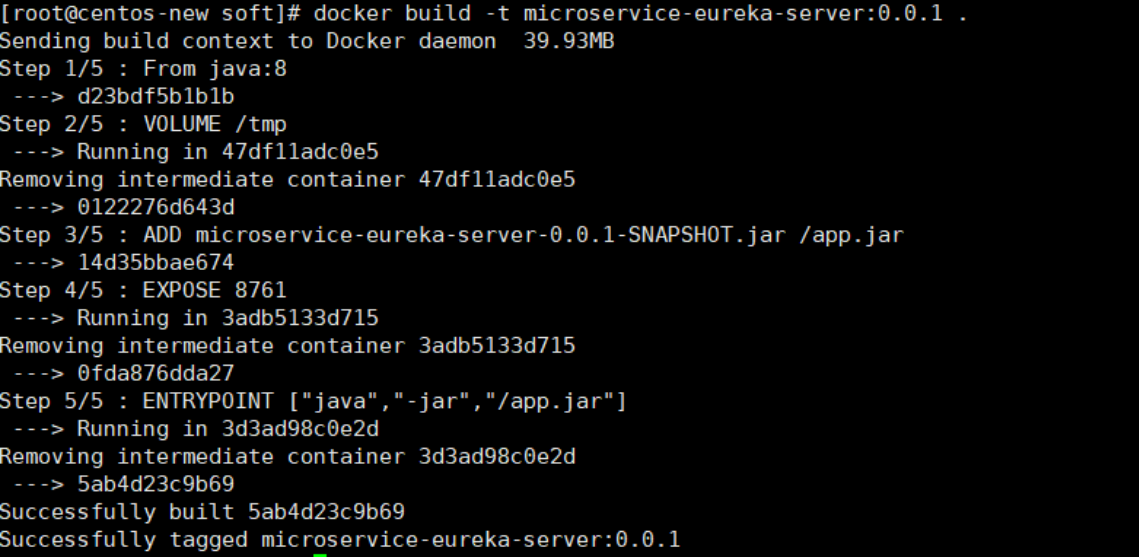
2、在Dockerfile中添加以下内容，暴露的端口号必须与项目中的端口号一致



3、 使用docker build命令构建镜像



# 格式：docker build -t 镜像名称:标签 Dockerfile的相对位置  
在这里，使用-t选项指定了镜像的标签。执行该命令后，终端将会输出如下的内容



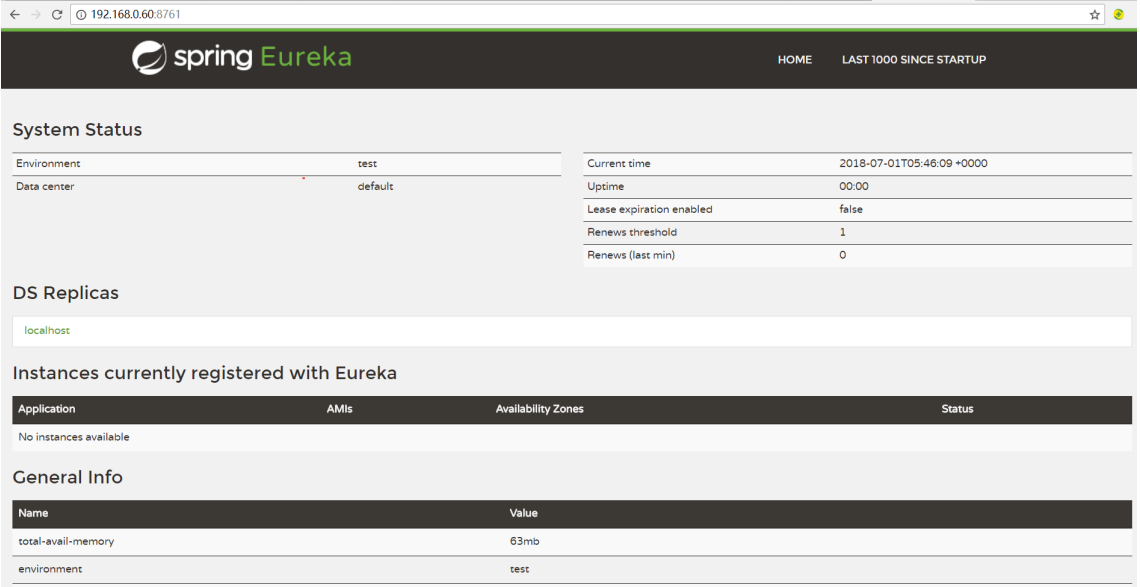
4、启动镜像，加-d可在后台启动



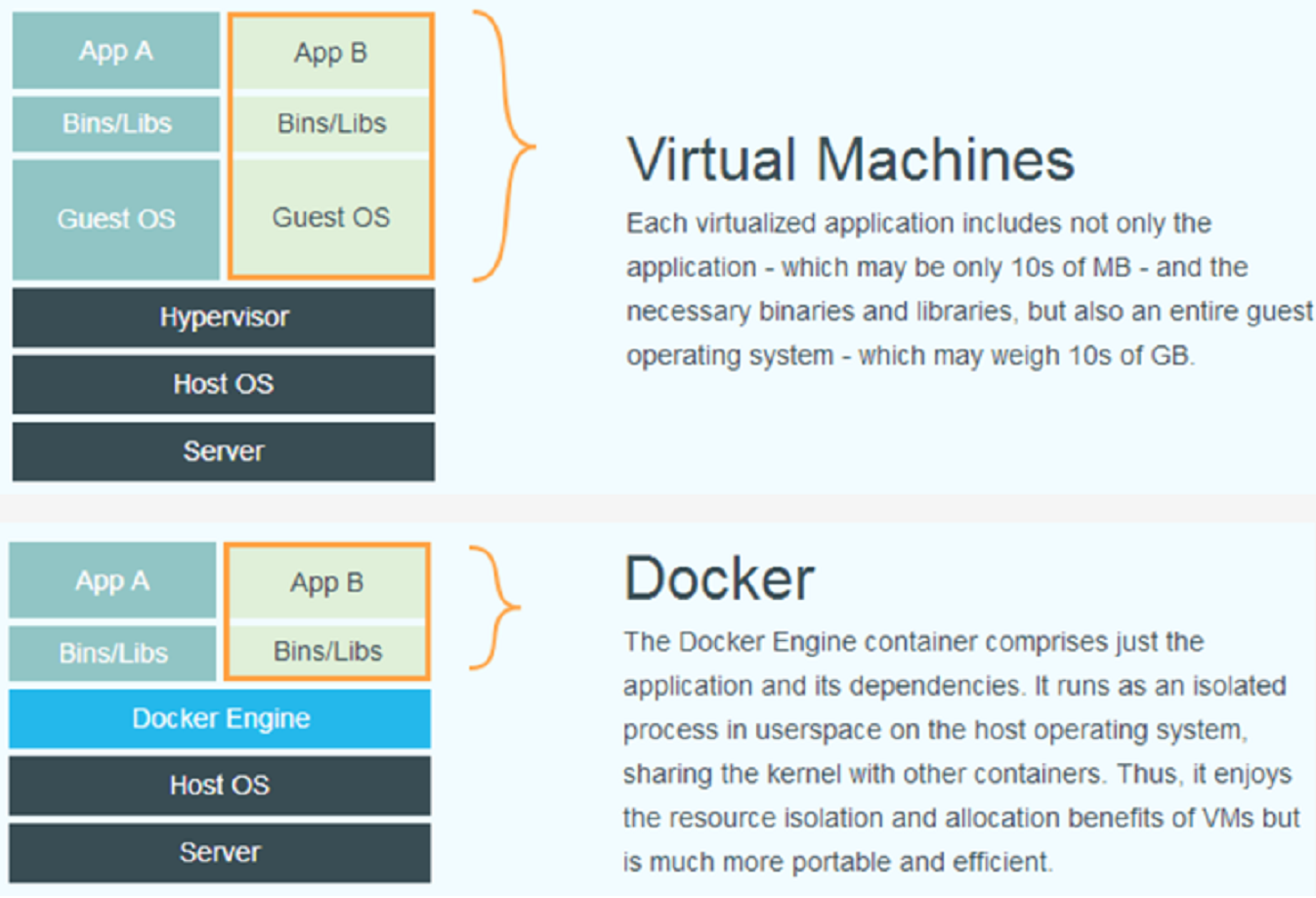
使用 -v 可以挂载一个主机上的目录到容器的目录



5、 访问http://Docker宿主机IP:8761 /， 可正常显示Eureka Server首页

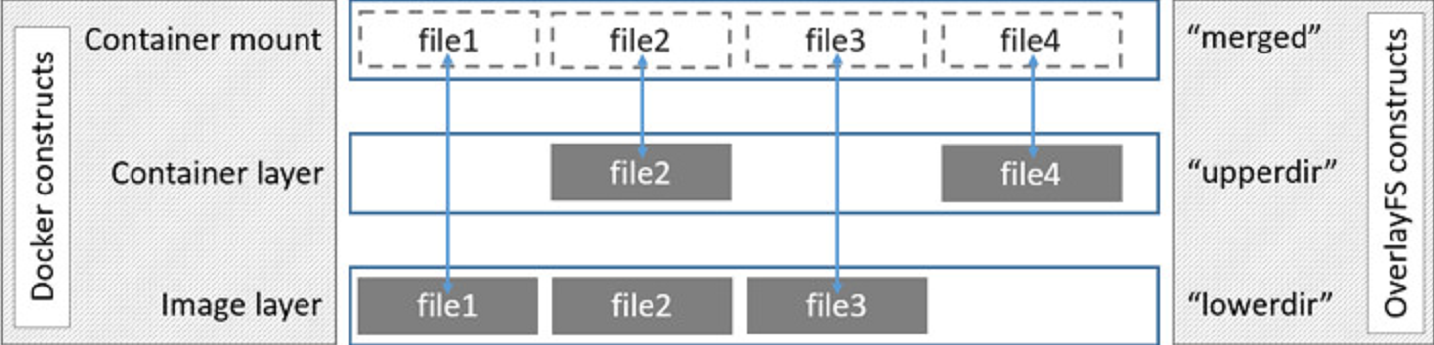


## 1.4 docker虚拟化技术的原理



传统虚拟化和容器技术结构比较：传统虚拟化技术是在硬件层面实现虚拟化， 增加了系统调用链路的环节， 有性能损耗；容器虚拟化技术以共享宿主机Kernel的方式实现，几乎没有性能损耗。  
docker利用的是宿主机的内核,而不需要Guest OS。 因此,当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。 避免了寻址、加载操作系统内核这些比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载Guest OS,这个新建过程是分钟级别的。 而docker由于直接利用宿主机的操作系统,则省略了这个过程,因此新建一个docker容器只需要几秒钟。

**Docker是如何将机器的资源进行隔离的？**  
答案是联合文件系统，常见的有AUFS、Overlay、devicemapper、BTRFS和ZFS等。  
以Overlay2举例说明，Overlay2的架构图如下：



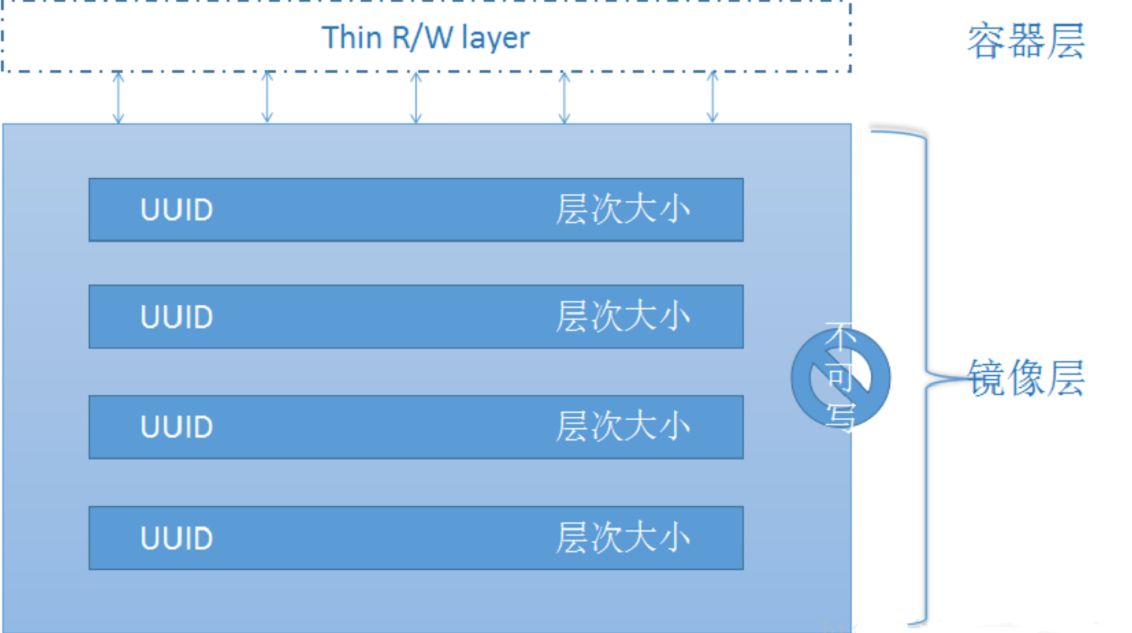
原理：overlayfs在linux主机上只有两层，一个目录在下层， 用来保存镜像(docker)， 另外一个目录在上层， 用来存储容器信息。 在overlayfs中， 底层的目录叫做lowerdir， 顶层的目录称之为upperdir， 对外提供统一的文件系统为merged。 当需要修改一个文件时， 使用COW(Copy-on-write)将文件从只读的Lower复制到可写的Upper进行修改， 结果也保存在Upper层。 在Docker中，底下的只读层就是image，可写层就是Container。  
**写时复制 (CoW) 技术详解**所有驱动都用到的技术—写时复制，Cow全称copy-on-write，表示只是在需要写时才去复制，这个是针对已有文件的修改场景。 比如基于一个image启动多个Container， 如果每个Container都去分配一个image一样的文件系统，那么将会占用大量的磁盘空间。 而CoW技术可以让所有的容器共享image的文件系统， 所有数据都从image中读取， 只有当要对文件进行写操作时， 才从image里把要写的文件复制到自己的文件系统进行修改。 所以无论有多少个容器共享一个image， 所做的写操作都是对从image中复制到自己的文件系统的副本上进行，并不会修改image的源文件，且多个容器操作同一个文件，会在每个容器的文件系统里生成一个副本，每个容器修改的都是自己的副本，互相隔离，互不影响。使用CoW可以有效的提高磁盘的利用率。所以容器占用的空间是很少的。

查看容器占用磁盘大小指令：

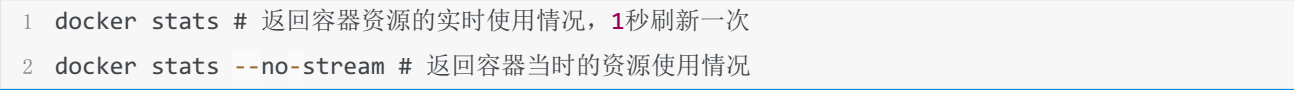


**用时分配 （allocate-on-demand）**  
用时分配是针对原本没有这个文件的场景， 只有在要新写入一个文件时才分配空间， 这样可以提高存储资源的利用率。 比如启动一个容器， 并不会因为这个容器分配一些磁盘空间， 而是当有新文件写入时，才按需分配新空间。

**docker中的镜像分层技术的原理是什么呢？**  
docker使用共享技术减少镜像存储空间，所有镜像层和容器层都保存在宿主机的文件系统/var/lib/docker/中，由存储驱动进行管理， 尽管存储方式不尽相同， 但在所有版本的Docker中都可以共享镜像层。在下载镜像时， Docker Daemon会检查镜像中的镜像层与宿主机文件系统中的镜像层进行对比， 如果存在则不下载， 只下载不存在的镜像层， 这样可以非常节约存储空间。



最后附一个查看容器资源使用情况的指令：





默认情况下，stats 命令会每隔 1 秒钟刷新一次输出的内容直到你按下 ctrl + c。下面是输出的主要内容：  
[CONTAINER]： 以短格式显示容器的 ID。  
[CPU %]： CPU 的使用情况。  
[MEM USAGE / LIMIT]： 当前使用的内存和最大可以使用的内存。  
[MEM %]： 以百分比的形式显示内存使用情况。  
[NET I/O]： 网络 I/O 数据。  
[BLOCK I/O]： 磁盘 I/O 数据。  
[PIDS]： PID 号。

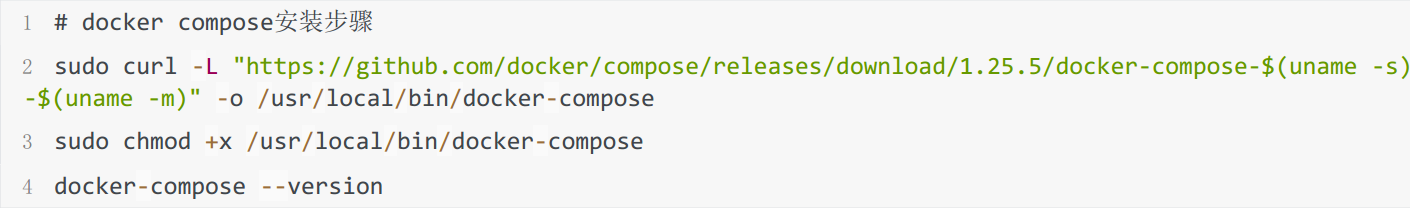
# 2、docker compose

使用微服务架构的应用系统一般包含若干个微服务， 每个微服务一般都会部署多个实例。 如果每个微服务都要手动启停， 那么效率之低、 维护量之大可想而知。 使用 Docker Compose来轻松、 高效地管理容器。 为了简单起见将 Docker Compose简称为 Compose。  
Compose 是一个用于定义和运行多容器的Docker应用的工具。 使用Compose， 你可以在一个配置文件（yaml格式）中配置你应用的服务， 然后使用一个命令， 即可创建并启动配置中引用的所有服务。

## 2.1 docker compose的简单使用

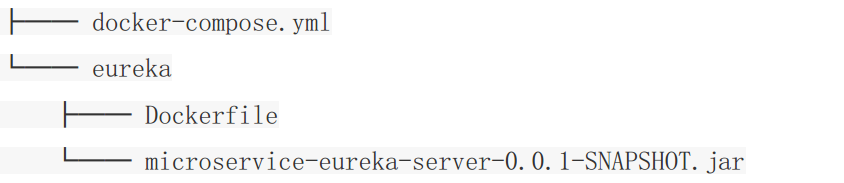
### 1.1 docker compose的安装

Compose的安装有多种方式， 例如通过shell安装、 通过pip安装、 以及将compose作为容器安装等等。 本文讲解通过shell安装的方式。 其他安装方式如有兴趣， 可以查看Docker的官方文档：  
<https://docs.docker.com/compose/install/>



### 1.2 docker compose的简单示例

Compose的使用非常简单，只需要编写一个docker-compose.yml，然后使用docker-compose 命令操作即可。dockercompose.yml描述了容器的配置，而docker-compose命令描述了对容器的操作。我们首先通过一个示例快速入门：  
还记得上节课， 我们使用Dockerfile为项目 microservice-eureka-server构建Docker镜像吗？我们还以此项目为例测试我们在microservice-eureka-server-0. 0. 1-SNAPSHOT. jar所在目录的上一级目录，创建docker-compose. yml 文件。  
目录树结构如下：

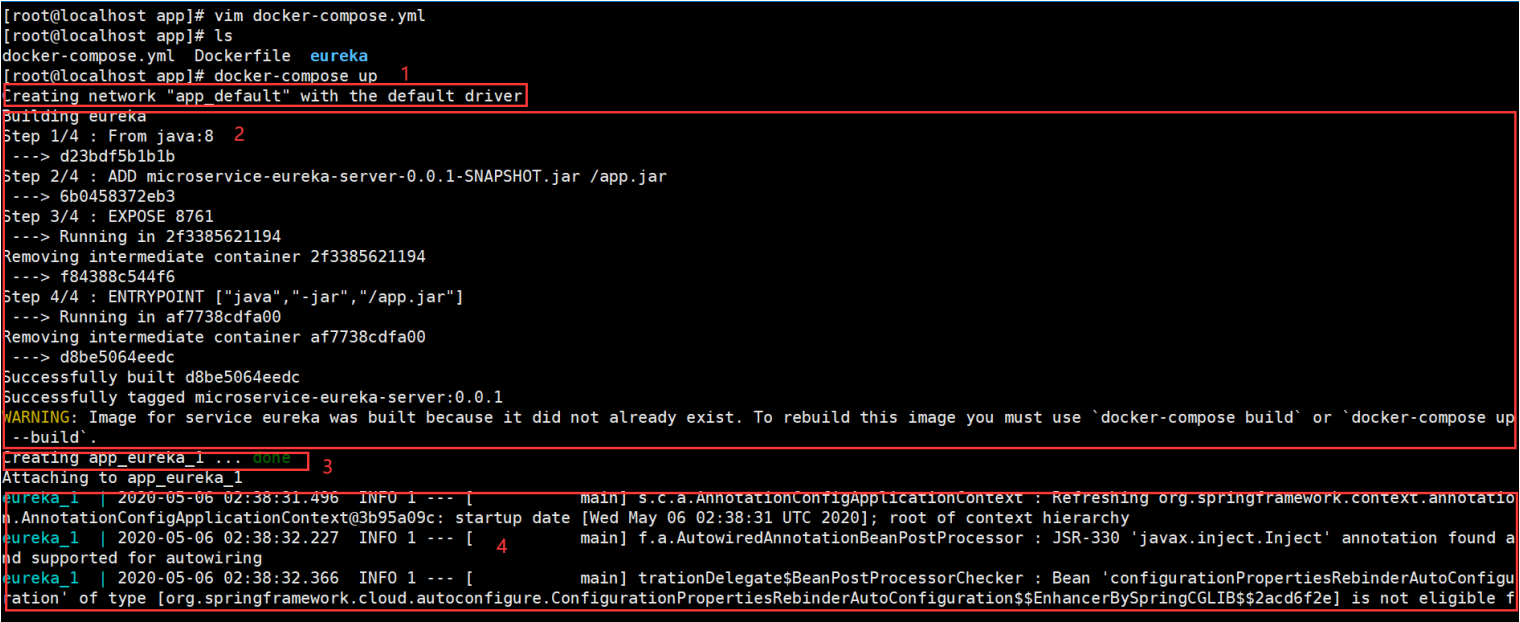


docker-compose.yml中添加以下内容



在docker-compose.yml所在目录下执行





如上图，compose启动会做几件事：  
1、 创建一个默认的网络app\_default， 默认以compose所在文件目 录名加"\_default"命名， compose内的所有容器都会加入此网络，可以相互用服务名访问。  
2、如果镜像 microservice-eureka-server:0.0.1 不存在先构建镜像，如果镜像存在则不构建，加上 --build 参数可以强制先构建镜像，如果镜像之前构建过且构建文件没有变化或构建的内容没有变化，就算加上 --build 参数也不会重新构建。  
3、根据构建的镜像创建一个名称叫 app\_eureka\_1 的容器。  
4、启动容器。访问： http: //宿主机IP: 8761/ ，发现可以正常访问eureka主页。

### 1.3 docker compose管理容器的结构

Docker Compose将所管理的容器分为三层，分别是工程（ project），服务（service）以及容器（ container）。  
Docker Compose运行目录下的所有文件（ docker-compose.yml、extends文件或环境变量文件等） 组成一个工程（默认为 docker-compose.yml所在目录的目录名称）。一个工程可包含多个服务， 每个服务中定义了容器运行的镜像、参数和依赖，一个服务可包括多个容器实例。  
上节示例里工程名称是 docker-compose.yml 所在的目录名。该工程包含了1个服务，服务名称是 eureka，执行docker-compose up时， 启动了eureka服务的1个容器实例。  
同一个docker compose内部的容器之间可以用服务名相互访问，服务名就相当于hostname，可以直接 ping 服务  
名，得到的就是服务对应容器的ip，如果服务做了扩容， 一个服务对应了多个容器，则ping服务名会轮询访问服务对应的每台容器ip，docker底层用了LVS等技术帮我们实现这个负载均衡。

### 1.4 docker compose yml文件的常用命令

**image**  
指定镜像名称或者镜像id，如果该镜像在本地不存在，Compose会尝试pull下来。  
示例：  
image: java

**build**  
指定Dockerfile文件的路径。可以是一个路径，例如：  
build: . /dir  
也可以是一个对象，用以指定Dockerfile和参数，例如：  
build:  
context: . /dir  
dockerfile: Dockerfile-alternate  
args:  
buildno: 1

**command**  
覆盖容器启动后默认执行的命令。  
示例：  
command: bundle exec thin -p 3000  
也可以是一个list， 类似于Dockerfile总的CMD指令，格式如下：  
command: [bundle, exec, thin, -p, 3000]

**links**显示链接到其他服务中的容器。可以指定服务名称和链接的别名使用SERVICE: ALIAS 的形式，或者只指定服务名称，示例：  
web:  
links:  
- db  
- db: database  
- redis

**external\_links**表示链接到docker­compose.yml外部的容器，甚至并非Compose管理的容器，特别是对于那些提供共享容器或共同服务。 格式跟links类似

示例：  
external\_links:  
- redis\_1  
- project\_db\_1: mysql  
- project\_db\_1: postgresql

**ports**暴露端口信息。使用宿主端口:容器端口的格式，或者仅仅指定容器的端口（此时宿主机将会随机指定端口） ， 类似于docker run -p

示例：  
ports:  
- "3000"  
- "3000-3005"  
- "8000: 8000"  
- "9090-9091: 8080-8081"  
- "49100: 22"  
- "127. 0. 0. 1: 8001: 8001"  
- "127. 0. 0. 1: 5000-5010: 5000-5010"

**expose**  
暴露端口，只将端口暴露给连接的服务，而不暴露给宿主机

示例：  
expose:  
- "3000"  
- "8000"

**volumes**卷挂载路径设置。 可以设置宿主机路径 （HOST: CONTAINER） 或加上访问模式 （HOST: CONTAINER: ro） 。 示例：  
volumes:  
# Just specify a path and let the Engine create a volume  
- /var/lib/mysql  
# Specify an absolute path mapping  
- /opt/data: /var/lib/mysql  
# Path on the host, relative to the Compose file  
- . /cache: /tmp/cache  
# User-relative path  
- ~/configs: /etc/configs/: ro  
# Named volume  
- datavolume: /var/lib/mysql

**volumes\_from**从另一个服务或者容器挂载卷。 可以指定只读或者可读写， 如果访问模式没有指定， 则默认是可读写。 示例：  
volumes\_from:  
- service\_name  
- service\_name: ro  
- container: container\_name  
- container: container\_name: rw

**environment**  
设置环境变量。 可以使用数组或者字典两种方式。 只有一个key的环境变量可以在运行Compose的机器上找到对应的值，这有助于加密的或者特殊主机的值。

示例：  
environment:  
RACK\_ENV: development  
SHOW: ' true'  
SESSION\_SECRET:  
environment:  
- RACK\_ENV=development  
- SHOW=true  
- SESSION\_SECRET

**env\_file**  
从文件中获取环境变量，可以为单独的文件路径或列表。 如果通过 docker-compose -f FILE 指定了模板文件，则 env\_file 中路径会基于模板文件路径。 如果有变量名称与 environment 指令冲突， 则以envirment 为准。

示例：  
env\_file: . env  
env\_file:  
- . /common. env  
- . /apps/web. env  
- /opt/secrets. env  
**extends**继承另一个服务， 基于已有的服务进行扩展。

**net**设置网络模式。

示例：  
net: "bridge"  
net: "host"  
net: "none"  
net: "container: [service name or container name/id] "

**dns**  
配置dns服务器。 可以是一个值， 也可以是一个列表。

示例：  
dns: 8. 8. 8. 8  
dns:  
- 8. 8. 8. 8  
- 9. 9. 9. 9

**dns\_search**配置DNS的搜索域， 可以是一个值， 也可以是一个列表

示例：  
dns\_search: example. com  
dns\_search:  
- dc1. example. com  
- dc2. example. com

其他  
docker­compose.yml 还有很多其他命令， 这里仅挑选常用命令进行讲解， 其它不作赘述。 如果感兴趣的，可以参考docker­compose.yml文件官方文档： https://docs.docker.com/compose/compose­file/