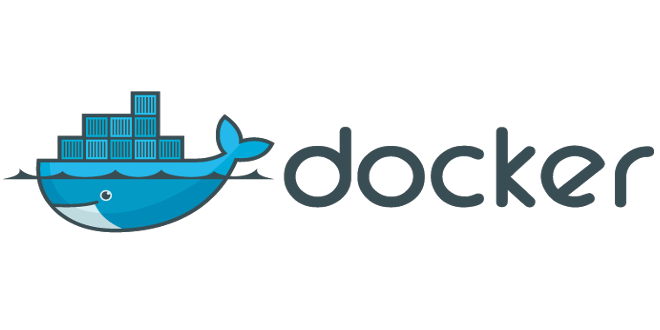
2013年发布至今， [Docker](https://www.docker.com/) 一直广受瞩目，被认为可能会改变软件行业。

但是，许多人并不清楚 Docker 到底是什么，要解决什么问题，好处又在哪里？今天就来详细解释，帮助大家理解它，还带有简单易懂的实例，教你如何将它用于日常开发。



# Docker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。

Docker的优点如下：

**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。

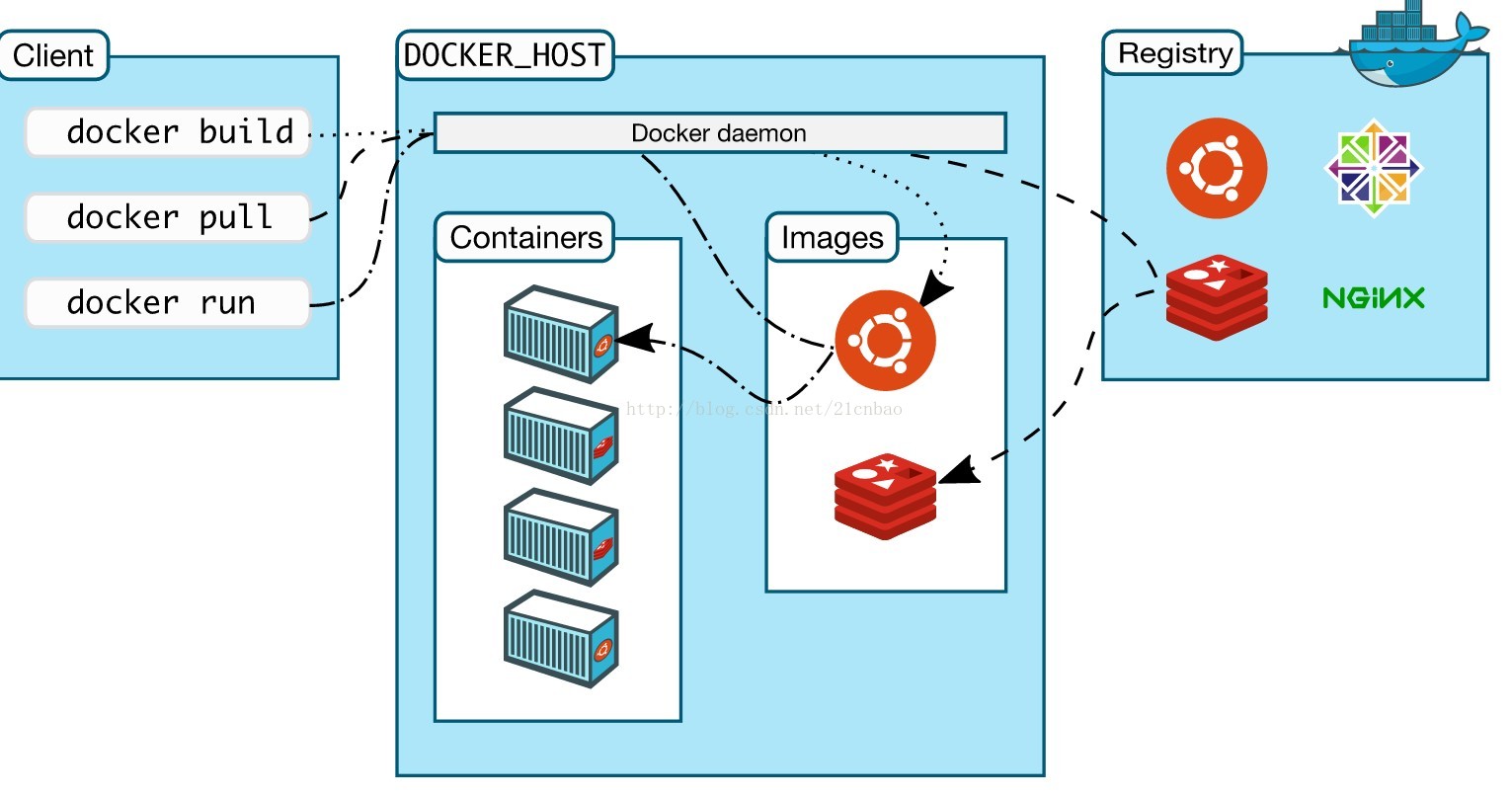
**2、避免选择恐惧症**

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。

# 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。

# Docker 的安装

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 3.10 ，查看本页面的前提条件来验证你的CentOS 版本是否支持 Docker 。

通过 uname -r 命令查看你当前的内核版本

# uname -r

2、使用 root 权限登录 Centos。确保 yum 包更新到最新。

# yum -y update

3、卸载旧版本(如果安装过旧版本的话)

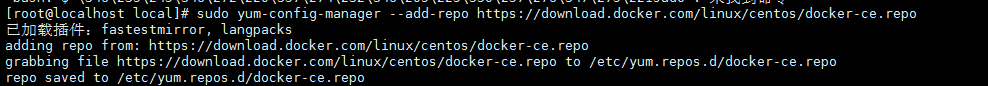
# yum remove docker docker-common docker-selinux docker-engine

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

# yum install -y yum-utils device-mapper-persistent-data lvm2

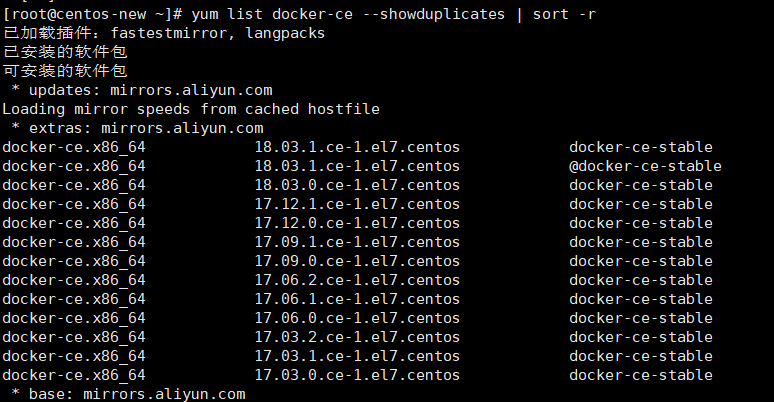
5、设置yum源

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



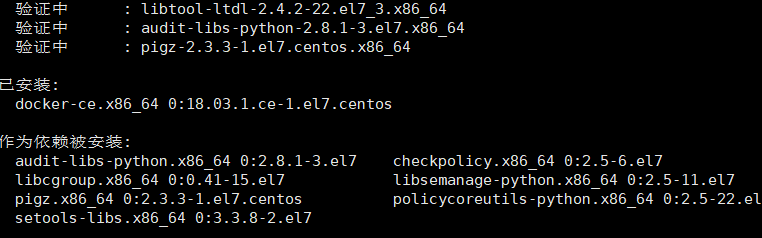
6、可以查看所有仓库中所有docker版本，并选择特定版本安装

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



7、安装docker

# sudo yum install docker-ce #由于repo中默认只开启stable仓库，故这里安装的是最新稳定版18.03.1# sudo yum install <FQPN> # 例如：sudo yum install docker-ce-18.03.1.ce

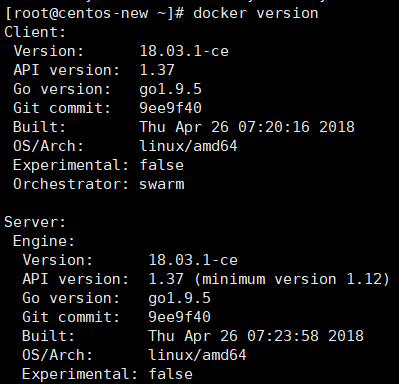


8、启动并加入开机启动

# systemctl start docker # systemctl enable docker

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)

# docker version



10、卸载docker

# yum -y remove docker-engine

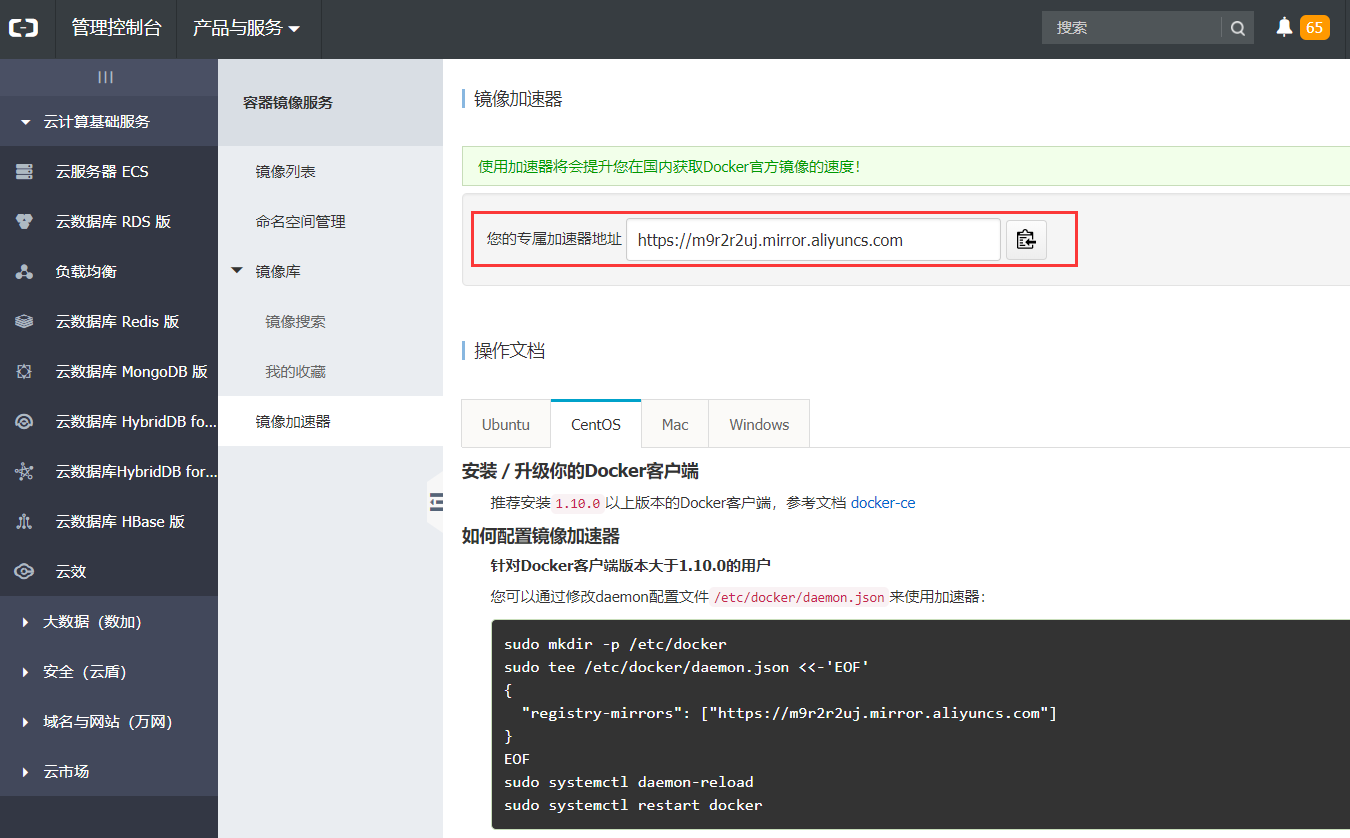
## 注意：需要配置镜像加速器

# docker search java

Error response from daemon: Get https://index.docker.io/v1/search?q=java: read tcp 52.200.132.201:443: 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：



# cd /etc/docker

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。

# vim daemon.json

{

  "registry-mirrors": ["<https://m9r2r2uj.mirror.aliyuncs.com>"]

}

保存退出。

重启docker服务

# service docker restart

成功！

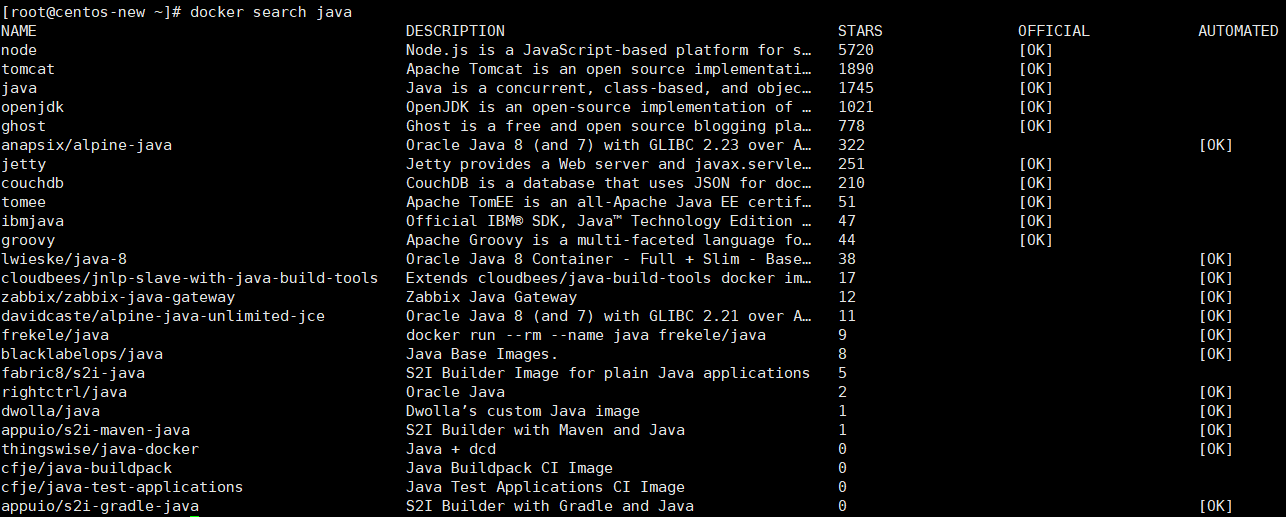
# Docker常用命令

### 镜像相关命令

**1、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。

# docker search java



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

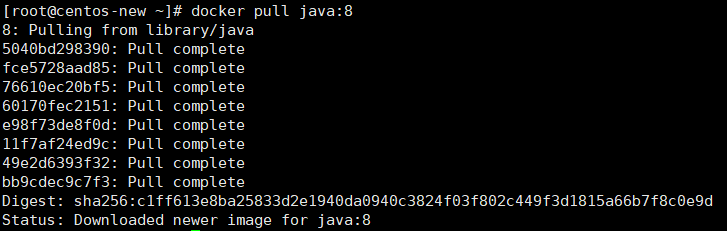
- OFFICAL:表示是否为官方仓库，该列标记为[0K]的镜像均由各软件的官方项目组创建和维护。

- AUTOMATED：表示是否是自动构建的镜像仓库。

**2、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

# docker pull java:8



**3、列出镜像**

使用 docker images命令即可列出已下载的镜像

# docker images

clipboard.png

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。

**4、删除本地镜像**

使用 docker rmi命令即可删除指定镜像

# docker rmi java

### 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。

--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。

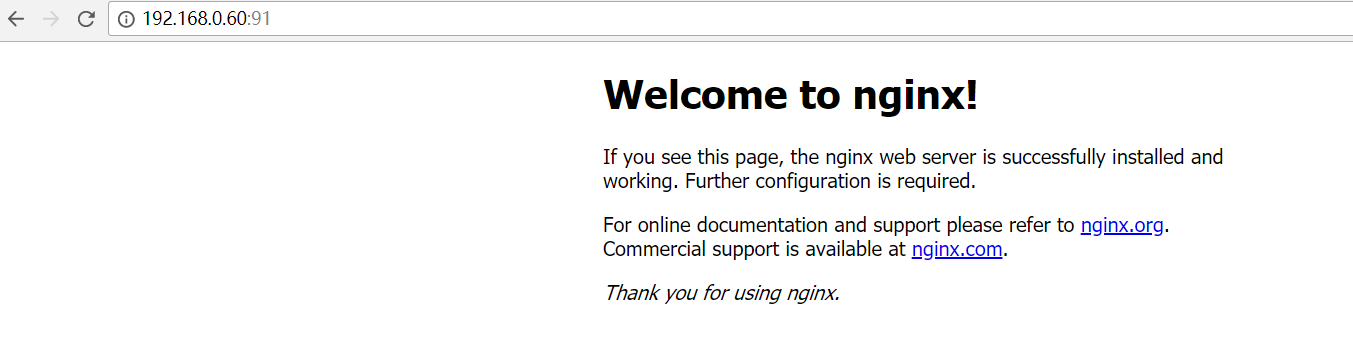
# docker run -d -p 91:80 nginx

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

# docker ps

clipboard.png

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

# docker stop f0b1c8ab3633

其中f0b1c8ab3633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

# docker kill f0b1c8ab3633

**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**

# docker start f0b1c8ab3633

**6、查看容器所有信息**

# docker inspect f0b1c8ab3633

**7、查看容器日志**

# docker container logs f0b1c8ab3633

**8、查看容器里的进程**

# docker top f0b1c8ab3633

**9、进入容器**

使用docker container 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

# docker container exec -it f0b1c8ab3633 /bin/bash

**9、删除容器**

使用 docker rm命令即可删除指定容器

# docker rm f0b1c8ab3633

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

**将微服务运行在docker上**

## 使用Dockerfile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：

FROM nginx

RUN echo '<h1>This is Tuling Nginx!!!</h1>' > /usr/share/nginx/html/index.html

该Dockerfile非常简单，其中的 FORM、 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：

# docker build -t nginx:tuling .

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

# docker run -d -p 92:80 nginx:tuling

4、访问 http://Docker宿主机IP:92/，可看到下图所示界面



## Dockerfile常用指令

|  |  |
| --- | --- |
| 命令 | 用途 |
| FROM | 基础镜像文件 |
| RUN | 构建镜像阶段执行命令 |
| ADD <src> <dest> | 添加文件，从src目录复制文件到容器的dest，其中 src可以是 Dockerfile所在目录的相对路径，也可以是一个 URL,还可以是一个压缩包 |
| COPY | 拷贝文件，和ADD命令类似，但不支持URL和压缩包 |
| CMD | 容器启动后执行命令 |
| EXPOSE | 声明容器在运行时对外提供的服务端口 |
| WORKDIR | 指定容器工作路径 |
| ENV | 指定环境变量 |
| ENTRYPINT | 容器入口， ENTRYPOINT和 CMD指令的目的一样，都是指定 Docker容器启动时执行的命令，可多次设置，但只有最后一个有效。 |
| USER | 该指令用于设置启动镜像时的用户或者 UID,写在该指令后的 RUN、 CMD以及 ENTRYPOINT指令都将使用该用户执行命令。 |
| VOLUME | 指定挂载点，该指令使容器中的一个目录具有持久化存储的功能，该目录可被容器本身使用，也可共享给其他容器。当容器中的应用有持久化数据的需求时可以在 Dockerfile中使用该指令。格式为： VOLUME["/data"]。 |

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/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

# 基于哪个镜像

From java:8

# 将本地文件夹挂载到当前容器

VOLUME /tmp

# 复制文件到容器

ADD microservice-eureka-server-0.0.1-SNAPSHOT.jar /app.jar

# 声明需要暴露的端口

EXPOSE 8761

# 配置容器启动后执行的命令

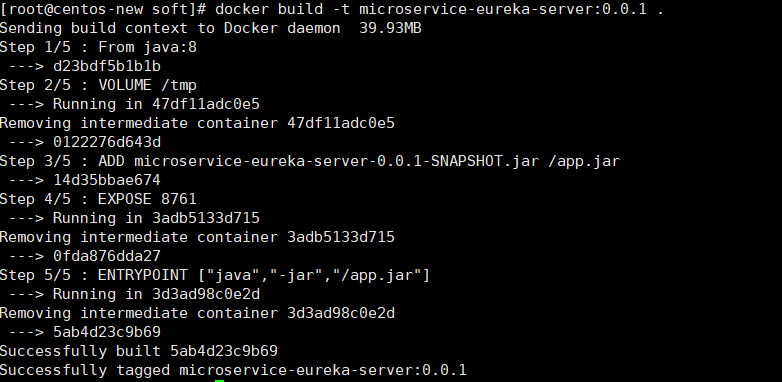
ENTRYPOINT ["java","-jar","/app.jar"]

3、使用docker build命令构建镜像

# docker build -t microservice-eureka-server:0.0.1 .

# 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

# docker run -p 8761:8761 microservice-eureka-server:0.0.1

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

