目录

[第一章、Docker 安装 2](#_Toc43401968)

[第一步 卸载旧的安装包 2](#_Toc43401969)

[第二步 安装需要的安装包 2](#_Toc43401970)

[第三步 设置镜像的仓库 2](#_Toc43401971)

[第四步# 安装docker相关的 3](#_Toc43401972)

[第五步启动docker 服务 3](#_Toc43401973)

[第六步 hello world 3](#_Toc43401974)

[第二章、Docker 卸载 3](#_Toc43401975)

[# 卸载依赖 3](#_Toc43401976)

[# 删除资源 3](#_Toc43401977)

[第三章、Docker镜像相关指令 4](#_Toc43401978)

[1 docker镜像基本命令： 4](#_Toc43401979)

[查看所有镜像 4](#_Toc43401980)

[docker images 4](#_Toc43401981)

[获取新的镜像： 4](#_Toc43401982)

[查找镜像 4](#_Toc43401983)

[删除镜像(会提示先停止使用中的容器) 5](#_Toc43401984)

[创建镜像 5](#_Toc43401985)

[更新镜像 5](#_Toc43401986)

[设置镜像标签 5](#_Toc43401987)

[第四章、 Docker容器基本命令 5](#_Toc43401988)

[查看全部命令 5](#_Toc43401989)

[容器实例基本操作 6](#_Toc43401990)

[容器导入导出 7](#_Toc43401991)

[第五章、Docker 仓库管理 7](#_Toc43401992)

[1 注册-登录 7](#_Toc43401993)

[2 DockerHub基本操作 7](#_Toc43401994)

[第六章、Dockerfile指令 8](#_Toc43401995)

[FROM：指定基础镜像，必须为第一个命令 8](#_Toc43401996)

[MAINTAINER: 维护者信息 8](#_Toc43401997)

[RUN：构建镜像时执行的命令 8](#_Toc43401998)

[ADD命令 9](#_Toc43401999)

[COPY： 10](#_Toc43402000)

[CMD： 10](#_Toc43402001)

[ENTRYPOINT： 10](#_Toc43402002)

[LABEL： 11](#_Toc43402003)

[ENV： 11](#_Toc43402004)

[EXPOSE： 12](#_Toc43402005)

[VOLUME： 12](#_Toc43402006)

[WORKDIR： 13](#_Toc43402007)

[USER: 13](#_Toc43402008)

[ARG： 14](#_Toc43402009)

[ONBUILD： 14](#_Toc43402010)

docker run \

--detach=true run the container in the background

--log-driver=syslog \

--name="elasticsearch" name of the container

--restart=always automatically start the container when the Docker daemon starts

--publish 9200:9200 Docker host port : mapped port in the container

--volume /dockerhost/dir/for/data:/var/lib/elasticsearch path that elasticsearch will use to store its data on the Docker host : mapped path in the container --env CLUSTER\_NAME=mycluster \

--env CLUSTER\_NODES="["node2.example.com", "node3.example.com"]" \

--env DOCKER\_HOST\_IP=ip.of.docker.host \

--env NODE\_NAME=node1 \

--publish 9300:9300 \

bryanhong/elasticsearch:latest

# 第一章、Docker 安装

#在root 用户下

su root ,然后回让你输入密码

## 第一步 卸载旧的安装包

yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

# 第二章、Docker 卸载

## # 卸载依赖

yum remove docker-ce docker-ce-cli containerd.io

## # 删除资源

rm -rf /var/lib/docker

## 第二步 安装需要的安装包

yum install -y yum-utils

## 第三步 设置镜像的仓库

yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo --默认是国外的

#如果没有vpn 建议安装阿里云的

yum-config-manager \

--add-repo \

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

#更新yum 索引安装包

yum makecache fast

## 第四步# 安装docker相关的

yum install docker-ce docker-ce-cli containerd.io (可能会出错，多试几次)

## 第五步启动docker 服务

systemctl start docker

#查看docker 是否安装完成

docker --version

## 第六步 hello world

docker run hello-world

#查看所有的docker 镜像

docker images

docker search aspnet

docker pull mcr.microsoft.com/dotnet/core/aspnet:latest

# 第三章、Docker镜像相关指令

## 1 docker镜像基本命令：

### 查看所有镜像

### docker images

* **REPOSITORY：**表示镜像的仓库源
* **TAG：**镜像的标签
* **IMAGE ID：**镜像ID
* **CREATED：**镜像创建时间
* **SIZE：**镜像大小

### 获取新的镜像：

docker pull 名称

### 查找镜像

docker search httpd

**NAME:**镜像仓库源的名称

**DESCRIPTION:**镜像的描述

**OFFICIAL:**是否 docker 官方发布

**stars:**类似 Github 里面的 star，表示点赞、喜欢的意思。

**AUTOMATED:**自动构建。

### 删除镜像(会提示先停止使用中的容器)

docker rmi  镜像name/镜像id

### 创建镜像

使用 Dockerfile 指令来创建一个新的镜像

**docker build** ， 从零开始来创建一个新的镜像。为此，我们需要创建一个 Dockerfile 文件，其中包含一组指令来告诉 Docker 如何构建我们的镜像。

### 更新镜像

1 运行的容器

2 docker exec -it fd2c868cadlc /bin/bash 进入容器操作

3 **apt-get update**

**4** exit

5 docker commit 来提交容器副本

docker commit -m="nginx has update0618" -a="eleven" 79323dxds323 nginx-8081-8082:vCustom2

* **-m:** 提交的描述信息
* **-a:** 指定镜像作者
* 79323dxds323**：**容器 ID
* nginx-8081-8082:vCustom2**:** 指定要创建的目标镜像名-tag

(要点时间)

### 设置镜像标签

docker tag 命令，为镜像添加一个新的标签。

docker tag 860c279d2fec runoob/centos:dev

docker tag 镜像ID，这里是 860c279d2fec ,用户名称、镜像源名(repository name)和新的标签名(tag)。

# 第四章、 Docker容器基本命令

## 查看全部命令

docker

具体命令详情 docker ps –help

启动容器 docker run -it nginx /bin/bash

* **-i**: 交互式操作。
* **-t**: 终端。
* **-d 后台运行**
* nginx: nginx镜像。
* **/bin/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 /bin/bash。(exit 退出终端)

## 容器实例基本操作

查看所有容器 docker ps -a

查看容器运行日志 docker logs 容器名称/容器id

停止容器运行 docker stop 容器name/容器id

终止容器后运行 docker start 容器name/容器id

容器重启 docker restart 容器name/容器id

删除容器 docker rm -f 容器name/容器id

#删除镜像

docker rmi -f 镜像id (可以根据 docker images 查询)

docker rmi -f $(docker images) --删除所有镜像

#查询docker 的详细信息

docker stats dockerid

查看 Docker 的底层信息

**docker inspect** 来查看 Docker 的底层信息

二、停止一个正在运行的容器

1、docker stop 此方式常常被翻译为优雅的停止容器

docker stop 容器ID或容器名

参数 -t：关闭容器的限时，如果超时未能关闭则用kill强制关闭，默认值10s，这个时间用于容器的自己保存状态

docker stop -t=60 容器ID或容器名

2、docker kill

docker kill 容器ID或容器名 :直接关闭容器

由此可见stop和kill的主要区别:stop给与一定的关闭时间交由容器自己保存状态，kill直接关闭容器

想更进一步了解处理机制的可以看下面这篇文章，比较详细但是需要其他方面的

1.停用全部运行中的容器:

docker stop $(docker ps -q)

2.删除全部容器：

docker rm $(docker ps -aq)

3.一条命令实现停用并删除容器：

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

## 容器导入导出

**导出容器**

如果要导出本地某个容器，可以使用 **docker export** 命令。

docker export 1e560fca3906 > nginx.tar

**导入容器快照**

可以使用 docker import 从容器快照文件中再导入为镜像

$ cat docker/ubuntu.tar | docker import - test/ubuntu:v1

也可以通过指定 URL 或者某个目录来导入

$ docker import http://example.com/exampleimage.tgz example/imagerepo

# 第五章、Docker 仓库管理

## 1 注册-登录

[https://hub.docker.com](https://hub.docker.com/)

eleven202001

eleven202001

xuyang@ZhaoxiEdu.Net

## 2 DockerHub基本操作

docker login

登录

docker logout

退出

搜索镜像：

docker search nginx

拉取镜像

docker pull nginx

推送镜像

docker push nginx-8081-8082:vCustom2（权限没成功）

# 第六章、Dockerfile指令

## FROM：指定基础镜像，必须为第一个命令

[复制代码](javascript:void(0);)

格式：

　　FROM <image>

　　FROM <image>:<tag>

　　FROM <image>@<digest>  
示例：  
　　FROM mysql:5.6  
注：  
　　tag或digest是可选的，如果不使用这两个值时，会使用latest版本的基础镜像

[复制代码](javascript:void(0);)

## MAINTAINER: 维护者信息

格式：

MAINTAINER <name>

示例：

MAINTAINER Jasper Xu

MAINTAINER sorex@163.com

MAINTAINER Jasper Xu <sorex@163.com>

## RUN：构建镜像时执行的命令

[复制代码](javascript:void(0);)

RUN用于在镜像容器中执行命令，其有以下两种命令执行方式：

**shell执行**

格式：

RUN <command>

**exec执行**

格式：

RUN ["executable", "param1", "param2"]

示例：

RUN ["executable", "param1", "param2"]

RUN apk update

RUN ["/etc/execfile", "arg1", "arg1"]

注：  
　　RUN指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不想使用这些缓存镜像，可以在构建时指定--no-cache参数，如：docker build --no-cache

[复制代码](javascript:void(0);)

## ADD命令

**：将本地文件添加到容器中，tar类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似wget**

[复制代码](javascript:void(0);)

格式：

ADD <src>... <dest>

ADD ["<src>",... "<dest>"] 用于支持包含空格的路径

示例：

ADD hom\* /mydir/ # 添加所有以"hom"开头的文件

ADD hom?.txt /mydir/ # ? 替代一个单字符,例如："home.txt"

ADD test relativeDir/ # 添加 "test" 到 `WORKDIR`/relativeDir/

ADD test /absoluteDir/ # 添加 "test" 到 /absoluteDir/

[复制代码](javascript:void(0);)

## COPY：

**功能类似ADD，但是是不会自动解压文件，也不能访问网络资源**

## CMD：

**构建容器后调用，也就是在容器启动时才进行调用。**

[复制代码](javascript:void(0);)

格式：

CMD ["executable","param1","param2"] (执行可执行文件，优先)

CMD ["param1","param2"] (设置了ENTRYPOINT，则直接调用ENTRYPOINT添加参数)

CMD command param1 param2 (执行shell内部命令)

示例：

CMD echo "This is a test." | wc -

CMD ["/usr/bin/wc","--help"]  
注：  
 　　CMD不同于RUN，CMD用于指定在容器启动时所要执行的命令，而RUN用于指定镜像构建时所要执行的命令。

[复制代码](javascript:void(0);)

## ENTRYPOINT：

**配置容器，使其可执行化。配合CMD可省去"application"，只使用参数。**

[复制代码](javascript:void(0);)

格式：

ENTRYPOINT ["executable", "param1", "param2"] (可执行文件, 优先)

ENTRYPOINT command param1 param2 (shell内部命令)

示例：

FROM ubuntu

ENTRYPOINT ["top", "-b"]

CMD ["-c"]  
注：  
　　　ENTRYPOINT与CMD非常类似，不同的是通过docker run执行的命令不会覆盖ENTRYPOINT，而docker run命令中指定的任何参数，都会被当做参数再次传递给ENTRYPOINT。Dockerfile中只允许有一个ENTRYPOINT命令，多指定时会覆盖前面的设置，而只执行最后的ENTRYPOINT指令。

[复制代码](javascript:void(0);)

## LABEL：

**用于为镜像添加元数据**

格式：

LABEL <key>=<value> <key>=<value> <key>=<value> ...

示例：

　　LABEL version="1.0" description="这是一个Web服务器" by="IT笔录"

注：

　　使用LABEL指定元数据时，一条LABEL指定可以指定一或多条元数据，指定多条元数据时不同元数据之间通过空格分隔。推荐将所有的元数据通过一条LABEL指令指定，以免生成过多的中间镜像。

## ENV：

**设置环境变量**

[复制代码](javascript:void(0);)

格式：

ENV <key> <value> #<key>之后的所有内容均会被视为其<value>的组成部分，因此，一次只能设置一个变量

ENV <key>=<value> ... #可以设置多个变量，每个变量为一个"<key>=<value>"的键值对，如果<key>中包含空格，可以使用\来进行转义，也可以通过""来进行标示；另外，反斜线也可以用于续行

示例：

ENV myName John Doe

ENV myDog Rex The Dog

ENV myCat=fluffy

[复制代码](javascript:void(0);)

## EXPOSE：

**指定于外界交互的端口**

[复制代码](javascript:void(0);)

格式：

EXPOSE <port> [<port>...]

示例：

EXPOSE 80 443

EXPOSE 8080  
 EXPOSE 11211/tcp 11211/udp  
注：  
　　EXPOSE并不会让容器的端口访问到主机。要使其可访问，需要在docker run运行容器时通过-p来发布这些端口，或通过-P参数来发布EXPOSE导出的所有端口

[复制代码](javascript:void(0);)

## VOLUME：

**用于指定持久化目录**

[复制代码](javascript:void(0);)

格式：

VOLUME ["/path/to/dir"]

示例：

VOLUME ["/data"]

VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"  
注：  
　　一个卷可以存在于一个或多个容器的指定目录，该目录可以绕过联合文件系统，并具有以下功能：

1 卷可以容器间共享和重用

2 容器并不一定要和其它容器共享卷

3 修改卷后会立即生效

4 对卷的修改不会对镜像产生影响

5 卷会一直存在，直到没有任何容器在使用它

[复制代码](javascript:void(0);)

## WORKDIR：

**工作目录，类似于cd命令**

[复制代码](javascript:void(0);)

格式：

WORKDIR /path/to/workdir

示例：

WORKDIR /a (这时工作目录为/a)

WORKDIR b (这时工作目录为/a/b)

WORKDIR c (这时工作目录为/a/b/c)  
注：  
　　通过WORKDIR设置工作目录后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT、ADD、COPY等命令都会在该目录下执行。在使用docker run运行容器时，可以通过-w参数覆盖构建时所设置的工作目录。

[复制代码](javascript:void(0);)

## USER:

**指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户。使用USER指定用户时，可以使用用户名、UID或GID，或是两者的组合。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户**

[复制代码](javascript:void(0);)

 格式:  
　　USER user  
　　USER user:group  
　　USER uid  
　　USER uid:gid  
　　USER user:gid  
　　USER uid:group

 示例：  
　　USER www

 注：

　　使用USER指定用户后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT都将使用该用户。镜像构建完成后，通过docker run运行容器时，可以通过-u参数来覆盖所指定的用户。

[复制代码](javascript:void(0);)

## ARG：

**用于指定传递给构建运行时的变量**

格式：

ARG <name>[=<default value>]

示例：

ARG site

ARG build\_user=www

## ONBUILD：

**用于设置镜像触发器**

[复制代码](javascript:void(0);)

格式：  
　　ONBUILD [INSTRUCTION]

示例：

　　ONBUILD ADD . /app/src

　　ONBUILD RUN /usr/local/bin/python-build --dir /app/src

注：  
　　当所构建的镜像被用做其它镜像的基础镜像，该镜像中的触发器将会被钥触发

[复制代码](javascript:void(0);)