Docker学习笔记

# 安装

## 环境并安装操作

环境ubuntu

sudo apt-get install docker.io

安装后，需要使用sudo

## 安装后目录

/usr/bin/docker

/usr/share/bash-completion/completions/docker

/usr/share/code/resources/app/extensions/docker

/etc/docker

/etc/default/docker

/etc/init.d/docker

/run/docker

/var/lib/docker -- 工作目录，镜像和容器都在这里

## docker info

使用docker info命令，查看信息

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 1

Server Version: 17.12.1-ce

Storage Driver: overlay2

Backing Filesystem: extfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9b55aab90508bd389d7654c4baf173a981477d55

runc version: 9f9c96235cc97674e935002fc3d78361b696a69e

init version: v0.13.0 (expected: 949e6facb77383876aeff8a6944dde66b3089574)

Security Options:

apparmor

seccomp

Profile: default

Kernel Version: 4.15.0-30-generic

Operating System: Ubuntu 18.04.1 LTS

OSType: linux

Architecture: x86\_64

CPUs: 4

Total Memory: 7.767GiB

Name: ubuntu

ID: WLMB:BPMV:YSOJ:433Q:DPTL:7VZ4:2GGK:CKY3:4CAI:IBSV:VT72:FNGO

Docker Root Dir: /var/lib/docker -- 根目录

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Labels:

Experimental: false

Insecure Registries:

127.0.0.0/8

Live Restore Enabled: false

WARNING: No swap limit support

## docker默认配置文件

/etc/default/docker文件

# Docker Upstart and SysVinit configuration file

#

# THIS FILE DOES NOT APPLY TO SYSTEMD

#

# Please see the documentation for "systemd drop-ins":

# https://docs.docker.com/engine/admin/systemd/

#

# Customize location of Docker binary (especially for development testing).

#DOCKERD="/usr/local/bin/dockerd"

# Use DOCKER\_OPTS to modify the daemon startup options.

#DOCKER\_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"

# If you need Docker to use an HTTP proxy, it can also be specified here.

#export http\_proxy="http://127.0.0.1:3128/"

# This is also a handy place to tweak where Docker's temporary files go.

#export DOCKER\_TMPDIR="/mnt/bigdrive/docker-tmp"

## 设置国内仓库和镜像

Mkdir -p /etc/docker // --如果没有就创建

Vim /etc/docker/daemon.json //--如果没有就编写

{

"registry-mirrors": ["<your accelerate address>"]

}

可以使用如下：

https://registry.docker-cn.com

http://hub-mirror.c.163.com

https://3laho3y3.mirror.aliyuncs.com

http://f1361db2.m.daocloud.io

<https://mirror.ccs.tencentyun.com>

重新启动Docker

Sudo systemctl daemon-reload

Sudo systemctl restart docker

# 镜像常用命令

## 获取镜像

docker pull <镜像文件：版本>， 如：docker pull ubuntu

## 查看镜像信息

docker images //-- 列出本地主机上已有的镜像

docker inspect //--返回一个JSON格式的消息，如果只有其中一项内容时，可以使用-f参数来指定，例如，获取镜像Architecture信息：

sudo docker inspect -f {{“.Architecture”}} 550

## 搜寻镜像

docker search //--搜索远端仓库中共享的镜像

--automated=false 仅显示自动创建的镜像

--no-trunc=false 输出信息不截断显示

-s, --starts=0 指定仅显示评价为指定星级以上的镜像

## 删除镜像

docker rmi

docker rmi IMAGE

## 看本机上存在的所有容器

docker ps -a

## 创建镜像

### 基于已有镜像的容器创建

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

-a, --author=”” 作者信息。

-m, --message=”” 提交消息

-p, --pause=true 提交时暂停容器运行

### 基于本地模板导入

例如：sudo cat ubuntu-14.04-x86\_64-minimal.tar.gz | docker import - ubuntu:14.04

## 存出和载入镜像

存出镜像到本地文件：docker save

从存出的本地文件中再导入到本地镜像库：docker load

## 上传镜像

docker push NAME[:TAG]

# 容器

## 创建容器

docker create -- 使用docker create命令新建的容器处于停止状态，可以使用docker start命令来启动

新建并启动容器

docker run == docker create docker start

如：sudo docker run ubuntu /bin/echo ‘Hello world’

下面的命令启动一个bash终端，允许用户进行交互

sudo docker run -t -i ubuntu:14.04 /bin/bash

在交互模式下，用户可以通过所创建的终端来输入命令，

用户可以按Ctrl+d 或是输入exit命令来退出容器

守护态运行， -d //--让Docker容器在后台以守护态形式运行。

sudo docker ps //-- 查看容器信息

获取容器输出信息 ：docker logs

## 终止容器

docker stop

docker ps -a -q命令看处于终止状态的容器的ID信息。

启动终止状态的docker start/restart

## 进入容器

attach命令

exec命令

nsenter工具

## 删除容器

docker rm [OPTIONS] CONTAINER [CONTAINER...]

-f, --force=false 强行终止并删除一个运行的容器

-l, --link=false 删除容器的连接，当保留容器

-v, --volumes=false 删除容器挂载的数据卷。

## 导入和导出容器

导出容器

docker export CONTAINER

导入容器： docker import - test/ubuntu:v1.0

# 仓库

## Docker Hub

<https://hub.docker.com>

## 创建和使用私有仓库

使用registry镜像创建私有仓库

sudo docker run -d -p 5000:5000 registry

这将自动下载并启动一个registry容器，创建本地的私有仓库服务。

默认情况下，会将仓库创建在容器/tmp/registry目录下。可以通过-v参数将镜像文件存放在本地的知道路径上。

sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry

# 数据管理

## 数据卷

在容器内创建一个数据卷

sudo docker run -d -p --name web -v /webapp training/webapp python app.py

挂载一个主机目录作为数据卷

sudo docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py

主机目录/src/webapp, 容器目录：/opt/webapp

Docker挂载数据卷默认权限是读写(rw)，用户也可以通过，ro指定为只读

sudo docker run -d -P --name web -v /src/webapp:/opt/webapp:ro training/webapp python app.py

挂载一个本地文件作为数据卷

sudo docker run --rm -it -v ~/.bash\_history:/.bash\_history ubuntu /bin/bash

## 数据卷容器

数据卷容器其实就是一个普通的容器，专门用它提供数据卷供其他容器挂载使用

首先创建数据卷容器，并在其中创建一个数据卷挂载到/dbdata:

sudo docker run -it -v /dbdata --name dbdata ubuntu

然后，可以在其他容器中使用--volumes-from来挂载dbdata容器中的数据卷

sudo docker run -it --volumes-from dbdata --name db1 ubuntu

sudo docker run -it --volumes-from dbdata --name db2 ubuntu

# 网络基础配置

## 端口映射实现访问容器

从外部访问容器应用

-P标记，Docker会随机映射一个49000~49900的端口至容器内壁

docker logs命令来查看应用的信息

-p则可以指定要映射的端口，并且，在一个指定端口上只可以绑定一个容器。支持的格式由:ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort.

映射所有接口地址

使用hostPort:containerPort格式将本地5000端口映射到容器的5000端口：

sudo docker run -d -p 5000:5000 training/webapp python app.py

此时默认会绑定本地所有接口上的所有地址。多次使用-p标记可以绑定多个端口。

sudo docker run -d -p 5000:5000 -p 3000:80 training/webapp python app.py

映射到指定地址的指定端口ip:hostPort:containerPort格式，指定映射使用一个特定地址，

sudo docker run -d -p 127.0.0.1:5000:5000 training/webapp python app.py

映射到指定地址的任意端口

使用ip::containerPort绑定localhost的任意端口到容器的5000端口，本地主机自动分配一个端口：

还可以使用udp标记指定udp端口：

sudo docker run -d -p 127.0.0.1:5000:5000/udp training/webapp python app.py

查看映射端口配置

docker port

## 容器互联实现容器间通信

略

# 使用Dockerfile创建镜像

## 基本结构

基础镜像信息/维护者信息，镜像操作指令和容器启动时执行指令。

## 指令

指令的一般格式为INSTRUCTION arguments，指令包括FROM、MAINTAINER、RUN等。

### from

### maintainer

格式为maintainer <name>，指定维护者信息。

### RUN

格式为RUN <command> 或Run [“executable”, “param1”,”param2”].

前者将在shell终端中运行命令，即/bin/sh -c; 后者则使用exec执行。

每条RUN指令将在当前镜像基础上执行指定命令，并提交为新的镜像。当命令较长时可以使用\来换行。

### CMD

支持3中格式：

CMD [“executable”,”param1”,”param2”] 使用exec执行，推荐方式。

CMD command param1 param2 在/bin/sh中执行，提供给需要交互的应用。

CMD [“param1”,”param2“]提供给ENTRYPOINT的默认参数。

指定启动容器时执行的命令，每个Dockerfile只能有一条CMD命令。如果指定了多条命令，只有最后一条会被执行。

如果用户启动容器时指定了运行的命令，则会覆盖掉CMD指令的命令。

### EXPOSE

EXPOSE <port> [<port>...] //-- EXPOSE 22 80 8443

告诉Docker服务端容器暴露的端口号，供互联系统使用。在启动容器时需要通过-P，Docker主动会自动分配一个端口转发到指定的端口；使用-p, 则可以具体指定哪个本地端口映射过来。

### ENV

格式为ENV <key> <value>。指定一个环境变量，会被后续RUN指令使用，并在容器运行时保持。

### ADD

ADD <src> <dest>

复制指定的<src>到荣去中的<dest>。

### COPY

COPY <src> <dest> 复制本地主机的<src>为容器中的<dest>。目标路径不存在时，会自动创建。

### ENTRYPOINT

ENTRYPOINT [“executable”, “param1”, “param2”]

ENTRYPOINT command param1 param2 (shell中执行)。

配置容器启动后执行的命令，并且不可被docker run提供的参数覆盖。

每个Dockerfile中只能由一个ENTRYPOINT，当指定多个ENTRYPOINT时，只有最后一个生效。

### VOLUME

VOLUME [“/data”]

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。

### USER

USER daemon

指定运行容器时的用户名或UID，后续的RUN也会指定用户。

### WORKDIR

WORKDIR /path/to/workdir

为后续的RUN,CMD,ENTRYPOINT指令配置工作目录。

可以使用多个WORKDIR指令，后续命令如果参数时相对路径，则会基于之前命令指定的路径。例如：

WORKDIR /a

WORKDIR b

WORKDIR C

RUN pwd

则最终路径为/a/b/c

### ONBUILD

ONBUILD [INSTRUCTION]

配置当所创建的镜像作为其他新创建镜像的基础镜像时，所执行的操作指令。

## 创建镜像

docker build命令来创建镜像。

基本格式为docker build [选项] 路径。

# docker操作命令

## 命令汇总

|  |  |  |  |
| --- | --- | --- | --- |
|  | 命令 | purpose | 用途 |
| 1 | attach | Attach to a running container | #当前shell下attach 连接指定运行镜像 |
| 2 | build | Build an image from a Dockerfile | #通过Dockerfile定制镜像 |
| 3 | commit | create a new image from a container’s changes | #提交当前容器为新的镜像 |
| 4 | cp | Copy files/folders from the containers filesystem to the host path | #从容器中拷贝指定文件或者目录到宿主主机中 |
| 5 | create | create a new container | # 创建一个新的容器，同run, 但不启动容器 |
| 6 | diff | Inspect changes on a container’s filesystem | #查看docker容器变化 |
| 7 | events | Get real time events from the server | #从docker服务获取容器实时事件 |
| 8 | exec | Run a command in an existing container | #在已存在的容器上运行命令 |
| 9 | export | stream the contents of a container as a tar archive | 导出容器的内容流作为一个tar归档文件 |
| 10 | history | show the history of an image | 展示一个镜像形成历史 |
| 11 | images | list images | 列出系统当前镜像 |
| 12 | import | create a new filesystem image from the contents of a tarball | 从tar包中的内容创建一个新的文件系统影像 |
| 13 | info | Display system-wide information | 显示系统相关信息 |
| 14 | inspect | return low-level information on a container | 显示容器详细信息 |
| 15 | kill | kill a running container | kill 指定docker容器 |
| 16 | load | load a image from a tar archive | 从一个tar包中加载一个镜像 |
| 17 | login | register or login to the docker registry server | 注册或者登陆一个docker源服务器 |
| 18 | logout | log out from a docker registry server | 从当前Docker registry 退出 |
| 19 | logs | Fetch the logs of a container | 输出当前容器日志信息 |
| 20 | port | lookup the public-facing port which is NAT-ed to private\_port | 查看映射端口对应的容器内部源端口 |
| 21 | pause | pause all processes within a container | 暂停容器 |
| 22 | ps | list containers | 列出容器列表 |
| 23 | pull | pull a image or a repository from the docker registry server | 从docker镜像源服务器拉取指定镜像或者库镜像 |
| 24 | push | push an image or a repository to the docker registry server | 推送指定镜像或者库镜像至docer源服务器 |
| 25 | restart | restart a running container | 重启运行的容器 |
| 26 | rm | remove one or more containers | 移除一个或者多个容器 |
| 27 | rmi | remove one or more images | 移除一个或多个镜像[无容器使用该镜像才可以删除，否则需删除相关容器才可以继续或-f强制删除] |
| 28 | run | run a command in a new contaier | 创建一个新的容器并运行一个命令 |
| 29 | save | save a image to a tar archive | 保存一个镜像为一个tar包 |
| 30 | search | search for an image on the docker hub | 在docker hub中搜索镜像 |
| 31 | start | start a stopped containers | 启动容器 |
| 32 | stop | stop a running containers | 停止容器 |
| 33 | tag | tag a image into a repository | 给源中镜像打标签 |
| 34 | top | lookup the running processes of a container | 查看容器中运行的进程信息 |
| 35 | unpause | unpause a paused container | 取消暂停容器 |
| 36 | version | show the docker version information | 参看docker版本号 |
| 37 | wait | Block until a container stops, then print its exit code | 截取容器停止是的退出状态 |

## Docker 命令

### 容器生命周期

#### Docker run

Docker run: 创建一个新的容器并运行一个命令

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

options说明：

* -a stdin 指定标准输入输出的内容类型，可选STDIN/STDOUT/STDERR三项
* -d 后台运行容器，并返回容器ID
* -i 以交互模式运行容器，通常与-t同时使用；
* -P 随机端口映射，容器内部端口随机映射到主机的高端口
* -p 指定端口映射，格式为：主机（宿主）端口：容器端口
* -t: 为容器重新分配一个伪输入终端，通常与-i同时使用；
* --name=“nginx-lb”:为容器指定一个名称；
* --dns 8.8.8.8 指定容器使用的DNS服务器，默认和宿主一致；
* --dns-search example.com指定容器DNS搜索域名，默认和宿主一致;
* -h “mars”: 指定容器的hostname;
* -e username = “ritchie”:设置环境变量
* --env-file=[]:从指定文件读入环境变量
* --cpuset=”0-2” or –cpuset=”0,1,2”:绑定容器到指定cpu运行
* -m:设置容器使用内存最大值；
* --net=”bridge”:指定容器的网络连接类型，支持bridge/host/none/container四种类型
* --link=[]:添加链接到另一个容器
* --expose=[]:开放一个端口或一组端口；
* --volume, -v 绑定一个卷
* --rm

在Docker容器退出时，默认容器内部的文件系统仍然被保留，以方便调试并保留用户数据。

但是，对于foreground容器，由于其只是在开发调试过程中短期运行，其用户数据并无保留的必要，因而可以在容器启动时设置--rm选项，这样在容器退出时就能够自动清理容器内部的文件系统。示例如下：

docker run --rm ba-208

等价于

docker run --rm=true ba-208

显然，--rm选项不能与-d同时使用（或者说同时使用没有意义），即只能自动清理foreground容器，不能自动清理detached容器。

注意，--rm选项也会清理容器的匿名data volumes。

所以，执行docker run命令带--rm命令选项，等价于在容器退出后，执行docker rm -v。

### 本地镜像管理

#### Save

将指定镜像保存成tar归档文件

Docker save [OPTIONS] IMAGE [IMAGE]

OPTIONS 说明：

-o :输出到的文件。

实例

将镜像 runoob/ubuntu:v3 生成 my\_ubuntu\_v3.tar 文档

runoob@runoob:~$ docker save -o my\_ubuntu\_v3.tar runoob/ubuntu:v3

runoob@runoob:~$ ll my\_ubuntu\_v3.tar

-rw------- 1 runoob runoob 142102016 Jul 11 01:37 my\_ubuntu\_v3.ta

#### Load

Docker load: 导入docker save命令导出的镜像

--input, -i :指定导入的文件，代替STDIN

--quiet, -q ：精简输出信息

#### Docker build命令

Docker build命令用于使用Dockerfile创建镜像

Docker build [OPTIONS] PATH | URL | -

* --build-arg=[] :设置镜像创建时的变量；
* --cpu-shares: 设置cpu使用权重
* --cpu-period:限制CPU CFS周期
* --cpu-quota:限制CPU CFS配额
* --cpuset-cpus:指定使用的CPU id;
* --cpuset-mems: 指定使用的内存id;
* --disable-content-trust: 忽略校验，默认开启
* -f 指定要使用的Dockerfile路径；
* --force-rm：设置镜像过程中删除中间容器；
* --isolation: 使用容器隔离技术；
* --label=[]:设置镜像使用的元数据
* -m:设置内存最大值
* --memory-swap:设置swap的最大为内存+swap，“-1”表示不限swap;
* --no-cache:创建镜像的过程不使用缓存；
* --pull:尝试去更新镜像的新版本；
* --quiet， -q:安静模式，成功后只输出镜像ID;
* --rm :设置镜像成功后删除中间容器；
* --shm-size：设置/dev/shm的大小，默认值是64M;
* --ulimit: Ulimit配置
* --tag -t 镜像的名字及标签，通常name:tag或者name格式；可以在一次构建中为一个镜像设置多个标签。
* --network：默认default。在构建期间设置Run指令的网络模式

使用当前目录的 Dockerfile 创建镜像，标签为 runoob/ubuntu:v1。

docker build -t runoob/ubuntu:v1 .

使用URL github.com/creack/docker-firefox 的 Dockerfile 创建镜像。

docker build github.com/creack/docker-firefox

也可以通过 -f Dockerfile 文件的位置：

$ docker build -f /path/to/a/Dockerfile .

## 操作技巧

### 进入docker内部，启动bash进行交互

sudo docker exec -it [containname] bash

### 跟踪日志

Usage: docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

Options:

--details Show extra details provided to logs

-f, --follow Follow log output

--since string Show logs since timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for 42 minutes)

--tail string Number of lines to show from the end of the logs (default "all")

-t, --timestamps Show timestamps

--until string Show logs before a timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for 42 minutes)

如果是配合文件或日志文件之类的，可以使用-v指令，将其外挂

# docker-compose命令

安装：sudo apt install docker-compose

## 命令汇总

|  |  |  |
| --- | --- | --- |
| 1 | build | Build or rebuild services |
| 2 | bundle | generate a docker bundle from the compose file |
| 3 | config | validate and view the compose file |
| 4 | create | create services |
| 5 | down | stop and remove containers, networks, images, and volumes |
| 6 | events | receive real time events from containers |
| 7 | exec | execute a command in a running container |
| 8 | help | get help on a command |
| 9 | kill | kill containers |
| 10 | logs | view output from containers |
| 11 | pause | pause serviecs |
| 12 | port | print the public port for a port binding |
| 13 | ps | list containers |
| 14 | pull | pull service images |
| 15 | push | push service images |
| 16 | restart | restart services |
| 17 | rm | remove stopped containers |
| 18 | run | run a one-off command |
| 19 | scale | set number of containers for a service |
| 20 | start | start services |
| 21 | stop | stop services |
| 22 | top | display the running processes |
| 23 | unpause | unpause services |
| 24 | up | create adn start containers |
| 25 | version | show the Docker-compose version information |

## 操作技巧

### 启动

sudo docker-compose -f docker-compose-cli.yaml up -d

### 关闭

sudo docker-compose -f docker-compose-cli.yaml stop

# Docker实践学习笔记

## Docker初探-构建一个Docker应用

### 写一个Dockerfile

创建一个目录docker-in-practice/ch1

Cd ch1

Vim Dockerfile

FROM node

MAINTAINER haolifengwang@126.com

RUN git clone -q https://github.com/docker-in-practice/todo.git

WORKDIR todo

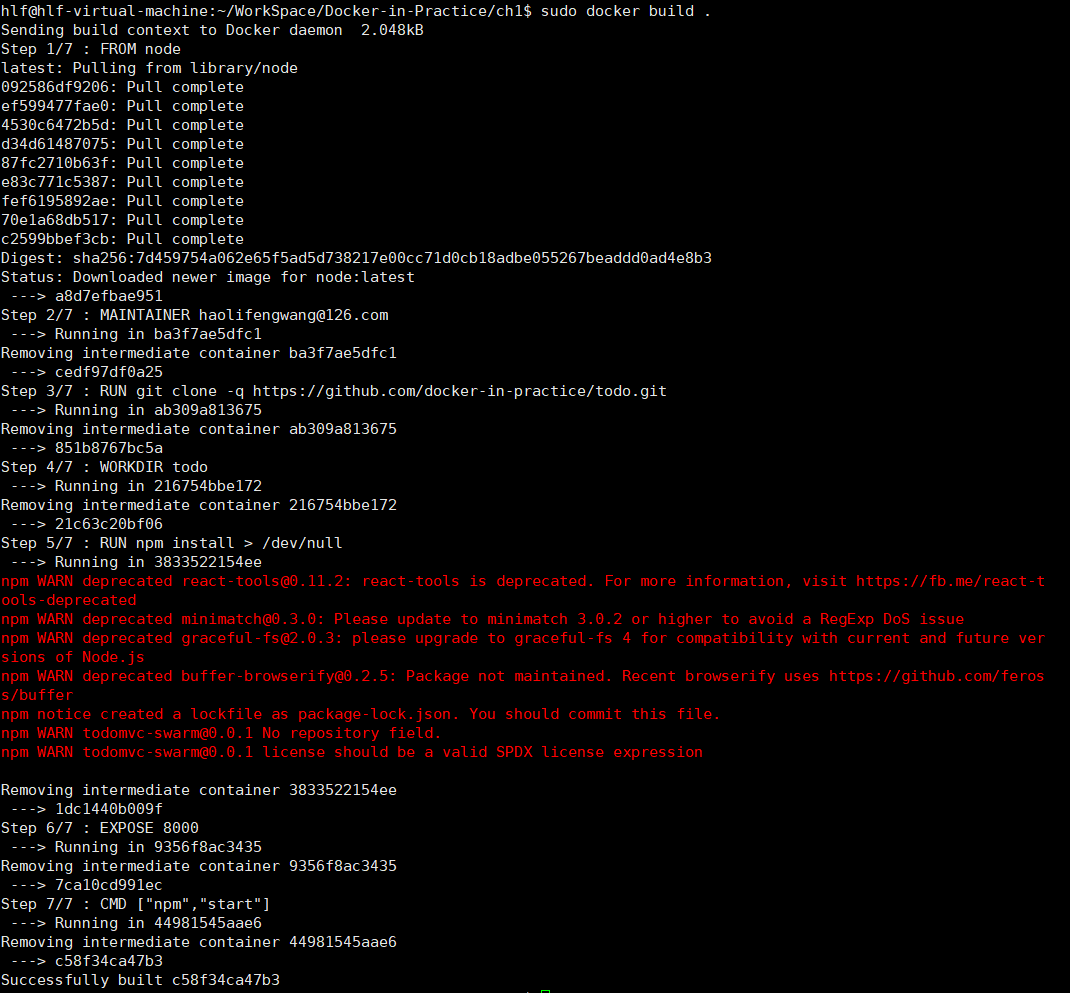
RUN npm install > /dev/null

EXPOSE 8000

CMD ["npm","start"]

### build

sudo docker build .



构建镜像后打标签

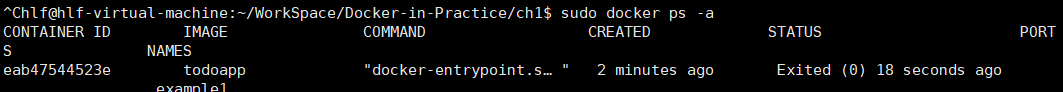
sudo docker tag c58f34ca47b3 todoapp

### 运行docker容器

sudo docker run -p 8000:8000 --name example1 todoapp

启动完毕后按ctrl-C键结束容器，使用docker ps -a 查看已经启动和移除的容器

sudo docker ps -a



重新启动容器

sudo docker start example1

再次docker ps -a

docker diff subcommand shows you which files have been affected since the image was instantiated as a container.

