1. 创建和启动一个容器
2. docker run --detach --name web nginx:latest

Docker 将从 docker hub 上的Nginx 仓库下载, 安装 Nginx:latest 镜像, 然后运行该软件. --detach( 或者使用 -d 缩写形式 )选项, 在后台启用该程序

1. Docker run -d --name mailer dockerinaction/ch2\_mailer

2. 运行交互容器

1. Docker run --interactive --tty \

--link web:web \

--name web\_test

Busybox:latest /bin/sh

该命令使用run命令的两个标志: --interactive(或-i)和--tty(或-t).

1. --interactive 选项告诉docker 保持标准输入流(stdin, 标准输入)对容器开放， 即使容器没有终端连接。
2. --tty选项告诉docker 为容器分配一个虚拟终端， 这将允许你发信号给容器

2.验证是否运行： wget -O - <http://web:80/>

3. docker run -it \

--name agent \

--link web:insideweb \

--line mailer:insidemailer \

Dockerinaction/ch2\_agent

使用缩略标志在交互式容器中启动监控器

3.列举, 停止, 重新启动和查看容器输出

1. Docker ps

会显示每个运行的容器中的一下信息：

* 该容器id
* 使用的镜像
* 容器中执行的命令
* 容器运行的市场
* 容器暴露的网络端口
* 容器名

2.

docker logs web 查看web 服务器运行是否正常

Docker logs mailer 查看邮件服务器运行是否正常

Docker logs agent 查看监控器运行是否正常

Docker stop web 暂停web服务

4.

SQL\_CID = $(docker create -e MYSQL\_ROOT\_PASSWORD=ch2demo mysql:5)

docker start $SQL\_CID

MAILER\_CID = $(docker create dockerinaction/ch2\_mailer)

docker start $MAILER\_CID

WP\_CID = $(docker create --link $SQL\_CID:mysql -p 80 \

-v /run/lock/apache2/ -v /run/apache2/ \

--read-only wordpress:4)

docker start $WP\_CID

AGENT\_CID = $(docker create --link $WP\_CID:insideweb \

--link $MAILER\_CID:insidemailer \

dockerinaction/ch2\_mailer)

docker start $AGENT\_CID

5. 查看容器中那些进程在运行着

docker run -d -p 80:80 --name lamp-test tutum/lamp

docker top lamp-test

1. 在容器内杀死进程
2. 查看容器内的进程

Docker exec lamp-test ps

1. 杀死进程

Docker exec lamp-test kill <PID>

1. 链接别名
2. 创建有效的链接目标

docker run -d --name mydb --expose 3306 \

alpine:latest nc -l 0.0.0.0:3306

1. 测试布简历链接的情况

docker run -it --rm \

dockerinaction/ch5\_ff echo This “shouldn’t” work.

1. 测试不正确的链接别名情况

docker run -it --rm \

--link mydb:wrongalias \

dockerinaction/ch5\_ff echo Wrong

1. 测试错误别名

docker run -it --rm \

--link mydb:database \

dockerinaction/ch5\_ff echo It worked

1. 停止并删除链接目标容器

Docker stop mydb && docker rm mydb

5.7.3

创建一条链接会在新容器中添加链接信息。这些链接信息一方面存储在环境变量中，另一方面通过在dns覆盖系统中添加主机名的映射来将链接信息注入新容器中。

6.2.1 跨容器的进程间通信

在容器间共享内存比主机上共享内存来的更加安全

启动生产者进程

docker run -d -u nobody --name ch6\_ipc\_producer \

dockerinaction/ch6\_ipc -producer

启动消费者进程

docker run -d -u nobody --name ch6\_ipc\_consumer \

-ipc container:ch6\_ipc\_producer \

dockerinaction/ch6\_ipc -consumer

6.2.2 开放内存容器

想要和主机运行在同一个命名空间中，可以使用开放内存容器

启动生产者进程

docker run -d --name ch6\_ipc\_producer \

--ipc host \

dockerinaction/ch6\_ipc -producer

启动消费者进程

docker run -d --name ch6\_ipc-consumer \

--ipc host \

dockerination/ch6\_ipc -consumer

6.3.3

1. 主机上的特定文件被特定用户所使用
2. 在主机上创建一个新文件

echo “e=mc^2” > garbage

1. 使用这个文件只能被文件的所有者读取

chmod 600 garbage

1. 将文件的所有者改为root（假设你拥有sudo权限）

sudo chown root:root garbage

1. 尝试用nobody用户读取文件

docker run --rm -v “$(pwd)”/garbage:/test/garbage \

-u nobody \

ubuntu:latest cat /test/garbage

1. 尝试用容器root用户读取文件

docker run --rm -v “$(pwd)”/garbage:/test/garbage \

-u root \

ubuntu:latest cat /test/garbage

7.1.4 创建新镜像

1. 从一个合适的基础镜像创建一个容器, 并设置入口点

docker run -it --name image-dev ubuntu:latest /bin/bash

1. 在容器中安装Git

apt-get -y install git

1. 创建并标记一个名为ubuntu-git的新镜像

docker commit -a “@dockerinaction” -m “added git” image-dev ubuntu-git

测试：

docker run --name cmd-git ubuntu-git version

7.3 导入和导出扁平文件系统

1.创建一个新容器并且使用export子命令来获得新容器文件系统的扁平复制

导出文件系统内容

docker run --name export-test dockerinaction/ch7\_packed:latest ./echo For Export

docker export --output contents.tar export-test

docker rm export-test

显示归档内容

tar -tf contents.tar

2.

创建内容

package main

import “fmt”

func main() {

Fmt.Println(“hello, world!”)

}

拉取一个包含有go编译器的镜像，编译并且静态链接这个代码

docker run --rm -v “$(pwd)”:/usr/src/hello -w /usr/src/hello golang:1.3 go build -v

将这个程序放到压缩文件中

tar -cf static\_hello.tar hello

可以使用docker import命令将它导入到镜像中（通过Unix管道将tar文件重定向）

docker import -c “ENTRYPOINT [\”hello\”]” - \

dockerinaction/ch7\_static <static\_hello.tar

运行并且查看它

docker run dockerinaction/ch7\_static

docker history dockerinaction/ch7\_static

8.1 使用Dockerfile 打包 Git

1. 先创建了带有四个指令的Dockerfile

# an example Dockerfile for installing Git on Ubuntu

FROM ubuntu:latest

MAINTAINER "dockerinaction@allingeek.com"

RUN apt-get update

RUN apt-get install -y git

ENTRYPOINT ["git"]

1. 在包含Dockerfile 文件的目录中使用 docker build 命令，从Dockerfile 文件创建一个新镜像

docker build --tag ubuntu-git:auto .

1. 查看镜像及运行镜像

docker images

docker run --rm ubuntu-git:auto version

8.2 构建一个基础镜像和另外两个大包有不同版本邮件的镜像。

第一步：构建一个基础镜像

1. 创建一个名为 .dockerignore 的文件， 然后将以下内容复制到文件中：

.dockerignore

mailer-base.df

mailer-logging.df

mailer-live.df

2. 创建一个名为 mailer-base 的文件， 并将下面的内容复制到文件中：

FROM debian:wheezy

MAINTAINER Jeff Nickoloff “dia@allingeek.com”

RUN grouopadd -r -g 2200 example && \

useradd -rM -g exmaple -u 2200 example

ENV APPROOT=“/app” \

APP=“mailer.sh”

VERSION=“0.6”

LABEL base.name=“Mailer Archetype” \

base.version=“${VERSION}”

WORKDIR $APPROOT

ADD . $APPROOT

ENTRYPOINT [“/app/mailer.sh”]

EXPOSE 33333

#不要在基础镜像中设置默认用户，否则接下来的实现将不能够更新镜像

# USER example.example

3. 开始构建镜像

docker build -t dockerinaction/mailer-base:0.6 -f mailer-base.df .

第二步：在33333端口启动一个邮件程序后台进程

1. 在mailer-logging.df 文件中写入如下：（log-impl需在同一路径下）

FROM dockerinaction/mailer-base:0.6

COPY log-impl ${APPROOT}

RUN chmod a+x ${APPROOT}/${APP} && \

chown example:example /var/log

USER example:example

VOLUME [“/var/log”]

CMD [“/var/log/mailer.log”]

2. 创建一个名为mailer.sh 的文件，并且将厦门的脚本复制到文件中

#！/bin/sh

printf “Logging Mailer has started. \n”

while true

do

MESSAGE=${NC -L -P 33333}

printf “[Message]: %s\n” “$MESSAGE” > $1

sleep 1

done

3. 使用下面的命令从包含 mailer-logging.df 文件的目录中构建mailer-logging 镜像：

docker build -t dockerinaction/mailer-logging -f mailer-logging.df .

4. 启动一个命名容器

docker run -d —name logging-mailer dockerinaction/mailer-logging

第三步：使用Amazon Web Services提供的Email Service

1. 在mailer-live.df 文件中写入如下：

FROM dockerinaction/mailer-base:0.6

ADD live-impl ${APPROOT}

RUN apt-get update && \

apt-get install -y curl python && \

curl “https://bootstrap.pypa.io/get-pip.py” -o “get-pip.py” && \

python get-pip.py && \

pip install awscli && \

rm get-pip.py && \

chmod a+x “${APPROOT}/${APP}”

RUN apt-get install -y netcat

USER example:example

CMD [“mailer@dockerinaction.com”, “paget@dockerinaction.com”]

2. 在包含有mailer-live.df 文件的目录下创建一个名为live-impl 的子目录，并在这个子目录下， 将下面的脚本复制到名为mailer.sh的文件中:

#! /bin/sh

printf "Live Mailer has started. \n"

while true

do

MESSAGE=$(nc -l -p 33333)

aws ses send-email --from $1 \

--destination {\"ToAddresses\":{\"$2\"} \

\"Body\": {\"Text\":{\"Data\":\"$MESSAGE}\"}}}"

sleep 1

Done

3. 构建新景象，然后启动一个新容器

docker build -t dockerinaction/mailer-live -f mailer-live.df .

docker run -d name live-mailer dockerinaction/mailer-live