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 创建新景象

1. 从一个合适的基础景象创建一个容器, 并设置入口点

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

1. 创建并标记一个名为ubuntu-git的新景象

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

测试：

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