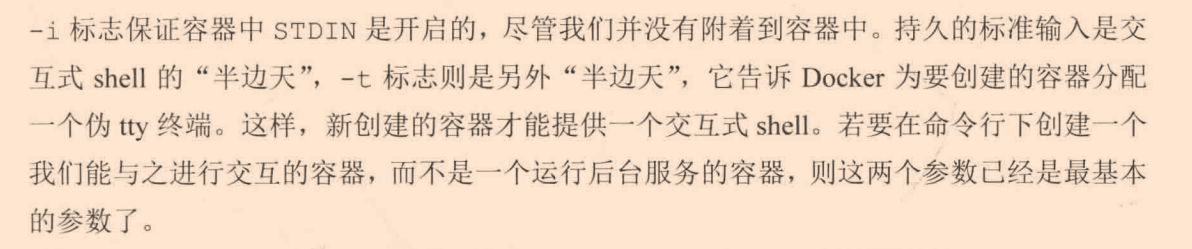
**运行容器： sodo docker run -i -t ubuntu /bin/bash**



**-d： 容器后台运行**

ubunte告诉docker基于ubuntu**镜像**来创建容器，然后在容器中运行/bin/bash启动了一个Bash shell （没有该镜像会创建）

**exit推出shell界面，但是会导致容器停止运行**

**列出容器列表: docker ps （正在运行的）、 docker ps -a 所有容器列表**

**给容器命名： sodu docker run --name XXX -I -t ubuntu /bin/bash**

**创建一个容器并命名为XXX**

**重新启动已经停止的容器：sudo docker start (uuid、id、name都可)**

**启动容器后,只会在后台运行，不会跳出来命令行窗口需要我们用： docker attach**

**命令，重新附着到该容器的会话上。可以通过id等附着到指定容器 docker attach name**

**但是如果退出容器的shell，容器再次回停止运行**

**创建守护线程，命令与创建容器差不多，但-i -t 改为了-d 表示为后台运行。**

**sudo docker run --name XX -d ubuntu /bin/sh -c “while true; do echo hello world ; sleep 1; done”**

**docker logs dockername查看docker容器日志**

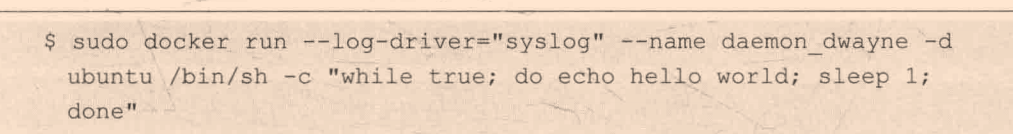
**docker logs -f dockername 与tail -f一致**

**docker logs --tail 10 dockername 查看最后10行**

**docker logs --tail 0 -f dockername 跟踪最新日志**

**也可以加-t 吧每个日志加上时间戳。docker logs -ft dockername**

**修改docker日志驱动，参数：--log-driver**



**查看容器内的进程：docker top dockername**

**docker 统计信息：docker stats dockername**

**docker中运行进程（后台任务）：docker exec -d dockername touch /erc/new\_config\_file**

**docker中运行进程（交互式任务）：docker exec -I -t dockername /bin/bash**

**停止进程：docker stop dockername**

**删除容器： docker rm dockername**

**查看镜像：docker images**

**搜索镜像：docker search XX**

**下载镜像： docker pull imagesname**

**使用镜像名称+冒号+版本号启动不同版本镜像：docker run -i -t --name ubuntu\_12 ubuntu:12.04 /bin/bash**

**构建镜像：**

1. **使用docker commit 命令**

**sudo docker commit 容器id 镜像仓库/镜像名**

**提交另一个新的定制容器：**

**docker commit -m “A new custom image” -a “James Turnbull” 容器id 仓库id/镜像名称**

**-m 指定新创建镜像提交信息 --a镜像作者**

1. **使用docker build和Dockerfile文件（推荐）**
2. **创建Dockerfile：**

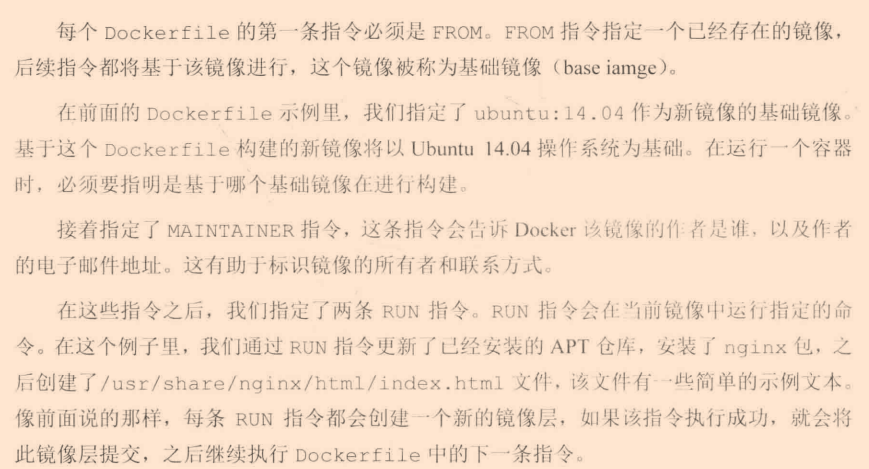
**FROM ubuntu:14.04**

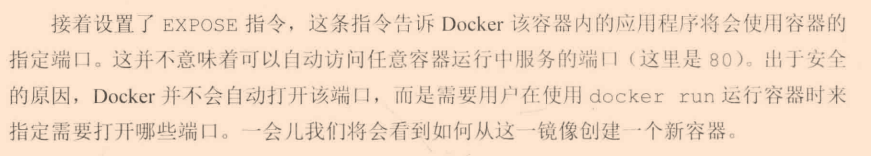
**MAINTAINER James Turnbull “james@example.com”**

**RUN apy-get update && apt-get install -y nginx**

**RUN echo ‘Hi, I am in your container’ >usr/share/nginx/html/index.html**

**EXPOSE 80**





1. **构建新镜像：  
   docker build -t=”jamtur01/static\_web” .仓库名为jamtur01，镜像名为static\_web**

**(a)也可以加上标签：**

**docker build -t=”jamtur01/static\_web:v1” .**

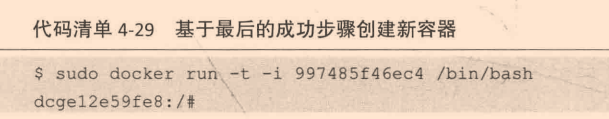
**(b)可以从git仓库指定Dockerfile：**

**docker build -t=”jamtur01/static\_web:v1” git@github.com:jamtur01**

**/docker-static\_web**

**这里假设在这个Git仓库的根目录下存在Dockerfile文件。**

**构建失败：**



**从新镜像启动容器： docker run -d -p 80 --name static\_web 仓库名/镜像名 nginx -g “daemon off;”**

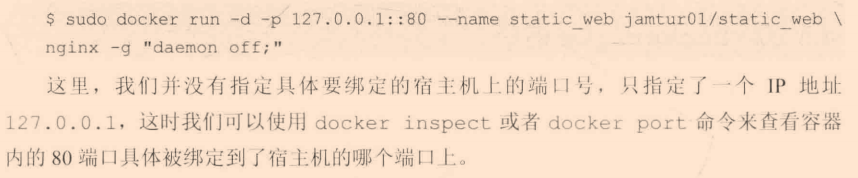
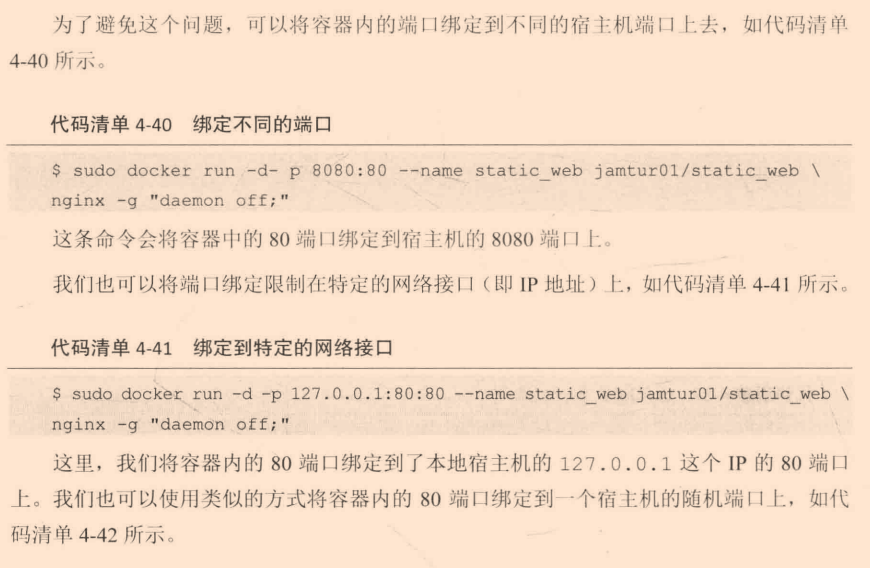
**这里启动了一个名为static\_web的新容器，同时指定了-d选项，告诉docker以分离的方式在后台运行。同时也执行了需要在容器中运行的命令 nginx -g “daemon off;”。这将以前台运行的方式启动Nginx，来作为我们的Web服务器。这里使用-p标志，控制docker在运行时应该公开哪些网络端口给外部。**

**运行一个容器时，docker可以通过两种方式来在宿主机上分配端口：**

1. **docker可以在宿主机上随机选择一个位于32768 ~ 61000的一个比较大的端口号来映射到容器的80端口上。**
2. **可以在Docker宿主机中指定一个具体的端口号来映射到容器中的80端口上。**

**使用docker ps 查看端口分配情况，也可以通过docker port来查看容器的端口映射情况。**

**也可以使用 docker run -d -p 80:80 --name static\_web 仓库名/镜像名 nginx -g “daemon off;” 来指定将80端口绑定到本机宿主机的80端口上。但是要小心使用，因为如果要运行多个容器，只有一个容器能成功的将端口绑定到本地宿主机上。这将限制docker的灵活性。**



# [docker重命名镜像](https://www.cnblogs.com/Presley-lpc/p/9315989.html)

docker tag IMAGEID(镜像id) REPOSITORY:TAG（仓库：标签）

**删除镜像： docker rmi** **镜像名**

**将本地镜像push到dockerhub上面：docker push 仓库名/镜像名**

挂载卷：将本地机上的某个文件夹让目标机器也能访问到，目标机器分配磁盘空间（猜测）。

docker挂载卷没有权限，则生成容器的时候添加：--privileged=true

或者外部 ：setenforce 0

# 两个docker互相能够访问端口（例如java镜像访问redis 镜像）

首先新建docker network：

docker network