# 如何使用Xshell连接VMware上的Linux虚拟机

参考：<https://www.cnblogs.com/shireenlee4testing/p/9469650.html>

# 安装docker

参考：<https://www.cnblogs.com/qgc1995/archive/2018/08/29/9553572.html>

centos7使用yum安装软件提示 cannot find a valid baseurl for repo:base/7/x86\_64 的解决方法

由于是本地yum源安装软件，无法联网，因此使用yum安装软件时报了错，解决方法是：打开vi /etc/resolv.conf文件 新增内容如下：nameserver 8.8.8.8

1.把yum包更新到最新的（如果更新yum有问题，也可以不更新试试）

2.安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

[root@localhost ~]# yum install -y yum-utils device-mapper-persistent-data lvm2

3.设置yum源

[root@localhost~]#yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

4.可以查看所有仓库中所有docker版本，并选择特定版本安装

[root@localhost ~]# yum list docker-ce --showduplicates | sort –r

5. 安装Docker，命令：yum install docker-ce-版本号，我选的是17.12.1.ce，如下

[root@localhost ~]# yum install docker-ce-17.12.1.ce

6. 启动Docker，命令：systemctl start docker，然后加入开机启动，如下

[root@localhost ~]# systemctl start docker

[root@localhost ~]# systemctl enable docker

Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

验证安装是否成功(有client和service两部分表示docker安装启动都成功了)

[root@localhost ~]# docker version

如果docker未启动则会报Cannot connect to the Docker daemon. Is the docker daemon running on this host?

Docker是服务器—客户端架构。命令行运行docker命令的时候，需要本机有docker服务。可以用下面的命令启动：

#service命令的用法

sudo service docker start

#systemctl命令的用法（centos7用这个）

sudo systemctl start docker

# 第一本DOCKER书阅读笔记

## Docker简介

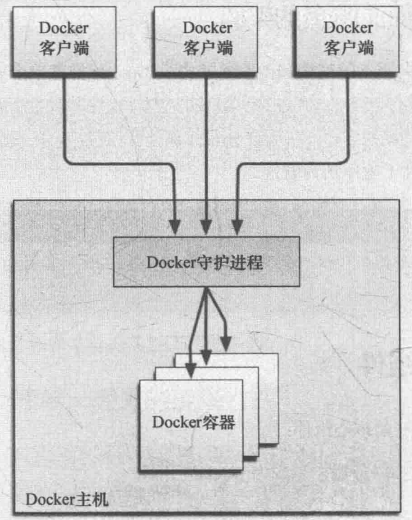
提供一个简单、轻量的建模方式；职责的逻辑分离；快速、高效的开发生命周期；鼓励使用面向服务的架构。

### 1.2 docker组件

核心组件：docker客户端和服务器，也称为docker引擎；docker镜像；registry；docker容器。

**Docker客户端和服务器**

Docker是一个客户端/服务器（C/S）架构的程序。Docker客户端只需向docker服务器或守护进程发出请求，服务器或守护进程将完成所有工作并返回结果。Docker守护进程有时也称为docker引擎。用户可以在同一台宿主机上运行docker守护进程和客户端，也可以从本地的docker客户端连接到运行在另一台宿主机上的远程docker进程。Docker架构如下：



**Docker镜像**

是构建docker世界的基石，用户基于镜像来运行自己的容器。而且镜像是基于联合（union）文件系统的一种层式的结构，由一系列指令一步一步构建出来的，例如：添加一个文件；执行一个命令；打开一个端口。

**Registry**

Docker用registry来保护用户构建的镜像。Registry分为公共和私有两种。

**容器**

用户把自己的应用程序或服务打包放进容器，容器是基于镜像启动起来的，容器中可以运行一个或多个进程。镜像是docker生命周期中的构建或打包阶段，而容器则是启动或执行阶段。总结起来，docker容器就是：一个镜像格式；一系列标准的操作；一个执行环境。

Docker借鉴了标准集装箱的概念。每个容器都包含一个软件镜像，也就是容器的货物，而且与真正的货物一样，容器里的软件镜像可以进行一些操作。例如，镜像可以被创建、启动、关闭、重启以及销毁。

### 1.5 docker的技术组件

包括以下几个部分：

一个原生的linux容器格式，docker中称为libcontainer

Linux内核的命名空间（namespace），用于隔离文件系统、进程和网络

文件系统隔离：每个容器都有自己的root文件系统

进程隔离：每个容器都运行在自己的进程环境中

网络隔离：容器间的虚拟网络接口和ip地址都是分开的。

资源隔离和分组：使用cgroups（即control group，linux的内核特性之一）将cpu和内存之类的资源独立分配给每个docker容器

写时复制：文件系统都是通过写时复制创建的，这就意味着文件系统时分层的、快速的，而且占用的磁盘空间更小。

日志：容器产生的STDOUT、STDERR和STDIN这些IO流都会被收集并记入日志，用来进行日志分析和故障排错

交互式shell：用户可以创建一个伪tty终端，将其连接到STDIN，为容器提供一个交互式的shell。

## Docker入门

**查看docker程序是否正常工作：** sudo docker info

该命令会返回所有容器和镜像（镜像即是docker用来构建容器的“构建快”）的数量、docker使用的执行驱动和存储驱动（execution and storage driver），以及docker的基本配置。

**运行我们的第一个容器**：可以使用docker run命令创建容器

sudo docker run –i –t Ubuntu /bin/bash

参数-i标志保证容器中STD是开启的，尽管我们并没有附着到容器中、持久的标准输入是交互式shell的“半边天”

参数-t标志则是另外“半边天”，它告诉docker为要创建的容器分配一个伪tty终端。这样，新创建的容器才能提供一个交互式shell。若要在命令行下创建一个能与之进行交互的容器，而不是一个运行后台服务的容器，则这两个参数是最基本的了。

(执行过程中遇到的报错：error pulling image configuration: Get https://production.cloudflare.docker.com/registry78a7489334c85e16bf4a9e08440c3286781783015a6c43d0/data?verify=1549179284-fa%2BB1UwaM6BuZdot yet valid ，原因是系统时间不同步，执行时间同步操作即可。ntpdate time.windows.com ，如果提示ntpdate 未找到命令，则需要执行yum install ntpdate，安装命令即可)

示例中使用的是ubuntu镜像。——首先docker会检查本地是否存在Ubuntu镜像，如果本地没有该镜像，那么docker就会连接官方维护的docker hub registry，查看docker hub中是否有该镜像。Docker一旦找到该镜像，就会下载该镜像并将其保存到本地宿主机中。随后，docker在文件系统内部用这个镜像创建了一个新容器。该容器拥有自己的网络、ip地址，以及一个用来和宿主机进行通信的桥接网络接口。最后，我们告诉docker在新容器中要运行什么命令，在本例中我们在容器中运行/bin/bash命令启动了一个bash shell。当容器创建完毕之后，docker就会执行容器中的/bin/bash命令，这时就可以看到容器内的shell了。

容器可以通过exit命令退出

**docker ps**

默认情况下，当执行docker ps命令时，只能看到正在运行的容器。如果指定-a标志的话，那么会列出所有容器，包括正在运行和已经停止的。也可以指定-l标志，列出最后一个运行的容器，无论其正在运行还是已经停止；也可以通过—format标志，进一步控制显示哪些信息以及如何显示这些信息。

该命令的输出结果包括：id、用于创建该容器的镜像、容器最后执行的命令、创建时间已经容器的退出状态、每个容器的名称。

### 容器命名

Docker会为创建的每一个容器自动生成一个随机的名称。也可以通过--name标志为容器指定一个名称

sudo docker run -–name bob\_the\_container –i –t Ubuntu /bin/bash

上述命令将会创建一个名为bob\_the\_container的容器。一个合法的容器名称只能包含以下字符：小写字母a~z、大写字母A~Z、数字0~9、下划线、圆点、横线（即[a-zA-Z0-9\_.-]）

在很多docker命令中，都可以用容器名称来替代容器id。容器名称有助于分辨容器。

容器的命名必须是唯一的，如果试图创建两个名称相同的容器，则命令将会失败。如果要使用的容器名称已经存在，可以先用docker rm删除已有的同名容器后，再创建新的容器。

### 重新启动已经停止的容器

通过容器名称：

docker start bob\_the\_container

也可以用容器id来指定容器：docker start aafldje309f

也可以使用docker restart命令重新启动一个容器。这时运行不带-a标志的docker ps命令就可以看到容器已经开始运行了。

也可以通过docker create命令创建一个容器但是并不运行它。这让我们可以在自己的容器工作流中对其进行细粒度的控制。

### 附着到容器上

在docker重新启动后，可以用docker attach命令重新附着到该容器的会话上：

docker attach bob\_the\_container

也可以使用容器id。通过这个命令又重新回到了容器的bash提示符。如果退出容器的shell，容器会再次停止运行。

### 创建守护式容器

除了交互式运行的容器，也可以创建长期运行的容器。守护式容器（daemonized container）没有交互式会话，非常适合运行应用程序和服务。

sudo docker run --name daemon\_dave -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"

通过-d参数，docker会将容器放到后台运行。上面还在容器要运行的命令里使用了一个while循环，会一直打印hello world直到容器或其进程停止运行。

该命令会返回一个容器id，如果执行docker ps命令，可以看到一个正在运行的容器。

### 容器内部都在干些什么

可以通过**docker logs**命令来获取容器的日志：docker logs daemon\_dave

可以看到while循环正向日志里打印hello world；可以通过-f参数来监控docker的日志，这与tail –f命令很相似

（tail -f等同于--follow=descriptor，根据文件描述符进行追踪，当文件改名或被删除，追踪停止）



提示：可以通过Ctr+C退出日志跟踪。

### Docker日志驱动

可以在启动docker守护进程或者执行docker run命令时通过—log-deriver选项控制docker守护进程和容器所用的日志驱动。

有好几个选项值，包括默认的json-file，json-file为前面看到的docker logs命令提供了基础。

其他可用的选项值还包括syslog，该选项将禁用docker logs命令，并将所有容器的日志输出都重定向到Syslog。

Sudo docker run –log-driver=”syslog” –name daemon\_dwayne –d Ubuntu /bin/sh –c “while true; do echo hello world; sleep 1; done”

这段命令会将daemon\_dwayne容器的日志都输出到Syslog，导致docker logs命令不输出任务东西。

同意可以通过选项值none禁用所有容器中的日志，导致docker logs命令也被禁用。

### 查看容器内的进程

除了容器的日志，也可以通过docker top命令查看容器内部运行的进程

sudo docker top daemon\_dave

### docker统计信息

通过docker stats 命令显示一个或多个容器的统计信息

docker stats daemon\_dave daemon\_kare daemon\_clare

可以看到一个守护式容器的列表，以及它们的CPU、内存、网络I/O及存储I/O的性能和指标。这对快速监控一台主机上的一组容器非常有用。

### 在容器内部运行进程

通过docker exec命令在容器内部额外启动新进程，包括两类：后天任务和交互式任务

**在容器中运行后台任务**：sudo docker exec –d daemon\_dave touch /etc/new\_config\_file

-d标志表明需要运行一个后台进程，-d标志之后，指定的是要在内部执行这个命令的容器的名字以及要执行的命令。上面例子中的命令会在daemon\_dave容器内创建一个文件名为/etc/new\_config\_file的空文件。也可以通过-u标志为新启动的进程指定一个用户属主

通过docker exec后台命令，可以在正在运行的容器中进行维护、监控及管理任务。

**在容器内运行交互命令**：docker exec –t –i daemon\_dave /bin/bash 在容器中启动了一个打开shell的交互式任务。和运行交互容器一样，-t和-i为进程创建了TTY并捕捉STDIN，接着指定了要在内部执行这个命令的容器的名字以及要执行的命令

### 停止守护式进程

只需要执行docker stop命令：docker stop daemon\_dave，也可以用容器id来指代容器名称

docker stop命令会向docker容器进程发送SIGTERM信号。如果想快速停止某个容器，也可以使用docker kill命令来向容器进程发送SIGKILL信号。

可以使用docker ps命令查看已停止的容器的状态。命令docker ps –n –x会显示最后x个容器，不论这些容器正在运行还是已经停止。

### 自动重启容器

如果由于某种错误而导致容器停止运行，可以通--restart参数让docker自动重新启动该容器。--restart标志会检查容器的退出代码，并据此来决定是否要重启容器，默认的行为是docker不会重启容器

docker run **--restart=always** --name daemon\_dave -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"

--restart被设置为always时，无论容器的退出代码是什么，docker都会自动重启该容器；当设置为on-failure时，只有当容器的退出代码为非0值的时候，才会自动重启

另外，可以指定重启次数：--restart=on-failure:5。这样，当容器退出代码为非0时，docker会尝试自动重启该容器，最多重启5次。

### 深入容器

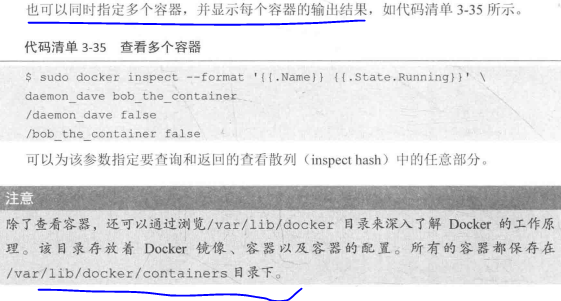
除了通过docker ps命令获取容器的信息，还可以使用docker inspect来获得更多的容器信息：docker inspect daemon\_dave

Docker inspect命令会对容器进行详细的检查，然后返回其配置信息，包括名称、命令、网络配置以及很多有用的数据

也可以用-f或者-format来选定查看结果

docker inspect --format=’{{ .State.Running}}’ daemon\_dave 返回结果为false，表示了容器的运行状态。如要获取容器ip地址：

docker inspect --format '{{.NetworkSettings.IPAddress}}' daemon\_dave



### 删除容器

命令dock rm。目前没有办法一次删除所有容器，可以如下小技巧来删除全部容器：

docker rm ‘docker ps –a –q’

其中的docker ps命令会列出现有的全部容器，-a代表列出所有容器，而-q表示只需要返回容器的ID而不会返回容器的其他信息。这样就得到了容器id的列表，并传给了docker rm命令，从而达到删除所有容器的目的。

## 使用Docker镜像和仓库

### 4.1 什么是docker镜像 ？？？

### 列出镜像

docker images

镜像从仓库下载下来，镜像保存在仓库中，而仓库存在于Registry中。。每个镜像仓库都可以存放很多镜像，通过标签（tag）进行区分。可以通过在仓库名后面加上一个冒号和标签名来指定仓库中的某一镜像：docker pull Ubuntu:12.04

Dokcer Hub中有两种类型的仓库：用户仓库和顶层仓库。用户仓库的镜像都是有docker用户创建的，而顶层仓库则是由docker内部的人来管理的。用户仓库的命名由用户名和仓库名两部分组成，如user001/puppet

### 拉取镜像

用docker run命令从镜像启动一个容器时，如果该镜像不在本地，docker会先从docker hub下载该镜像。如果没有指定具体的镜像标签，那么docker会自动下载latest标签的镜像。

其实也可以通过docker pull命令先将该镜像拉取到本地。使用docker pull可以节省从一个新镜像启动一个容器所需的时间。

可以使用docker images命令看到新镜像是否已经下载到本地docker宿主机上了。

可以通过在docker images命令后面指定镜像名来查看指定镜像的内容。

### 查找镜像

可以通过docker search命令来查找所有docker hub上公共的可用镜像，如：

docker search puppet

这条命令的返回内容包括：仓库名；镜像描述；用户评价（Starts）；是否官方（Official）;自动构建（Automated）—表示这个镜像是由Docker Hub的自动构建（Automated Build）流程构建的。

### 构建镜像

构建docker镜像有以下两种方法：使用docker commit命令或者使用docker build命令和dockerfile文件。并不推荐使用docker commit命令，而应该使用更灵活、更强大的dockerfile来构建docker镜像。

提示：一般来说，我们不是真正“构建”新镜像，而是基于一个已有的基础镜像， 如ubuntu或Fedora等， 构建新镜像而已。如果真的想从零构建一个全新的镜像，可以参考https://docs.dokcer.com/articles.baseimages/

**创建dokcer hub账号**

网址<https://hub.docker.com/account/signup>，注册账号

要登录到docker hub，可以使用docker login命令

可以使用docker logout命令从一个registry服务器退出。

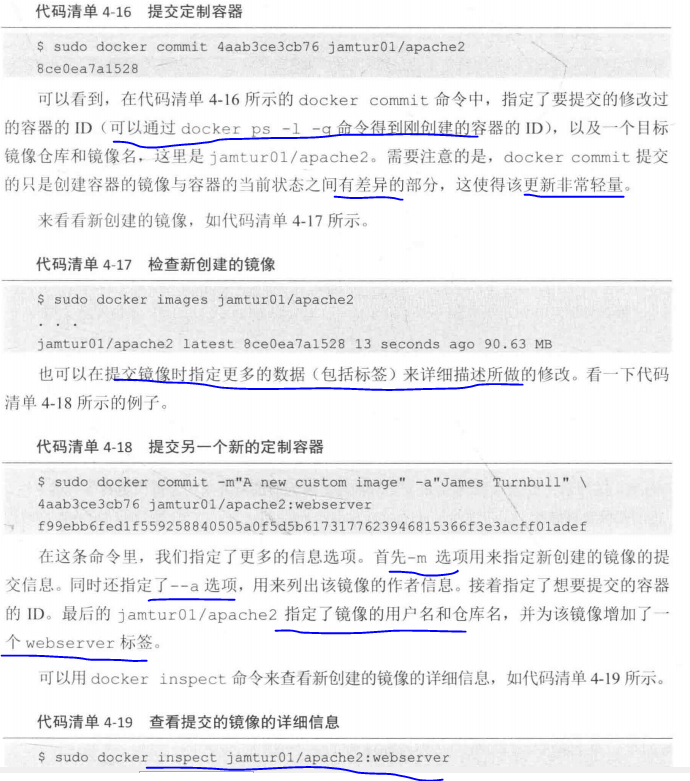
**用docker的commit命令创建镜像**

可以将此想象为是在往版本控制系统里提交变更。先创建一个容器，并在容器里做出修改，就像修改代码一样，最后再将修改提交为一个新镜像。

创建一个要进行修改的定制容器：docker run -i –t Ubuntu /bin/bash

安装vim编辑器：apt-get update apt-get install vim

在为容器安装了vim编辑器后，想把它当前状态保存下来，这样就不必每次都创建一个新容器并在里面安装vim了。为了完成此项工作，需要先使用exit命令从容器退出，之后再运行docker commit命令。



**用dockerfile构建镜像**

Dockerfile使用基本的基于DSL(Domain Specific Language)语法的指令来构建一个Docker镜像，推荐使用dockerfile的定义文件和docker build命令来构建镜像，因为这样更具备可重复性、透明性以及幂等性。

一旦有了dockerfile，我们就可以使用docker build命令基于该dockerfile中的指令构建一个新的镜像。



Dockerfile由一系列指令和参数组成。每条指令，都必须为大写字母，且后面要跟随一个参数，如：FROM Ubuntu:14.04。dockerfile中的指令会按顺序从上到下执行，所以应该根据需要合理安排指令的顺序。

每条指令都会创建一个新的镜像层并对镜像进行提交。Docker大体上按照如下流程执行dockerfile中的指令：

Docker从基础镜像运行一个容器；

执行一条指令，对容器做出修改；

执行类似docker commit的操作，提交一个新的镜像层；

Docker再基于刚提交的镜像运行一个新容器；

执行dockerfile中的下一条指令，直到所有指令都执行完毕

每个dockerfile的第一条指令必须是FROM。FROM指令指定一个已经存在的镜像，后续指令都将基于该镜像进行，这个镜像被称为基础镜像

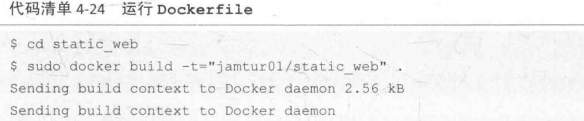
接着指定了MAINTAINER指令，这条指令会告诉Docker该镜像的作者是谁，以及作者的电子邮件地址。

之后，指定了两条RUN指令。RUN指令会在当前镜像中运行指定的命令。像前面说的那样，每条RUN指令都会创建一个新的镜像层，如果该指令执行成功，就会将此镜像层提交，之后继续执行Dockerfile中的下一条指令。



用户可以在运行时以docker run命令通过—expose选项来指定对外部公开的端口。

**基于dockerfile构建新镜像**



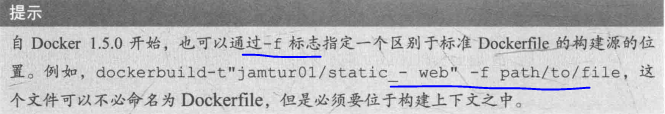
命令中通过-t为新镜像设置了仓库和名称，本例中仓库名为jamtur01，镜像名为static\_web。强烈建议各位为自己的镜像设置合适的名字以方便追踪和管理。也可以在构建镜像的过程中为镜像设置一个标签，其使用方法为“镜像名：标签”，如下：

docker build –t=”jamtur01/static\_web:v1

如果没有制定任何标签，docker将会自动为镜像设置一个latest标签。

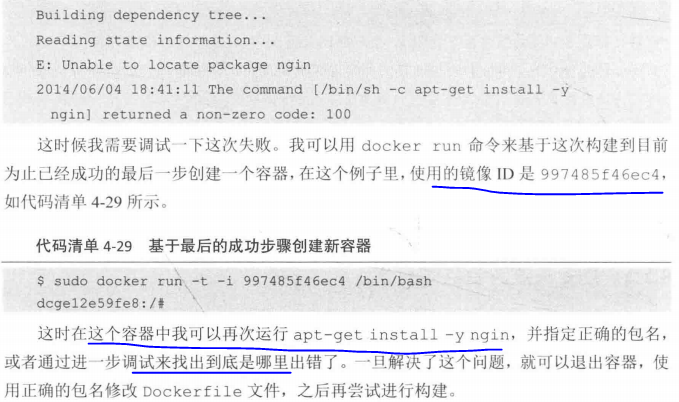
上面最后的 . 告诉docker到本地目录中去找dockerfile文件。也可以指定一个git仓库的源地址来指定dockerfile的位置，如下：

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



**指令失败时会怎样**





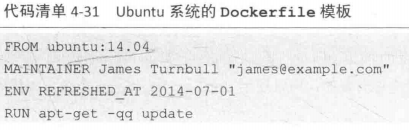
**Dockerfile和构建缓存**

由于每一步的构建过程都会将结果提交为镜像，所以docker的构建镜像过程就显得非常聪明，它会将之前的镜像层看作缓存。比如，在上面的调试例子里，不需要在第一步到第三步之间进行任何修改，因此docker会将之前构建时创建的镜像当做缓存并作为新的开始点。

有些时候需要确保构建过程不会使用缓存，要想忽略缓存功能，可以使用docker build的—no-cache，如：docker build –-no-cache –t=”jamtur01/static\_web”

**基于构建缓存的dockerfile模板**

构建缓存带来的一个好处就是，我们可以实现简单的dockerfile模板（比如在dockerfile文件顶部增加包仓库或者更新包，从而尽可能确保缓存命中）。一般都会在自己的dockerfile文件顶部使用相同的指令集模板，比如对Ubuntu：



通过ENV指令设置了一个名为REFRESHED\_AT的环境变量，这个环境变量用来表示该镜像模板最后的更新时间。

有了这个模板，如果想刷新一个构建，只需修改ENV指令中的日期。这使docker在命中ENV指令时开始重置这个缓存，并运行后续指令而无须依赖该缓存。

**查看新镜像**

可以使用docker images命令。也可以使用docker history命令查看镜像是如何被构建出来的：



**从新镜像启动容器**

docker run –d –p 80 –name static\_web jamtur01/static\_web nginx –g “daemon off;”

使用docker run命令，基于刚才创建的镜像的名字，启动了一个名为static\_web的新容器。-d告诉docker以分离（detached）的方式在后台运行。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web服务器。-p用来控制docker在运行时应该公开哪些网络端口给外部（宿主机）

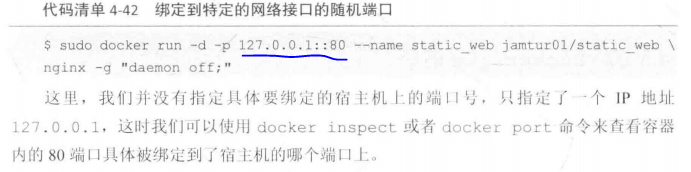
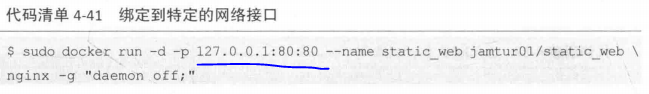
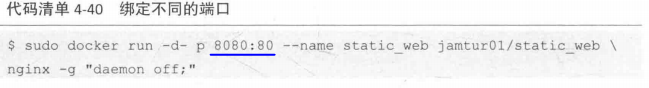
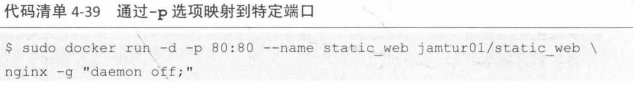
运行一个容器时，docker可以通过两种方法来在宿主机上分配端口：

\*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中的80端口上

\*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。



docker run中的-p选项还为我们在将容器端口向宿主机公开时提供了一定的灵活性。



**Dockerfile指令**

1. CMD

CMD指令用于指定一个容器启动时要运行的命令。有点类似于RUN指令，只是RUN是指定镜像被构建时要运行的命令，而CMD是指定容器被启动时要运行的命令。这和使用docker run命令启动容器时指定要运行的命令非常相似。

docker run –i –t jamtur01/static\_web /bin./true

和在dockerfile中使用cmd指令是等效的

CMD [“/bin/true”]

也可以为要运行的命令指定参数，如 CMD [“/bin/bash”, “-1”]，这里将-1标志传递给了/bin/bash命令  
而且，使用docker run命令可以覆盖CMD命令。Dockerfile中只能指定一条CMD命令，如果指定了多条CMD指令，也只有最后一条CMD指令会被使用。

1. ENTRYPOINT

ENTRYPOINT指令与CMD指令非常类似。ENTRYPOINT指令提供的命令没有那么容器在启动容器时被覆盖。实际上，docker run命令行中指定的任何参数都会被当做参数再次传递给ENTRYPOINT指令中指定的命令

指定ENTRYPOINT指令——ENTRYPOINT [“/usr/sbin/nginx”]

为ENTRYPOINT指令指定参数——ENTRYPOINT[“/usr/sbin/nginx”, “-g”, “daemon off;”]

用户也可以在运行时通过docker run的—entrypoint标志覆盖ENTRYPOINT指令

1. WORKDIR

用来在从镜像创建一新容器时，在容器内部设置一个工作目录，ENTRYPOINT和/或CMD指定的程序会在这个目录下执行。

也可以通过-w在运行时覆盖工作目录，如：docker run –ti –w /var/log ubuntu pwd

1. ENV

ENV指令用来在镜像构建过程中设置环境变量——ENV RVM\_PATH /home/rvm

这个新的环境变量可以在后续的任何RUN指令中使用，这就如同在命令前面指定了环境变量前缀一样。

也可以指定多个变量——ENV RVM\_PATH=/home/rvm RVM\_ARCHFLAGS=”-arch i386”

也可以在其他指令中使用这些环境变量：

ENV TARGET\_DIR /opt/app

WORKDIR $TARGET\_DIR

这些环境变量也会被持久化保存到从这个镜像创建的任何容器中。

也可以使用docker run命令行的-e来传递环境变量，这些变量将只会在运行时有效。

docker run –ti –e “WEB\_PORT=8080” ubuntu env

1. USER

用来指定该镜像会以什么样的用户去运行——USER nginx

也可以在docker run命令中通过-u来覆盖该指令指定的值。

如果不通过USER指令指定用户，默认用户为root

1. VOLUME

用来向基于镜像创建的容器添加卷。一个卷是可以存在于一个或者多个容器内的特定目录，这个目录可以绕过联合文件系统，并提供如下共享数据或者对数据进行持久化的功能：

卷可以在容器间共享和重用；一个容器可以不是必须和其他容器共享卷；对卷的修改是立时生效的；对卷的修改不会对更新镜像产生影响；卷会一直存在直到没有任何容器再使用它。

卷功能让我们可以将数据（如源代码）、数据库或者其他内容添加到镜像中而不是将这些内容提交到镜像中，并且允许我们在多个容器间共享这些内容。可以利用此功能来测试容器和内部的应用程序代码，管理日志，或者处理容器内部的数据库

VOLUME [“/opt/project”]——这条指令将会为基于此镜像创建的任何容器创建一个名为/opt/project的挂载点。

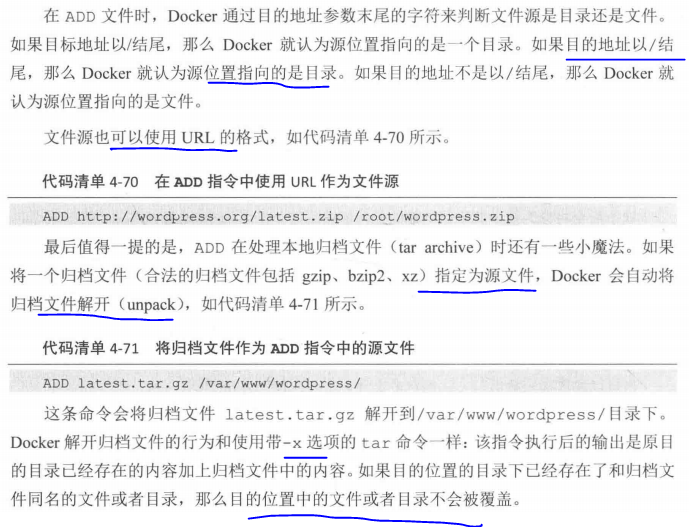
也可以通过指定数组的方式指定多个卷——VOLUME [“/opt/project”,”/data”]

1. ADD

ADD指令用来将构建环境下的文件和目录复制到镜像中，比如在安装一个应用程序时。ADD指令需要源文件位置和目的文件位置两个参数

ADD software.lic /opt/application/software.lic

这里的ADD指令将会构建目录下的software.lic文件复制到镜像中的/opt/applicaton/software.lic。



1. COPY

COPY指令非常类似于ADD，它们根本的不同是COPY只关心在构建上下文中复制本地文件，而不会去做文件提取（extraction）和解压（decompression）的工作。

COPY conf.d /etc/apache2

把本地conf.d目录中的文件复制到/etc/apach2/目录中

1. LABEL

用于为Docker镜像添加元数据。元数据以键值对的形式展现。

LABEL version=”1.0”

LABEL location=”New York” type=”Data Center” role=”Web Server”

可以指定多个元数据，不同的元数据之间用空格分隔。推荐将所有的元数据都放到一条LABEL指令中，以防止不同的元数据指令创建过多镜像层。可以通过docker inspect来查看Docker镜像中的标签信息。

1. ARG

用来定义可以在docker build命令运行时传递给构建运行时的变量，只需要在构建时使用—build-arg即可。

添加ARG指令：ARG build

ARG webapp\_user=user

使用ARG指令：docker build –build-arg build=1234 –t jamtur01/webapp .

这里构建jamtur01/webapp镜像时，build变量将会设置1234，而webapp\_user变量则会继承设置的默认值user

1. ONBUILD

为镜像添加触发器（trigger）。当一个镜像被用做其他镜像的基础镜像时（比如用户的镜像需要从某未准备好的位置添加源代码，或者用户需要执行特定于构建镜像的环境的构建脚本），该镜像中的触发器将会被执行。

ONBUILD ADD . /app/src

ONBUILD RUN cd /app/src && make

ONBUILD指令可以在镜像上运行docker inspect命令来查看。

ONBUILD触发器会按照在父镜像中指定的顺序执行，并且只能被继承一次（也就是说只能在子镜像中执行，而不会在孙子镜像中执行）

提示：有好几条指令是不能用在ONBUILD指令中的，包括FROM、MAINTAINER和ONBUILD本身，是为了防止在dockerfile构建过程中产生递归调用的问题。

### 将镜像推送到Docker Hub

尝试推送root镜像——docker push static\_web

推送docker镜像——docker push jamtur01/static\_web

除了从命令行构建和推送镜像，docker hub还允许定义**自动构建。**只需将github或bitBuild中含有dockerfile文件的仓库连接到docker hub即可。

### 删除镜像

Docker rmi jarmtur01/static\_web

该操作只会将本地的镜像删除。如果之前已经将该镜像推送到docker hub上，需要登录docker hub后使用delete repository链接来删除。

同时删除多个docker镜像

docker rmi jamtur01/apache2 jamtur01/puppetmaster

或者类似于使用docker rm命令那样：docker rmi ‘docker images –a -q’

### 运行自己的Docker Registry

运行基于容器的registry

docker run –p 5000:5000 registry:2

该命令将会启动一个运行registry应用2.0版本的容器，并将5000端口绑定到本地宿主机。

………

………

### 容器之间进行通信

两种比较现实的连接Docker容器的方式是Docker Networking和Dokcer链接（docker link）

**Docker Networking**

创建Docker网络：docker network create app

查看app网络：docker network inspect app

docker network ls

在docker网络中创建redis容器：

docker run –d –-net=app –-name db jamtur01/redis

添加已有容器连接到docker网络：docker network connect app db2

从网络中断开一个容器：docker network disconnect app db2

**通过docker链接连接容器**

启动一个redis容器

docker run –d -- name redis jamtur01/redis

链接redis容器

docker run –p 4567 --name webapp --link redis:db -t -i -v $PWD/webapp\_redis:/opt/webapp jamtur01/Sinatra /bin/bash

其中的--link 创建了两个容器间的客户-服务链接，两个参数：一个是要链接的容器的名字，另一个是链接的别名

## 不使用SSH管理Docker容器

**可以借助一个叫nsenter的小工具，在需要时登入容器**。

nsenter让我们可以进入Docker用来构成容器的内核命名空间。从技术上说，这个工具可以进入一个已经存在的命名空间，或者在新的一组命名空间里执行一个进程。简单来说，使用nsenter可以进入一个已经存在的容器的shell，即便这个容器没有运行ssh或者任何类似目的的守护进程。

可以通过docker容器安装nsenter：

docker run -v /usr/local/bin:/target jpetazzo/nsenter

为了使用nsenter，首先要拿到进入的容器的进程ID（PID）。可以使用docker inspect命令获得PID：docker inspect -f {{.State.Pid}} ce59ee387c33

可以通过：docker ps 查看容器id

使用nsenter进入容器：nsenter --target 14446 --mount --uts --ipc --net –pid（14446为得到的pid）

这会在容器里启动一个shell，而不需要ssh或者其他类似的守护进程或者进程。还可以将想在容器内执行的命令添加在nsenter命令行的后面

nsenter --target 14446 --mount --uts --ipc --net –pid ls

这会在目标容器内执行ls命令。

## 集群管理工具：docker swarm

Docker swarm是一个原生的docker集群管理工具。Swarm将一组docker主机作为一个虚拟的docker主机来管理。Swarm有一个非常简单的架构，它将多台docker主机作为一个集群，并在集群级别上以标准docker api的形式提供服务。

### 安装swarm

安装swarm最简单的方法就是使用docker自己。要想支持swarm，用户的所有docker主机都必须在1.4.0或者更高版本之上。此外，运行swarm的所有docker节点也都必须运行着同一个版本的docker。

举例：

在主机smoker和主机joker上安装swarm。其中smoker的主机ip是10.0.0.125，joker的主机ip是10.0.0.135.

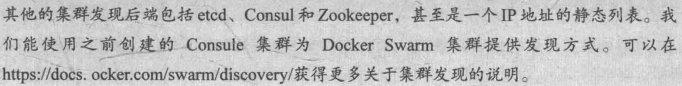
在smoker上拉取docker swarm镜像：docker pull swarm

查看swarm镜像是否下载成功：docker images swarm

### 创建swarm集群

在两台主机上下载了swarm镜像，之后就可以创建swarm集群了。集群中的每台主机都运行着一个swarm节点代理。每个代理都将该主机上的相关docker守护进程注册到集群中。和节点代理相对的是swarm管理者，用于对集群进行管理。

集群注册可以通过多种可能的集群发现后端（discovery backend）来实现。默认的集群发现后端是基于docker hub。**它允许用户在docker hub中注册一个集群，然后返回一个集群id，我们之后可以使用这个集群id向集群添加额外的节点**



1、使用默认的docker hub作为集群发现服务创建一个swarm集群。在smoker主机上创建swarm集群：docker run –rm swarm create

该命令会返回一个字符串，代表集群id。可以利用这个id向swarm集群中添加节点。

2、接着在每个节点上运行swarm代理。比如在smoker上运行swarm代理：

docker run -d swarm join –addr=10.0.0.125:2375 token://fajflaskd30nfalks03390i45klnf

这里通过传递给容器的join标志，通过-addr选项传递的本机ip地址，以及代表集群id的token，启动一个代理。每个代理都会绑定到它们所在主机的ip地址上。每个代理都会加入swarm集群中。

可以通过查看代理容器的日志来了解代理内部是如何工作的。查看smoke代理的日志：

docker logs fjldf56kllef65

从中可以看到，代理每隔25秒就会向发现服务进行注册。这将告诉发现后端docker hub该代理可用，该docker服务器也可以被使用。

列出swarm节点：docker run --rm swarm list token://fjsaldkfjlkoweui3498vn32324r

会显示如下结果：10.0.0.125:2375 10.0.0.135:2375

这个指令运行了swarm镜像，并制定了list标志以及集群的token。返回结果为集群中所有节点的列表。

启动swarm集群管理者，通过swarm集群管理者来对集群进行管理：

docker run -d -p 2380:2375 swarm manage token://fjsaldkfjlkoweui3498vn32324r

这将创建一个新容器来运行swarm集群管理者。同时将2380端口映射到了2375端口。2375是docker的标准端口，将使用这个端口来和标准docker客户端或者api进行交互。通过指定manager选项来启动管理者，还指定了集群id。然后就可以通过这个管理者来向集群发送命令了。

docker -H tcp://localhost:2380 info

除了标准的docker info输出之外，swarm还输出了所有节点信息，可以看到每个节点、节点的IP地址、每台节点上有多少容器在运行，以及CPU和内存这样的容量信息。

### 创建容器

通过一个shell循环操作来创建6个Nginx容器：

for i in ‘seq 1 6’;do docker -H tcp://localhost:2380 run -d --name www-$i -p 80 ngnix;done

这里运行了包装在一个shell循环里的docker run命令。通过-H选项为Docker客户端指定了tcp://localhost:2380地址，也就是swarm管理者的地址。并且告诉docker以守护方式启动容器，并将容器命名为www-加上一个循环变量$i。这些容器都是基于ngnix镜像创建的，并都打开了80端口。

执行docker -H tcp://localhost:2380 ps。它不是在本地docker守护进程中，而是跨swarm集群运行的。从结果中可以看到有6个容器在运行，平均分配在集群的两个节点上。

Swarm根据过滤器（filter）和策略（strategy）的结合来决定在哪个节点上运行容器。

### 过滤器

**1.约束过滤器**

约束过滤器依赖于用户给各个节点赋予的标签。举例来说，用户想为使用特殊存储类型或者指定操作系统的节点来分组。约束过滤器需要在启动docker守护进程时，设置键值对标签，通过-label标注来设置

运行docker守护进程时设置约束标签：docker daemon --label datacenter=us-east1

启动容器时指定约束过滤器：docker -H tcp://localhost:2380 run -e constraint:datacenter=us-east1 -d –name www-use1 -p 80 nginx

这里启动了一个名为www-use1的容器，并通过-e选项指定约束条件，这里用来匹配datacenter=us-east1。这样将会在设置了这个标签的docker守护进程中启动该容器。这个约束过滤器支持相等匹配==和不等匹配!=，也支持使用正则表达式：

docker -H tcp://localhost:2380 run -e constraint:datacenter=us-east\* -d –name www-use1 -p 80 nginx

**2.亲和过滤器**

亲和过滤器让容器运行更互相接近，比如让容器web1挨着haproxy1容器或者挨着指定id的容器运行：

docker run -d --name www-use2 -e affinity:container==www-use1 nginx

通过亲和过滤器启动了一个容器，并告诉这个容器运行在www-use1容器所在的swarm结点上。

也能匹配已经拉取了指定镜像的节点，如affinity:image==ngnix将会让容器在任何已经拉取了nginx镜像的节点上运行。也可以像约束过滤器一样，通过按名字或者正则表达式来搜索容器来匹配特定的节点。

**3.依赖过滤器**

在具备指定卷或者容器链接的节点上启动容器

4.端口过滤器

通过网络端口进行调度，在具有指定端口可用的节点上启动容器

Docker -H tcp://localhost:2380 run -d --name haproxy -p 80:80 haproxy

**5.健康过滤器**

利用健康过滤器，swarm就不会将任何容器调度到被认为不健康的节点上。通常来说，不健康是指swarm管理者或者发现服务报告某集群节点有问题。

可以通过为swarm manage命令传递-filter标志来控制哪些过滤器能用。

### 策略

策略允许用户用集群节点更隐式的特性来对容器进行调度，比如该节点可用资源的数量等，只在拥有足够内存或者cpu的节点上启动容器。Docker swarm现在有3种策略：平铺（spread）策略、紧凑（binpacking）策略和随机（random）策略。默认的策略是平铺策略。

可以在执行swarm manage命令时，通过-strategy标志设置用户想选用的策略。

## 问题记录

1. 在docker容器Ubuntu运行时，想要其中安装软件，但是国外源下载不到资源。在需要更改国内源时，又没有安装vi、vim命令，没法通过编辑的方式更改/etc/apt/sources.list文件。这时可以通过追加符的方式往sources.list中添加内容，如下

mv /etc/apt/sources.list /etc/apt/sources.list.bak

echo "deb http://mirrors.163.com/debian/ jessie main non-free contrib" >> /etc/apt/sources.list

#更新安装源

apt-get update

国内源参考：<https://www.jianshu.com/p/eb9bd6142c71>

最好把原来的内容注释掉。

然后就可以通过apt-get install vim安装vim了

2. 这里介绍如何把CentOS默认镜像源更换为阿里云镜像源

<https://www.cnblogs.com/Chan94/p/10658756.html>