Docker

A container is a running instance of an image

Install Docker

install virtualBox and Ubuntu

refer to: <https://docs.docker.com/linux/step_one/>

ctrl + alt + t打开终端

$ curl -fsSL https://get.docker.com/ | sh

reboot Ubuntu

$docker run helloworld

若无法connection

$sudo gedit /etc/default/docker

export http\_proxy=”<http://HOST:PORT/>”

或者命令行

export http\_proxy=http://165.225.96.34:10015/

export https\_proxy=https://165.225.96.34:10015/

$sudo service docker restart //restart the docker service

docker: 告诉操作系统使用docker程序

run：docker的子命令创建与运行Docker容器

hello-world: 告诉Docker将hello-world 镜像装入到容器里

Build your own image

1. Write a Dockerfile

FROM docker/whalesay:latest

RUN apt-get -y update && apt-get install -y fortunes

CMD /usr/games/fortune -a | cowsay

1. Build an image from your Dockerfile

$docker build – t docker-whale .

（Note: 若无法connection，改为

$docker build - -build-arg HTTP\_PROXY= <http://165.225.96.34:10015/> -t docker-whale .

1. Run your image

$docker run docker-whale

Tag & push your image to DockerHub

$ docker tag imageID qizhonglin/docker-whale:latest

$ docker login - - username=qizhonglin - - [email=qizhong.lin@philips.com](mailto:email=qizhong.lin@philips.com)

$ docker push qizhonglin/docker-whale

自动构建

只需要将GitHub or BitBucket中含有Dockerfile文件的仓库连接到Docker Hub即可。向这个代码仓库推送代码时，将会触发一次镜像构建活动并创建一个新镜像。

Docker Hub -> Add Repository -> Automated Build -> GitHub -> Public and Private -> select projects

不能通过docker push命令推送一个自动构建，只能通过更新你的GitHub or BigBucket仓库来更新你的自动构建。

<第一本Docker书>

$docker info

该命令返回所有容器和镜像的数量，Docker使用的执行驱动和存储驱动以及Docker的基本配置

$docker run –it ubuntu /bin/bash

该命令提供Docker容器的创建到启动的功能。

- i 标志保证容器中STDIN是开启的

- t 标志告诉Docker为要创建的容器分配一个伪tty终端

ubuntu镜像告诉Docker基于ubuntu镜像来创建容器，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

/bin/bash告诉Docker在新容器中运行的命令，在这里指启动一个bash shell

[root@de28...:/#](mailto:root@de28...:/) apt-get updata && apt-get install vim

若无法连接网络，需要设置

export http\_proxy=http://165.225.96.34:10015/

export https\_proxy=https://165.225.96.34:10015/

$docker ps –a

查看当前系统中容器的列表

$docker run - - name bob\_the\_container - i - t ubuntu /bin/bash

--name: 指定容器名称，可以用容器的名称来替代容器ID,容器的命名必须是唯一的。

$docker start bob\_the\_container

启动容器

$docker attach bob\_the\_container

重新附着到该容器的会话上

$docker rmi - f imageID

or $docker rmi - f bob\_the\_container

除了交互式运行的容器(interactive container),我们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

$docker run - - name daemon\_dave -d ubuntu /bin/sh –c “while true; do echo hello world; sleep 1; done”

-d标识告诉Docker将容器入到后台运行

$docker ps

查看正在运行的容器

$docker logs –ft daemon\_dave

获取容器的日志

-f监控Docker的日志，这与tail –f 命令非常相信

$docker top daemon\_dave

查看容器内部运行的进程

$docker exec –d daemon\_dave touch /etc/new\_config\_file

-d: 标志表明运行一个后台进程

daemon\_dave: 指定的容器

touch /etc/new\_config\_file: 执行的命令

$docker exec -it daemon\_dave /bin/bash

$docker run - -restart=always –name daemon\_dave –d Ubuntu /bin/sh –c “while true;…”

restart=always标志表明无论容器的退出代码是什么，Docker都会自动重启该容器

restart=on-failure标志表明只有当容器的退出代码为非0值的时候，才会自动重启

$docker inspect daemon\_dave

该命令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

$docker rm daemon\_dave

删除容器

$docker rm `docker ps –a –q`

删除所有容器

docker ps命令会列出现有的全部容器

-a标志代表列出所有容器包括运行和停止

-q标志则表示只需要返回容器的ID而不会返回容器的其他信息

使用Docker镜像和仓库

Docker镜像是由文件系统叠加而成，最底端是一个引导文件系统bootfs,第二层是root文件系统rootfs,它位于引导文件系统之上，rootfs可以是一种或多种操作系统如Debian or Ubuntu。

Docker将这样的文件系统称为镜像。一个镜像可以放到另一个镜像的顶部。位于下面的镜像称为父镜像，可以依次类推，直至镜像栈的最底部，最底部的镜像称为基础镜像。最后，当从一个镜像启动容器时，Docker会在该镜像的最顶层加载一个读写文件系统。我们想在Docker中运行的程序就是在这个读写层中执行（见page: 50）

当Docker第一次启动一个容器时，初始的读写层是空的。当文件系统发生变化时，这些变化都会应用到这一层上。比如如果想修改一个文件，这个文件首先会从该读写层下面的只读层复制到该读写层。该文件的只读版本依然存在，但是已经被读写层中的该文件副本所隐藏。

这种机制称为写时复制copy on write.每个只读镜像层都是只读的，并且以后永远不会变化。当创建一个新容器时，Docker会构建出一个镜像栈，并在栈的最顶端添加一个读写。这个读写层再加上其下面的镜像层以及一些配置数据，就构成了一个容器。

$docker images

列出镜像列表，本地镜像都保存在Docker宿主机的/var/lib/docker目录下。可以在/var/lib/docker/container目录下面看到所有的容器

远程镜像，保存在仓库Registry，即Docker Hub

$docker search puppet

查找所有Docker Hub上公共的可用镜像

$docker pull 用户名/镜像名:tag

拉取顶层仓库或用户仓库中的所有内容，用户仓库镜像是由Docker用户创建，顶层仓库是由Docker公司和选定的厂商创建

$docker run –it --name next\_container Ubuntu /bin/bash

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。

构建镜像 = Dockerfile + docker build

一般基于一个已有的基础镜像，如ubuntu,构建新镜像

Dockerfile使用基本的基于DSL语法的指令来构建一个Docker镜像

目录结构

static\_web

Dockerfile

Static\_web目录就是我们的构建环镜build environment, Docker则称此环境为上下文context or 构建上下文.Docker会在构建镜像时将构建上下文和该上下文中的文件和目录上传到Docker守护进程。这样Docker守护进程就能直接访问你想在镜像中存储的任何代码、文件或者其他数据

Dockerfile

# Version: 0.0.1

FROM ubuntu:14.04

MAINTAINER QiZhong Lin [qizhong.lin@philips.com](mailto:qizhong.lin@philips.com)

RUN apt-get update

RUN apt-get install –y nginx

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

EXPOSE 80

#开头的行都会被认为是注释

Docker中的指令会按顺序从上而下执行，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，直至所有指令都执行完毕

如果你的Dockerfile由于某些原因如某条指令失败了没有正常结束，那么你将得到一个可以使用的镜像。这对调试非常有帮助

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

MAINTAINER指令：告诉Docker该镜像的作者是谁，以及作者的电子邮件地址。

RUN指令：在当前镜像中运行指定的命令。默认情况下，RUN指令会在shell里使用命令包装器/bin/sh –c来执行。

EXPOSE指令：告诉Docker该容器内的应用程序将会使用容器的指定端口。这并不意味着可以自动访问任意容器运行中服务的端口。出于安全考虑，Docker并不会自动打开该端口，而是需要你在使用docker run运行容器时来指定需要打开哪些端口。

$docker build –t=”qizhonglin/static\_web:v0” .

Dockerfile中的所有指令都会被执行并且提交，并且在该命令成功结束后返回一个新镜像

-t选项为新镜像设置了仓库和名称

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。

如果在构建上下文的根目录下存在以.dockerignore命名的文件的话，那么该文件内容会被按行进行分割，每一行都是一条文件过滤匹配模式。非常像.gitignore文件。该文件用来设置哪些文件不会被上传到构建上下文中去。

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。

$docker history qizhonglin/static\_web:v0

查看镜像是如何构建出来

$docker run –d –p 80 –name static\_web qizhonglin/static\_web:v0 nginx –g “daemon off;”

从新镜像启动容器

-d选项告诉Docker以分离detached的方式在后台运行。这种方式非常适合运行类似nginx守护进程这样的需要长时间运行的过程。

nginx –g “daemon off;” 容器里运行的命令，表明将以前台运行的方式启动nginx作来web服务器

-p标志用来控制Docker在运行时应该公开哪些网络端口给外部（宿主机）

$docker ps -l

查看容器中的80端口被映射到宿主机的随机端口号

$docker run –d –p 8080:80 --name static\_web qizhonglin/static\_web:v0 nginx –g “daemon off;”

将容器80端口映射到宿主机的指定端口8080

$curl localhost:映射端口号

Dockerfile指令

RUN指令：指定镜像被构建时要运行的命令

CMD指令：指定容器被启动时要运行的命令

举例：

$docker run –it qizhonglin/static\_web:v0 /bin/true

等价于

CMD [“/bin/true”]

docker run命令可以覆盖CMD指令。如果我们在Dockerfile里指定了CMD指令，而同时在docker run命令行中也指定了要运行的命令，命令行中指定的命令会覆盖Dockerfile中的CMD指令。

在Dockerfile中只能指定一条CMD指令。如果指定了多条CMD指令，也只最后一条CMD指令会被使用。

ENTRYPOINT指令：与CMD指令类似，但不容易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。

WORKDIR指令：从镜像创建一个新容器时，在容器内部设置一个工作目录，ENTRYPOINT和CMD指定的程序会在这个目录下执行

ENV指令：用来在镜像构建过程中设置环境变量

USER指令：用来指定该镜像会以什么样的用户去运行

VOLUMN指令：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的目录，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
* 卷会一直存在直至没有任何容器再使用它

卷功能让我们可以将数据（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。我们可以利用此功能来测试容器和内容的应用程序代码，管理日志，或者处理容器内部的数据库

ADD指令：将构建环境下的文件和目录 复制 到镜像中

COPY指令：构建环境下的文件和目录 复制 到镜像中，但不会做文件提取和解压工作

注意：复制时，文件源路径必须是一个与当前构建环境相对的文件或者目录，本地文件都放到和Dockerfile同一个目录下，不能复制该目录之外的任何文件。

ONBUILD指令：为镜像添加触发器。当一个镜像被用做其他镜像的基础镜像时，该镜像中的触发器将会被执行 。触发器会在构建过程中插入新指令，我们可以认为这些指令是紧跟在FROM之后指定的。触发器可以是任何构建指令

$docker rmi qizhonglin/static\_web:v0

删除一个镜像

该操作只会删除本地的镜像，如果想删除Docker Hub上的镜像仓库，DockerHub -> Delete repository

$docker rmi `docker images –a –q`

运行自己的Docker Registry，见page90