**Dockerfile(简单介绍)**

**一个简单的dockerfile，配置nginx服务器**

FROM ubuntu14:04 FROM:指定基于的基础镜像。Docker镜像基于已创建的镜像。指令格式为<镜像名称>:<标签> MAINTAINER Foo Bar <foo@bar.com> MAINTAINER:维护者的信息 RUN apt-get update RUN:运行shell脚本或命令 RUN apt-get install -y nginx RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf RUN chown -R www-data:www-data /var/lib/nginx VOLUME ["/data", "/etc/nginx/site-enabled", "/var/log/nginx"] VOLUME:要与主机共享的目录。也可以在docker run中使用-v的选项进行设置。例如，-v /root/data:/data将主机的/root/data目录连接到Docker容器的.data目录。 WORKDIR /etc/nginx EXPOSE 80 EXPOSE 443

**使用build创建镜像**

$sudo docker build --tag hello:0.1 . 命令格式为 docker build <选项><路径>。使用--tag可以设置镜像的名称和标签，若只设置镜像名称，标签自动为latest。 $sudo docker run --name hello-nginx -d -p 80:80 -v /root/data:/data hello:0.1 -d 选项在后台运行容器 -p 80：80 选项将主机的80号端口与容器的80号端口连接起来，并暴露在外部。这样设置后连接http：//<主机ip>:80 就可以连接到容器的80端口 -v /root/data:/data 选项将主机的 /root/data 目录连接到容器的 /data 目录。若将文件放入 /root/data 目录，则能从容器读取相应的文件

**查看Docker**

**使用history命令查看镜像命令历史**

$sudo docker history hello:0.1 命令格式为 docker history<镜像名称(ID)>:<标签>

**使用cp复制文件**

sudo docker cp hello-nginx:/etc/nginx/nginx.conf ./ 命令格式为docker cp<容器名称>:<路径><主机路径> 上述操作为将hello-nginx容器中的nginx文件复制到当前路径下

**使用commit命令从容器的修改中创建镜像**

docker commit 命令可以从容器的修改中创建新的镜像 假设 hello-nginx容器中的文件内容发生变化，将容器创建为镜像文件sudo docker commit -a "Foo Bar <foo@bar.com>" -m "add hello.txt" hello-nginx hello:0.2 命令格式为 docker commit <选项><容器名称(ID)><镜像名称>:<标签> -a "Foo Bar [foo@bar.com](mailto:foo@bar.com)" 与 -m "add hello.txt" 用于设置提交的用户与注册信息。将hello-nginx容器创建为 hello:0.2 镜像。

**使用diff命令检查容器文件的修改**

sudo docker diff hello-nginx 命令格式为docker diff<容器名称(ID)>，A为添加文件，C为修改文件，D为删除文件

**使用inspect命令查看详细信息**

sudo docker inspect hello-nginx 命令格式为docker inspect<镜像或容器名称(ID)>

**Docker 服务进阶**

**搭建Docker私有仓库**

sudo docker pull registry:latest sudo docker run -d -p 5000:5000 --name hello-registry \ -v /tmp/registry:/tmp/registry \ registry 运行后，经i想文件存储到主机的 /tmp/registry 的目录

**使用push命令上传镜像**

sudo docker tag hello:0.1 localhost:5000/hello:0.1 sudo docker push localhost:5000/hello:0.1 创建标签的命令格式为docker tag <镜像名称>:<标签>/<镜像名称>:<标签> 上传镜像的命令格式为 docker push /<镜像名称>:<标签>。 向私有仓库上传镜像时，需要先创建标签。使用docker tag 命令为hello:0.1 镜像创建标签 localhost：5000/hello:0.1, 然后使用docker push 命令将localhost:5000/hello:0.1 镜像上传到私有仓库 下面就可以使用其他的服务器下载镜像 sudo docker pull 192.168.0.39:5000/hello:0.1

**存储镜像数据到Amazon S3**

下载docker镜像 sudo docker pull registry:latest 以容器运行registry:latest镜像$sudo docker run -d -p 5000:5000 --name s3-registry \ -e SETTINGS\_FLAVOR=s3 \ -eAWS\_BUCKET=examplebucket10 \ -e STORAGE\_PATH=/registry \ -e AWS\_KEY=AKIABCDEFGHIJKLMNOPQ \ -e AWS\_SECRET=sF4321dddd \ registry SETTINGS\_FLAVOR:镜像存储方法 设置为s3 AWS\_BUCKET:存储镜像数据的S3 bucket名称。 示例中设置为 examplebucket10 STORAGE\_PATH:镜像数据存储路径。。设置为/registry。 AWS\_KEY:设置AWS访问密钥 AWS\_SECRET:设置AWS秘密密钥 下面，将DOcker镜像推送到s3-registry仓库后，镜像数据即存储到S3 bucket

**使用默认认证**

基本用不到，在此不再赘述

**连接Docker容器**

连接Docker容器时，要在docker run命令中使用--link选项。首先以容器运行DB镜像，此处要使用MongoDB sudo docker run --name db -d mongo --name db 将容器的名字设置为db 创建web容器，与db容器相连接。使用nginx镜像创建作为web服务的容器 sudo docker run --name web -d -p 80:80 --link db:db nginx 连接选项的命令格式为--link<容器名称>:<别名> db与web相连接可以在web容器内使用<别名>:<端口号>进行连接

**连接到其他服务器的Docker容器**

**使用Docker数据卷**

**使用Docker数据卷容器**

**在Docker中运行Docker**

先从github下载Dockerfile 与bash脚本 git clone https://github.com/pyrasis/dind.git 创建镜像 cd dind sudo docker build --tag dind . 等待后创建镜像 sudo docker run -i -t --privileged dind --privileged 获取主机的所有linux内核功能 sudo docker run -i -t --privileged -e LOG=file dind -e LOG=file 不输出日志 在Docker容器中运行Docker sudo docker run -i -t busybox:latest /bin/sh 按照主机->dind容器->busybox容器顺序执行

**详细了解Dockerfile**

**FROM**

From用于设置以那种方式为基础创建镜像，由于使用Dockerfile创建镜像时总以已有镜像为基础，所以必须跟From命令，下载时可以同时使用镜像名称+标签，如果没有标签，默认为latest 命令格式为 FROM <镜像>:<标签 >或 FROM <镜像>。 该命令必须为第一条命令，在创建镜像时，如果本地有相应的镜像，则直接使用，否则从Docker Hub进行下载。一个Dockerfile可以存在多个FROM命令，多个FROM命令创建多个镜像。若果使用了--tag的选项，则只会使用最后一个FROM的命令。

**MAINTAINER**

MAINTAINER用于设置镜像创建者的信息，格式比较自由可以省略

**RUN**

RUN用于在FROM中设置的镜像上运行的脚本或命令。RUN的运行结果会生成新的镜像，其详细信息会保存在镜像的历史中 RUN apt-get install nginx 使用格式为 RUN <命令>，可以使用shell的脚本语句 RUN ["apt-get","install","-y","nginx"] 没有shell的时候，可以使用RUN配置可执行文件加参数的形式进行操作 RUN的执行结果会被缓存，在dockerfile中使用--no-cache可以取消缓存

**CMD**

CMD在一次Dockerfile只能使用一次,与RUN具有同样的属性，可以使用shell或者直接运行参数设置 使用ENTRYPOINT时 ENTRYPOINT ["echo"] CMD ["hello"] 如果Dockerfile中存在ENTRYPOINT时，CMD只要传入参数即可使用，对应的操作使用ENTRYPOINT中的执行运行

**ENTRYPOINT**

ENTRYPOINT用于设置容器启动时运行的脚本或命令，即使用docker run或docker start 命令启动时运行，Dockerfile中，ENTRYPONT智能使用一次，属性与CMD、RUN类似。 区别： CMD sudo docker build --tag example . sudo docker run example echo world docker run 命令中设置的<可执行文件>与Dockerfile的CMD具有相同的功能，此时不输出CMD的操作 ENTRYPOINT sudo docker build --tag example . sudo docker run example echo world 此时同时输出ENTRYPOINT的内容与docker run的内容 ENTRYPOINT也可以docker run的时候使用，--entrypoint进行设置 sudo docker run --entrypoint="cat" example /etc/hostname 如果设置了--entrypoint，则忽略Dockerfile中的entrypoint

**EXPOSE**

EXPOSE用于设置与主机相连的端口号，与docker run 命令中的 --expose 选项功能一致。 EXPOSE 443 EXPOSE 80 或者 EXPOSE 80 443 EXPOSE只使用于于主机进行连接，并不对外暴露，使用docker run -p或-P设置对外暴露的接口

**ENV**

ENV用于设置环境变量。使用ENV设置的环境变量应用于RUN、CMD、ENTRYPOINT。 ENV GOPATH /go ENV PATH /go/bin:$PATH 使用格式为ENV<环境变量><值>。使用环境变量时要使用$符号。 docker run的使用格式为-e <环境变量>=<值>。可多次使用与--env选项相同

**ADD**

ADD用于向镜像添加文件 ADD hello.zip / ADD \*.txt /root/ 使用格式为 ADD <要复制文件的路径(相对路径)><文件在镜像中的路径(目标路径、绝对路径)> 要复制的文件路径要以上下文目录为基准，不能使用上下文之外的文件、目录或绝对路径 要复制的文件不仅可以是文件，还可以是目录，设置目录时，会复制目录下所有的文件 要复制的文件也可以是网络文件的URL 文件在镜像中的路径必须设置为绝对路径。如果路径以/结尾，则创建目录并将文件复制到目录中 解压缩文件

**COPY**

COPY用于向镜像添加文件。与ADD不同，使用COPY添加压缩文件时，不会解压缩，也不能使用文件URL。 COPY \*.txt /root/ 使用格式为 COPY <要复制的文件的路径(相对路径)> <文件在镜像中的路径(目标路径、绝对路径)>

**VOLUME**

VOLUME设置用于将目录下的内容存储到主机而非容器 VOLUME /data VOLUME ["/data","/var/log/hello"] 使用格式为VOLUME<容器目录>或VOLUME["容器目录1","容器目录2"]。但是，使用VOLUME不能与主机的特定目录连接。 若要连接数据卷与主机的特定目录，则必须在 docker run 命令中使用 -v 选项。 sudo docker run -v /root/data:/data example 选项格式为 -v <主机目录>:<容器目录>

**USER**

USER用于设置运行命令的用户账号，该用户会应用于 RUN,CMD,ENTRYPOINT。 USER nobody 使用格式 USER<账户用户名> USER后面的所有RUN、CMD、ENTRYPOINT都会得到应用,中间可以设置其他用户更改用户。

**WORKDIR**

WORKDIR 用于设置执行RUN、CMD、ENTRYPOIINT的目录 WORKDIR /var/www 使用格式为 WORKDIR <路径> 属性与USER相同。

**ONBUILD**

将当前镜像作为基础镜像创建其他镜像时，ONBUILD指令用于设置一些要触发的操作。ONBUILD指定的命令在构建镜像时并不执行，而是在子镜像中执行。换言之，ONBUILD中定义的指令会在用于生成其他镜像的Dockerfile文件的FROM之后执行。 ONBUILD RUN touch /hello.txt ONBUILD ADD world.txt /world.txt 指令格式为 ONBUILD 。除MAINTAINER、ONBUILD之外的所有Dockerfile指令都可以用于ONBUILD指令。

知乎写个东西真难，回去接着用我的Gayhub了，太不友好了。