链滴

docker base

作者：SmiteLi

原文链接：https://ld246.com/article/1568615120338

来源网站：链滴

一

docker pull [选项] [Docker Registry 地址[:端口号]/]仓库名[:标签]
$ docker pull ubuntu:16.04
上面的命令中没有给出 Docker 镜像仓库地址，因此将会从 Docker Hub 获取镜像。而镜像名称是 ubuntu:16.04 ，因此将会获取官方镜像 library/ubuntu 仓库中标签为 16.04 的镜像。

stop all the docker container:
docker stop `docker ps | grep -v 'IMAGE' | awk '{print $1}'`
docker stop `docker container ls -q`
d~~~~ocker start `docker ps -aq`

由于新旧镜像同名，旧镜像名称被取消，从而出现仓库名、标签均为 <none> 的镜像。
$ docker image ls -f dangling=true
remove the danging image:
$ docker image prune

我们希望看到在 mongo:3.2 之后建立的镜像，可以用下面的命令：
$ docker image ls -f since=mongo:3.2

二

docker image ls -q

因此当我们使用上面命令删除镜像的时候，实际上是在要求删除某个标签的镜像。所以首先需要做的是将满足我们要求的所有镜像标签都取消，这就是我们看到的 Untagged 的信息。因为一个镜像可以对应多个标签，因此当我们删除了所指定的标签后，可能还有别的标签指向了这个镜像，如果是这种情况，那么 Delete 行为就不会发生。所以并非所有的 docker rmi 都会产生删除镜像的行为，有可能仅仅是取消了某个标签而已。

当该镜像所有的标签都被取消了，该镜像很可能会失去了存在的意义，因此会触发删除行为。镜像是多层存储结构，因此在删除的时候也是从上层向基础层方向依次进行判断删除。镜像的多层结构让镜像复用变动非常容易，因此很有可能某个其它镜像正依赖于当前镜像的某一层。这种情况，依旧不会触发删除该层的行为。直到没有任何层依赖当前层时，才会真实的删除当前层。这就是为什么，有时候会奇怪，为什么明明没有别的标签指向这个镜像，但是它还是存在的原因，也是为什么有时候会发现所删除的层数和自己 docker pull 看到的层数不一样的源。

比如，我们需要删除所有仓库名为 redis 的镜像：
$ docker image rm $(docker image ls -q redis)
或者删除所有在 mongo:3.2 之前的镜像：
$ docker image rm $(docker image ls -q -f before=mongo:3.2)

Dockerfile 中每一个指令都会建立一层
每一个 RUN 的行为，
就和刚才我们手工建立镜像的过程一样：新建立一层，在其上执行这些命令，执行结束后， commit 一层的修改，构成新的镜像。

三、Dockerfile指令

Dockerfile指令：CMD
在指令格式上，一般推荐使用 exec 格式，这类格式在解析时会被解析为 JSON 数组，因此一定要使用双引号 " ，而不要使用单引号。
exec 格式： CMD ["可执行文件", "参数1", "参数2"...]

ENTRYPOINT:
当指定了 ENTRYPOINT 后， CMD 的含义就发生了改变，不再是直接的运行其命令，而是将 CMD 的内容作为参数传给 ENTRYPOINT 指令，换句话说实际执行时，将变为：
<ENTRYPOINT> "<CMD>"

之前我们说过，跟在镜像名后面的是 command ， 运行时会替换 CMD 的默认值。因此这里的 -i 替了原来的
CMD ， 而不是添加在原来的 curl -s http://ip.cn 后面。而 -i 根本不是命令，所以自然~~~~找不到

这是因为当存在 ENTRYPOINT 后， CMD 的内容将会作为参数传给 ENTRYPOINT ， 而这里 -i 就是新的 CMD ， 因此会作为参数传给 curl ， 从而达到了我们预期的效果。

之前说过每一个 RUN 都是启动一个容器、执行命令、然后提交存储层文件变更

USER 指令和 WORKDIR 相似，都是改变环境状态并影响以后的层。 WORKDIR 是改变工作目录， USER 则是改变之后层的执行 RUN , CMD 以及 ENTRYPOINT 这类命令的身份。