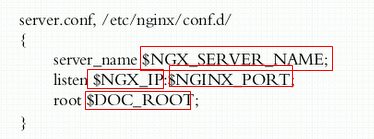


自制镜像的方式：

①基于容器：制成之后配置还是静态的 无法修改，还是需要为每种应用场景制作不同的

②基于dockerfile（环境变量修改配置）：

预先做好应用配置文件的模板+用环境变量的参数替换配置文件中变量的命令（exec），在容器启动之前自动根据环境变量修改好配置文件 然后启动容器。



Dockerfile由一行行命令语句组成，并且支持以 # 开头的注释行。一般的，Dockerfile 分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令。



指令不区分大小写，但一般用大写。

自上而下依次执行， 第一个非注释语句必须是`FROM`指令，用来指明该镜像制作是基于哪个基础镜像进行的

基于dockerfile制作镜像：

在某个特定的专业工作目录之下，放入dockerfile文件（首字母必须大写），dockerfile制作镜像时引用的文件只能位于工作目录或它的子目录之下。

.dockerignore文件：在其中写下一些文件路径（可以使用通配符），镜像打包时不会包含dockerignore中写到的路径下的文件。

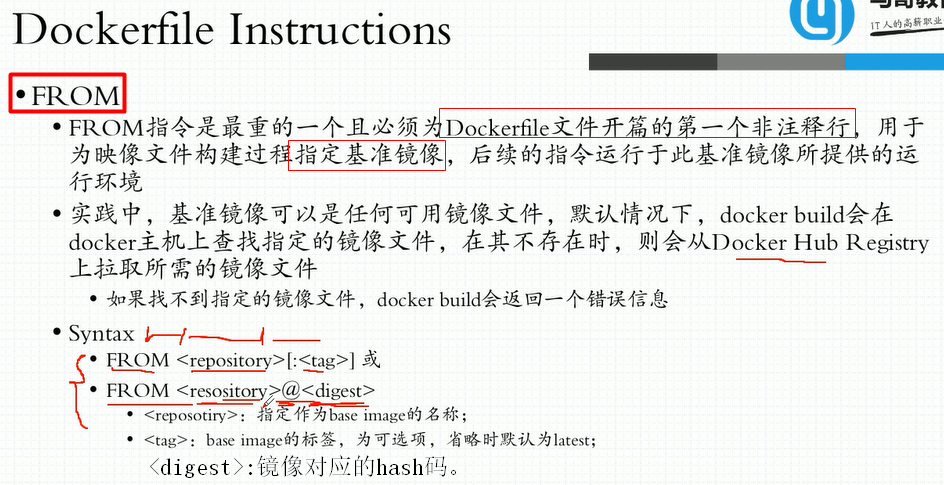
用dockerfile做镜像时可用的环境变量：$var\_name 或${var\_name}

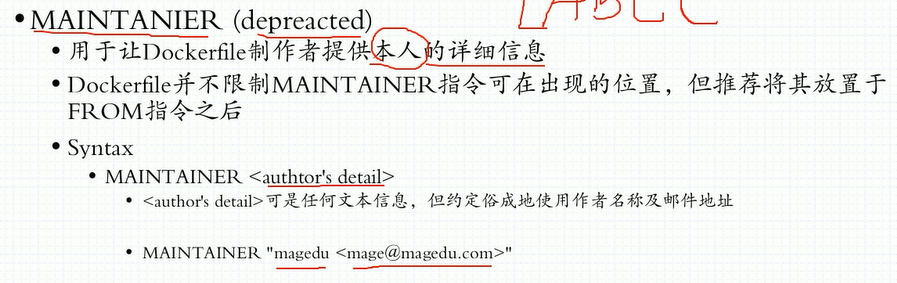
注： ${variable:-word} : 如果variable有值则整体值为variable，若variable为空或未初始化则整体为word（相当于默认值）

${variable:+word} : 如variable被设置且非空，则返回word，否则返回null，变量值不变（用于测试变量是否存在）

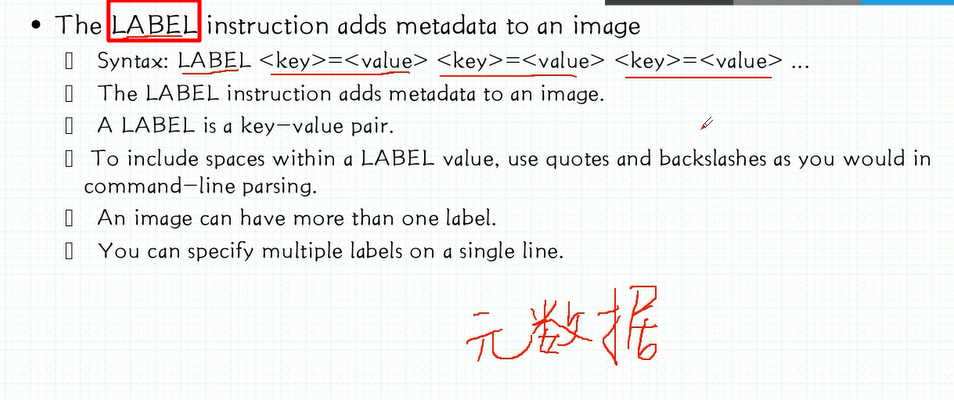
${variable:=word}：如variable被设置且非空，则返回该值，否则设置变量为word，同时返回word。

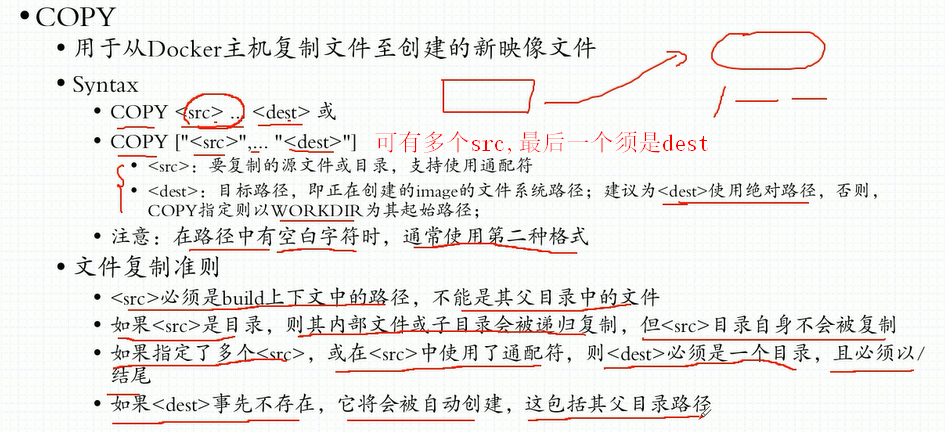
命令：docker build



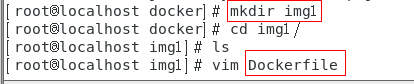


maintainer在新版本中已被弃用，现在一般用LABEL



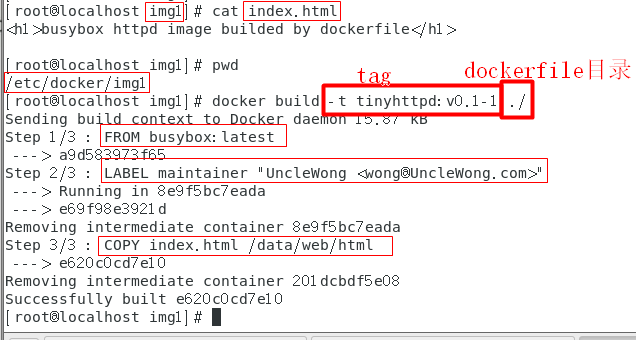


创建Dockerfile （COPY）:







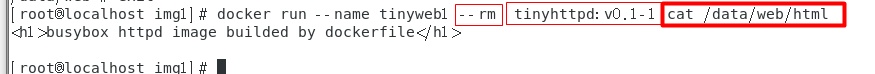


（COPY步骤中：将index.html复制到了"/data/web/"下的"html"文件中，

失误，应为/data/web/html/）

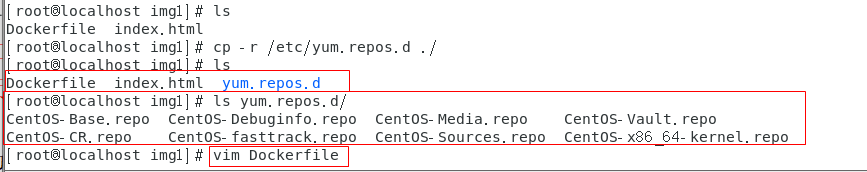
（注：Dockerfile中的每条指令，都会生成一个单独的镜像层，所以指令尽量合并 减少数量）

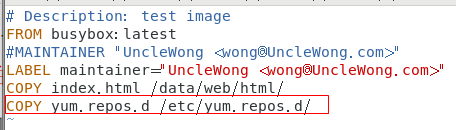




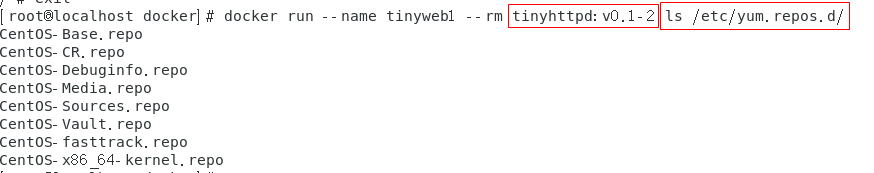
docker run的最后加上命令，表示用该命令替代镜像本来设置的默认启动的命令，在容器启动时自动执行，执行完后容器自动停止 (因为加了--rm 退出之后又自动删除)，测试时常用。

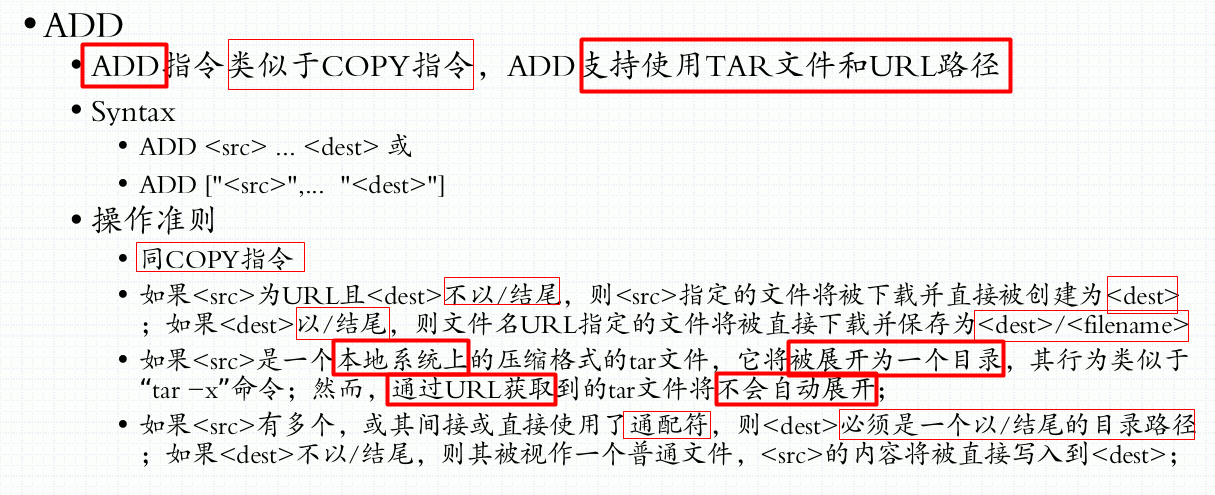
向镜像中COPY打包目录：







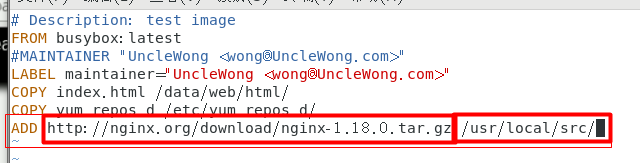




ADD：（以URL格式为来源）



vim Dockerfile







URL来的文件即使是tar格式也不会展开，还是压缩包格式。

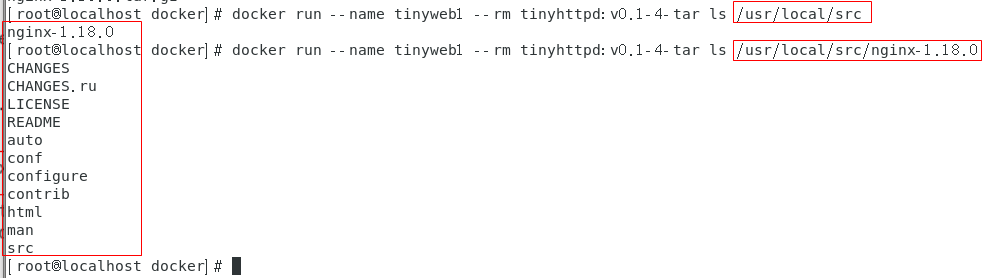
ADD：（以tar格式为来源）

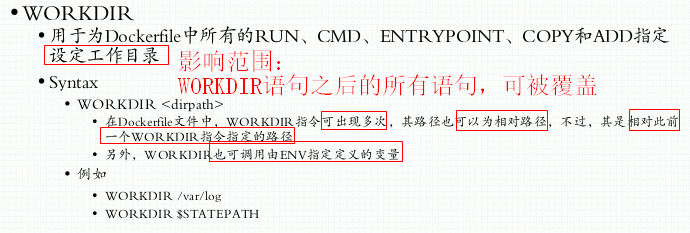
vim Dockerfile



相当于tar -x nginx-1.18.0.tar.gz -C /usr/local/src





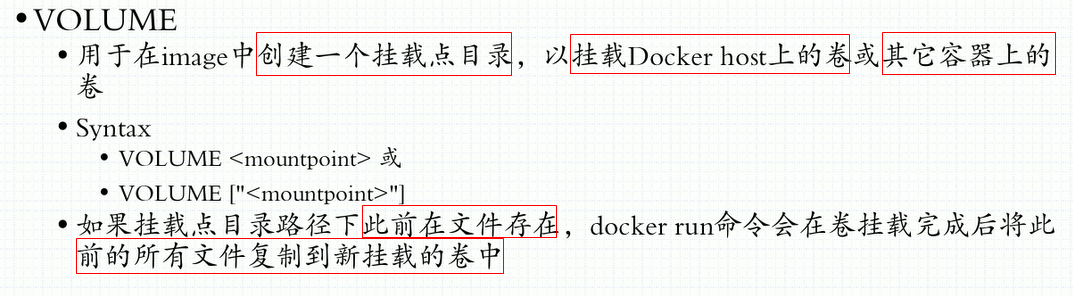


ADD nginx-1.18.0.tar.gz /usr/local/src/

=

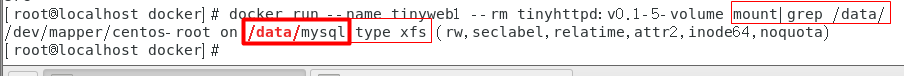
WORKDIR /usr/local/src

ADD nginx-1.18.0.tar.gz ./

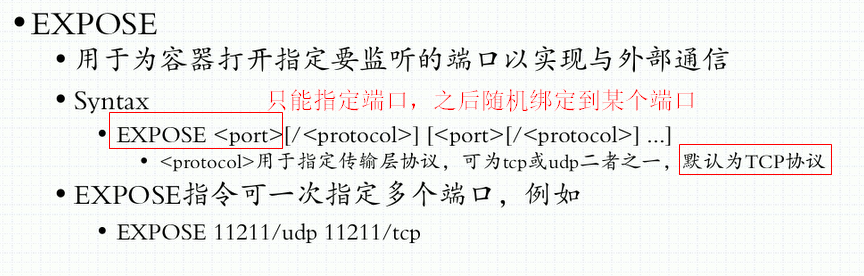










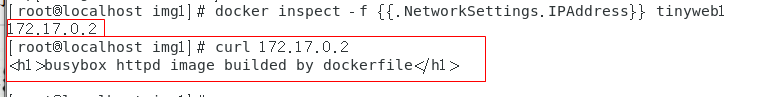








运行Httpd， -f运行在前台，-h主目录



但是单单在Dockerfile中加了EXPOSE并没有真正完全暴露端口（指外部浏览器能访问），写入镜像中的只能叫做待暴露端口，需要在容器启动时用-p -P才算真正暴露。

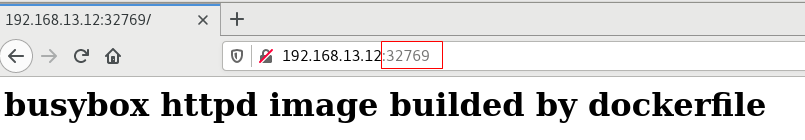


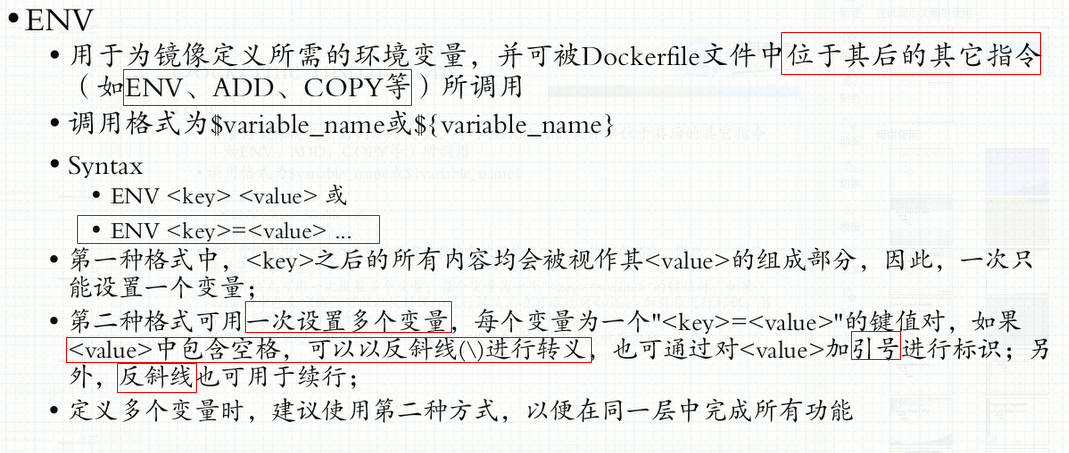


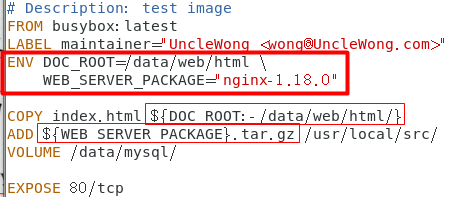
docker run时加上-P参数表示让容器暴露所有能暴露的端口



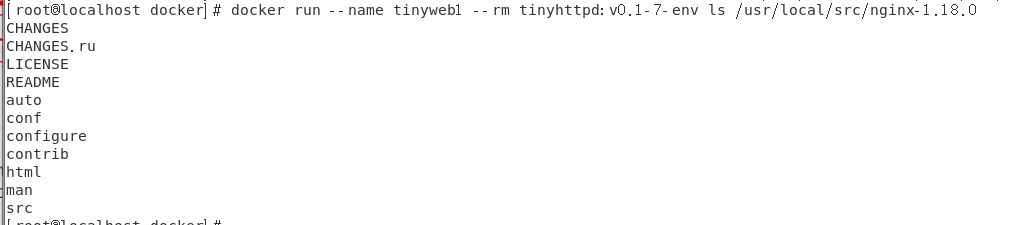
通过主机ip:暴露的端口 访问

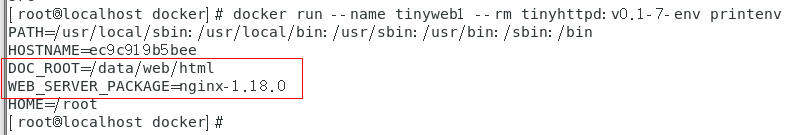




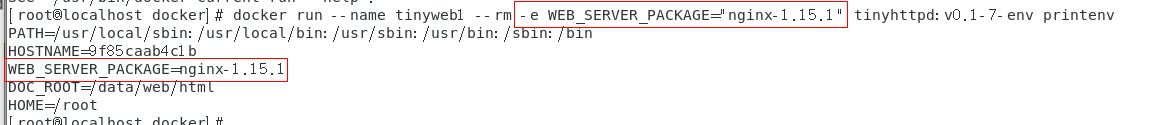




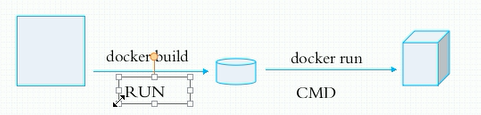


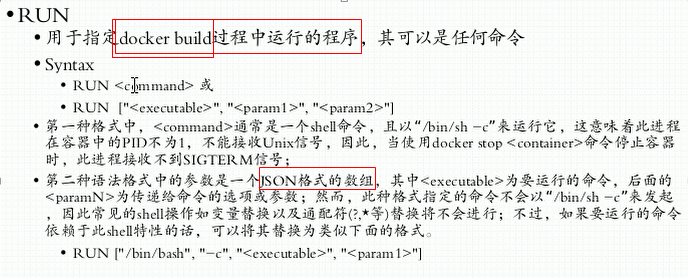


注：docker run的时候也可以通过-e修改或者添加环境变量 但改变不了镜像



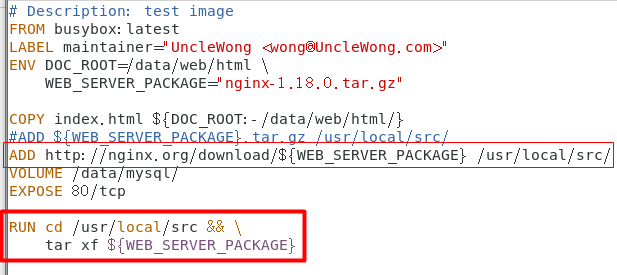
RUN & CMD

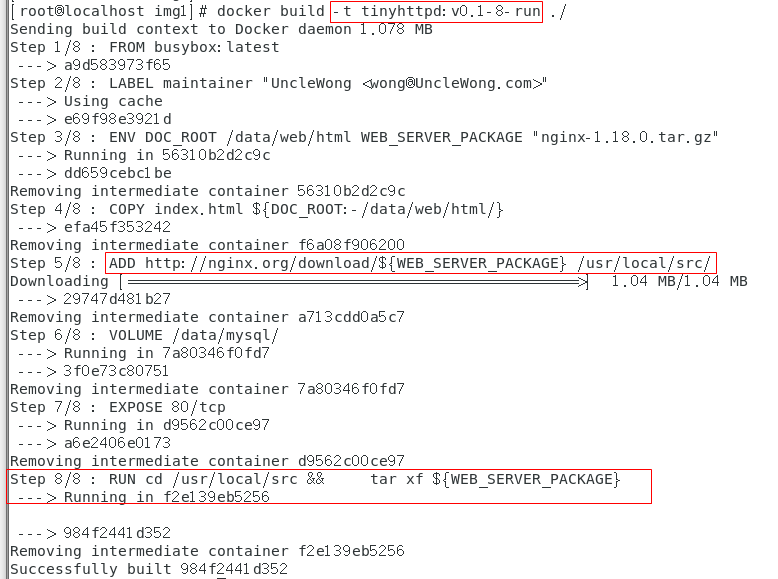


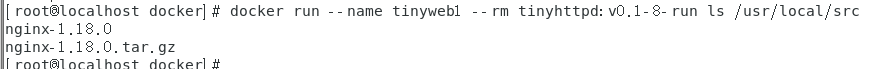


虽然一个Dockerfile可以有多个RUN，但还是建议尽量将一组RUN命令写成一条（减少镜像的层数），命令之间用 "&& \" 换行。

vim Dockerfile







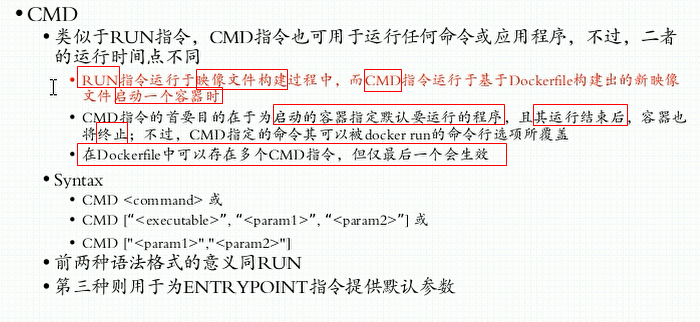
例：想基于centos制作nginx容器的话 Dockerfile:

FROM centos

RUN yum -y install epel-release && \

makecache && \

yum install nginx



CMD <command>启动的命令，自动作为shell的子进程运行，进程号pid不为1，因此没法用docker stop 来停止（只能stop进程号为1的超管进程）

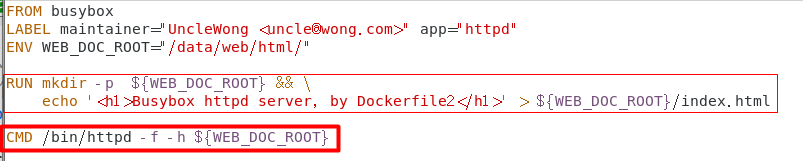
CMD ["<executable>","<param1>","<param2>"] 直接启动为pid=1的进程，不作为shell的子进程，所以可接受可处理shell信号，

CMD ["<param1>","<param2>"] 结合ENTRYPOINT指令提供默认参数

注意：json格式中要用双引号""

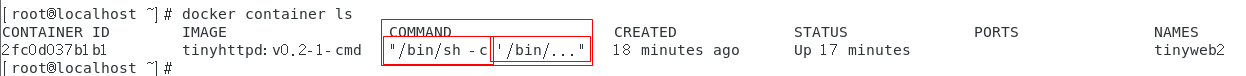
1）CMD <command>

vim Dockerfile







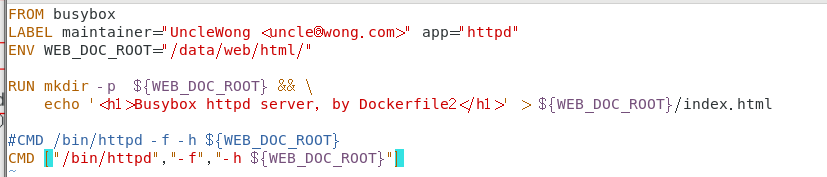


因为pid=1的超管进程不是shell的话，-it无法打开交互式命令终端。

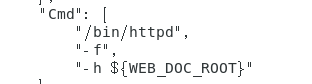
但是，可以用exec COMMAND 顶替sh的pid=1的位置



2）CMD用json数组格式传参：CMD ["<executable>","<param1>","<param2>"]



docker image inspect tinyhttpd:v0.2-2-cmd2



不再是 /bin/sh -c



报错：因为默认不以shell子进程的形式运行，而${WEB\_DOC\_ROOT}是个shell变量，不会被解析，只会被当做字符串

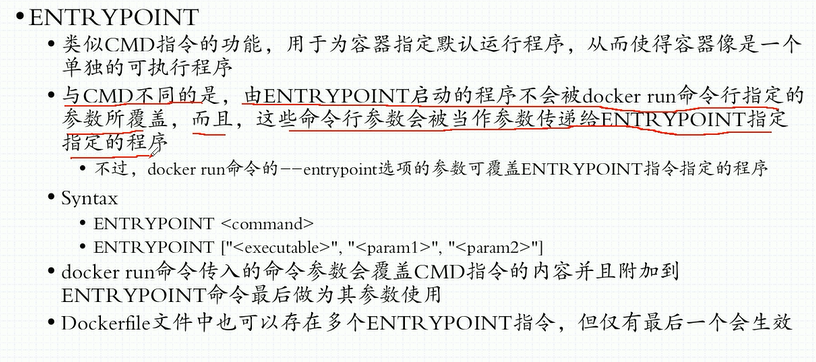
不过可以手动设置为sh的子进程：

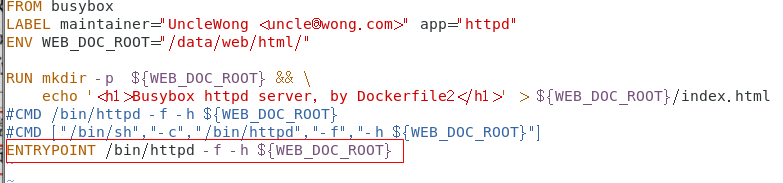


1. CMD ["<param1>","<param2>"] 结合ENTRYPOINT：

也就是说CMD ["<param1>","<param2>"]之后 在Dockerfile的结尾必须有一句ENTRYPOINT，

然后CMD把参数传给ENTRYPOINT









在docker run 最后加上命令（如果是CMD的话就会覆盖掉默认的Command）默认的命令没变，只会把命令行参数传给ENTRYPOINT后的命令



docker run最后加的ls命令没有运行 只是被当成了参数传给/bin/httpd

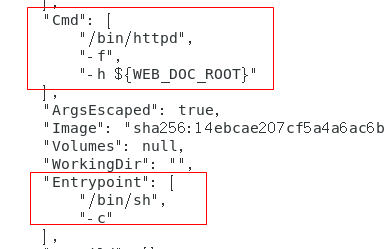


用CMD给ENTRYPOINT传参数：



结果就是ENTRYPOINT用/bin/sh -c运行了CMD中给的参数 /bin/httpd等











说明docker run 后面的参数替换了CMD定义的默认参数

ENTRYPOINT一般就用来指定/bin/sh这样就可以通过CMD和docker run 灵活地传参数使用了。

实例：期望Nginx能够通过环境变量灵活地接受参数并启动：

mkdir img3 && cd img3

vim Dockerfile





vim entrypoint.sh



容器启动时相当于运行了exec /usr/sbin/nginx -g daemon off

保证了nginx进程能作为容器中pid=1的主进程运行

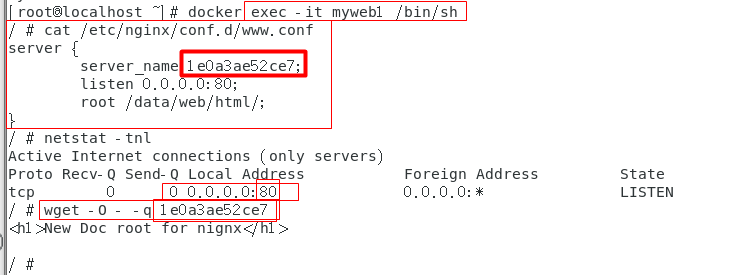
chmod 777 entrypoint.sh

vim index.html



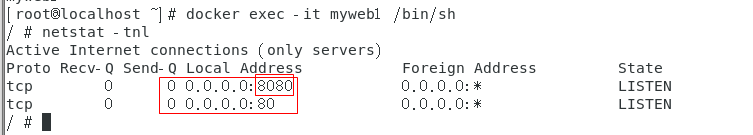


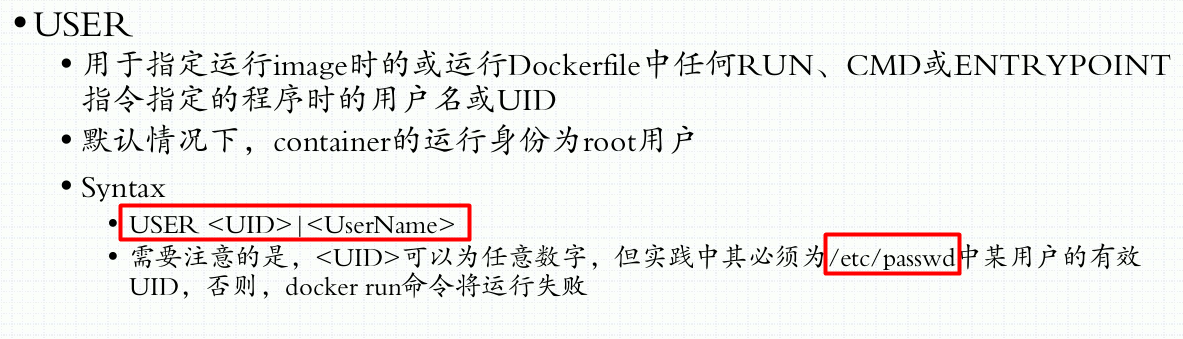


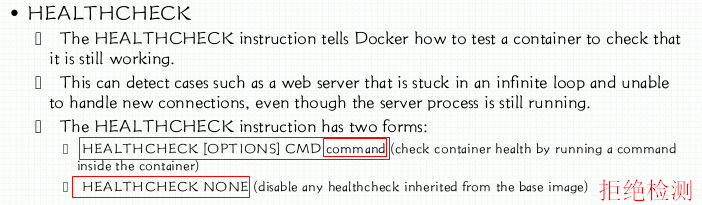


在启动容器时用-e传新的环境变量

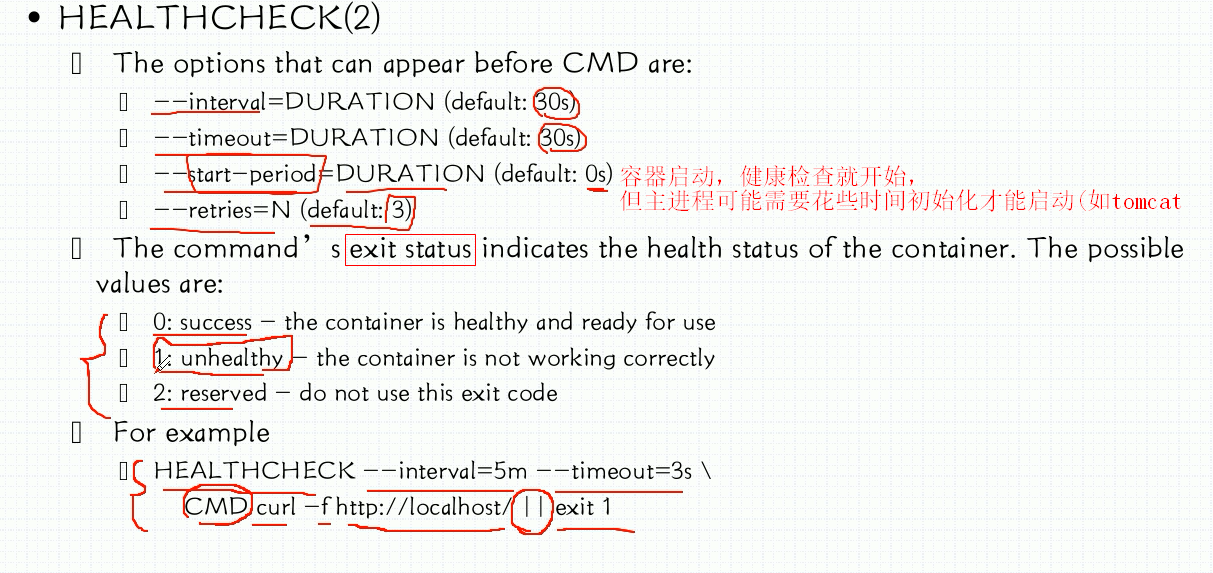








HEALTHCHECK让我们定义一个command用于检查容器的主进程工作正常与否。

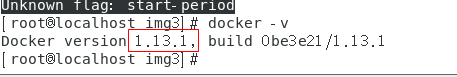


vim Dockerfile





HEALTHCHECK 报错 Unknown flag: start-period



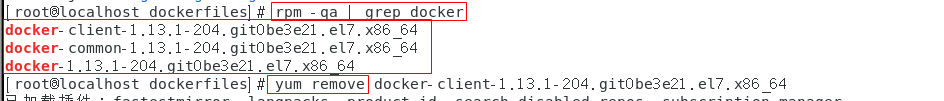
HEALTHCHECK 需要在docker-ce 至少为17.05

升级docker:

yum -y update

卸载旧版本docker：

rpm -qa | grep docker – – 列出包含docker字段的软件的信息，然后yum remove



//下载安装依赖包。

yum install -y yum-utils device-mapper-persistent-data lvm2

//设置yum源

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

//查看版本

yum list docker-ce --showduplicates | sort -f

//安装

sudo yum install -y docker-ce

//启动

systemctl start docker

systemctl enable docker

//查看版本

docker version

//添加阿里云镜像

sudo tee /etc/docker/daemon.json <<-‘EOF’

{

“registry-mirrors”: [“自己的阿里镜像地址”]

}

EOF

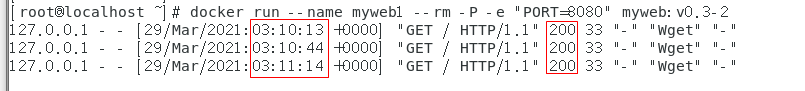
sudo systemctl daemon-reload

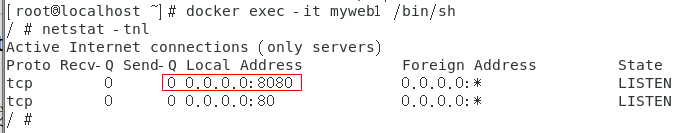
sudo systemctl restart docker

更新docker版本之后build成功无报错：



启动容器，之后可看到健康检查每30s发送一起检查请求。

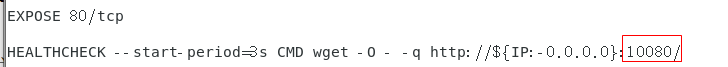




测试检测失败的场景：

vim Dockerfile

#改成检测固定端口 检测不到时会失败；





docker run 后因为检测失败 没有输出，三个周期都检测失败之后 会显示失败

SHELL： 改变默认使用的shell，

比如linux默认是["/bin/sh","-c"]， 而windows中shell是["cmd","/S"]

ARG: 参数变量，只在docker build的过程中使用，

而且能在build时用 --build-arg <变量名>=<变量> 传值，

（ENV在build时不能传值 只能用默认值， docker run时才能传）

使得一个Dockerfile能够适应多个环境，比方说可以用变脸传送版本号等信息。

注：不能在From中设变量，因为FROM必须是第一句，而变量需要先用ARG定义再用。

ONBUILD ：定义一个触发器，后面的指令“延时执行”

Dockerfile中用到了ONBUILD <指令>，后面的指令在本次build制作imageA的时候不执行，

当imageA被当做基础镜像build了imageB时 后面的<指令>才会触发

注：大部分指令都能注册为触发器指令；但是ONBUILD不能自我嵌套、不会触发FROM和MAINTAINER；

用包含ONBUILD的Dockerfile做的镜像 最好在名称中标注一下-onbuild.