**Dockerfile学习文档（busybox）**

**4、ENV/ARG**

# cd /root

# mkdir env

# cd env

# vi Dockerfile

FROM busybox

ENV NAME1 ZhangSan

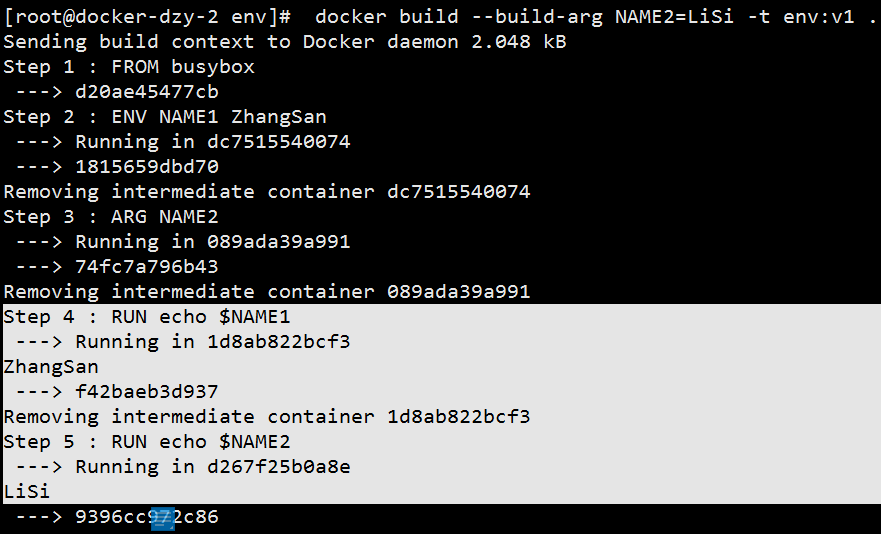
ARG NAME2

RUN echo $NAME1

RUN echo $NAME2

# docker build --build-arg NAME2=LiSi -t env:v1 .

在build执行过程中可以看到 RUN echo $NAME1 执行结果是 ZhangSan，RUN echo $NAME2执行结果是 LiSi，$NAME1直接在Dockerfile中定义的，$NAME2是在docker build命令中通过--build-arg参数传递过去的



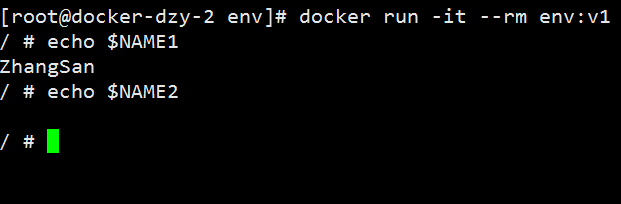
# docker run -it --rm env:v1

进入容器内

# echo $NAME1 正常输出ZhangSan

# echo $NAME2 输出为空，而不是LiSi

# exit



结论：

ARG：在Dockerfile中使用，仅仅在build docker image的过程中（包括CMD和ENTRYPOINT）有效，在image被创建和container启动之后，无效。

ENV：在Dockerfile中使用，在build docker imag的过程中有效，在image被创建和container启动后作为环境变量依旧也有效。

**5、VOLUME**

# cd /root

# mkdir volume

# cd volume

# vi Dockerfile

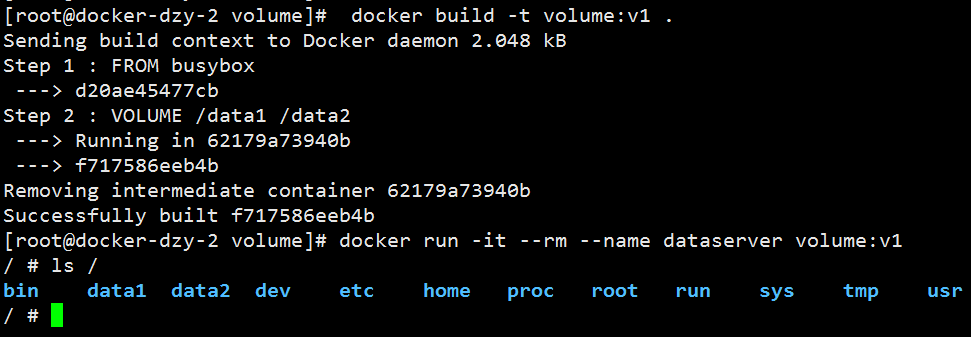
FROM busybox

VOLUME ["/data1","/data2"]

# docker build -t volume:v1 .

# docker run -it --rm --name dataserver volume:v1

容器内可以看到 /data1，/data2两个目录已经创建好：



不要退出该容器，再打开两个Shell

Shell 1终端

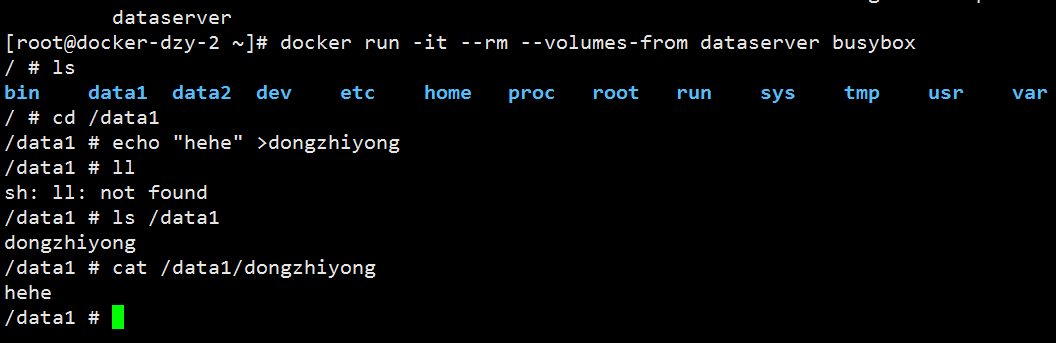
# docker run -it --rm --volumes-from dataserver busybox //--volumes-from:本容器共享dataserver容器的目录

Shell 2终端

# docker run -it --rm --volumes-from dataserver busybox

可以看到新建的两个容器中，都存在/data1，/data2目录

此时在3个容器任意一个容器的/data1，/data2目录中 增删改文件，其他容器都可以看到，实现了数据卷的共享：



退出Shell1和Shell2终端的容器，dataserver容器数据仍在，退出dataserver容器，由于创建容器的时候带了—rm参数，容器自动删除，数据也丢失。

**6、EXPOSE**

# cd /root

# mkdir expose

# cd expose

# vi Dockerfile

FROM busybox

RUN mkdir /html

RUN echo "hello docker" > /html/index.html

CMD ["httpd","-f","-h","/html/"]

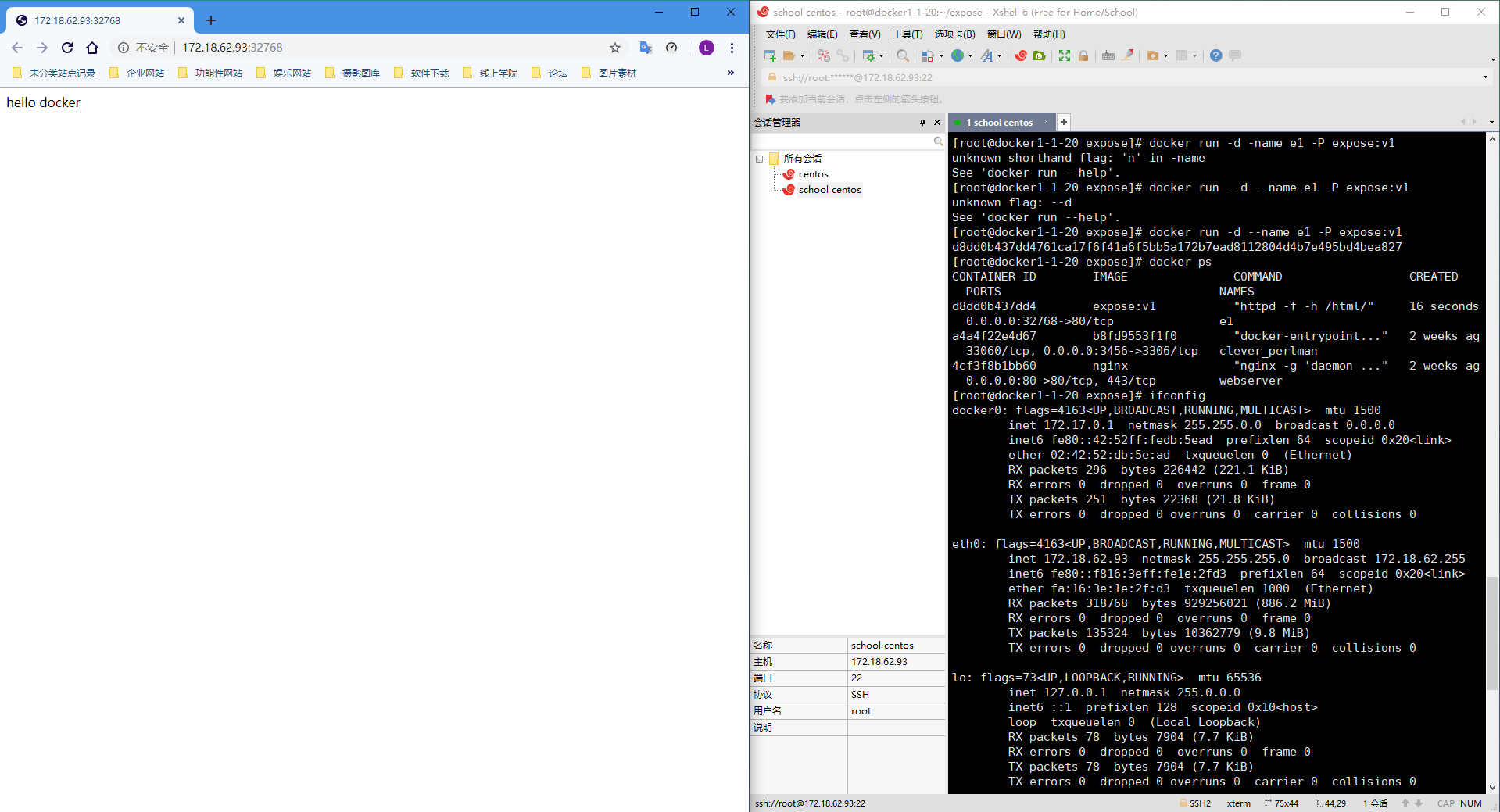
EXPOSE 80

# docker build -t expose:v1 .

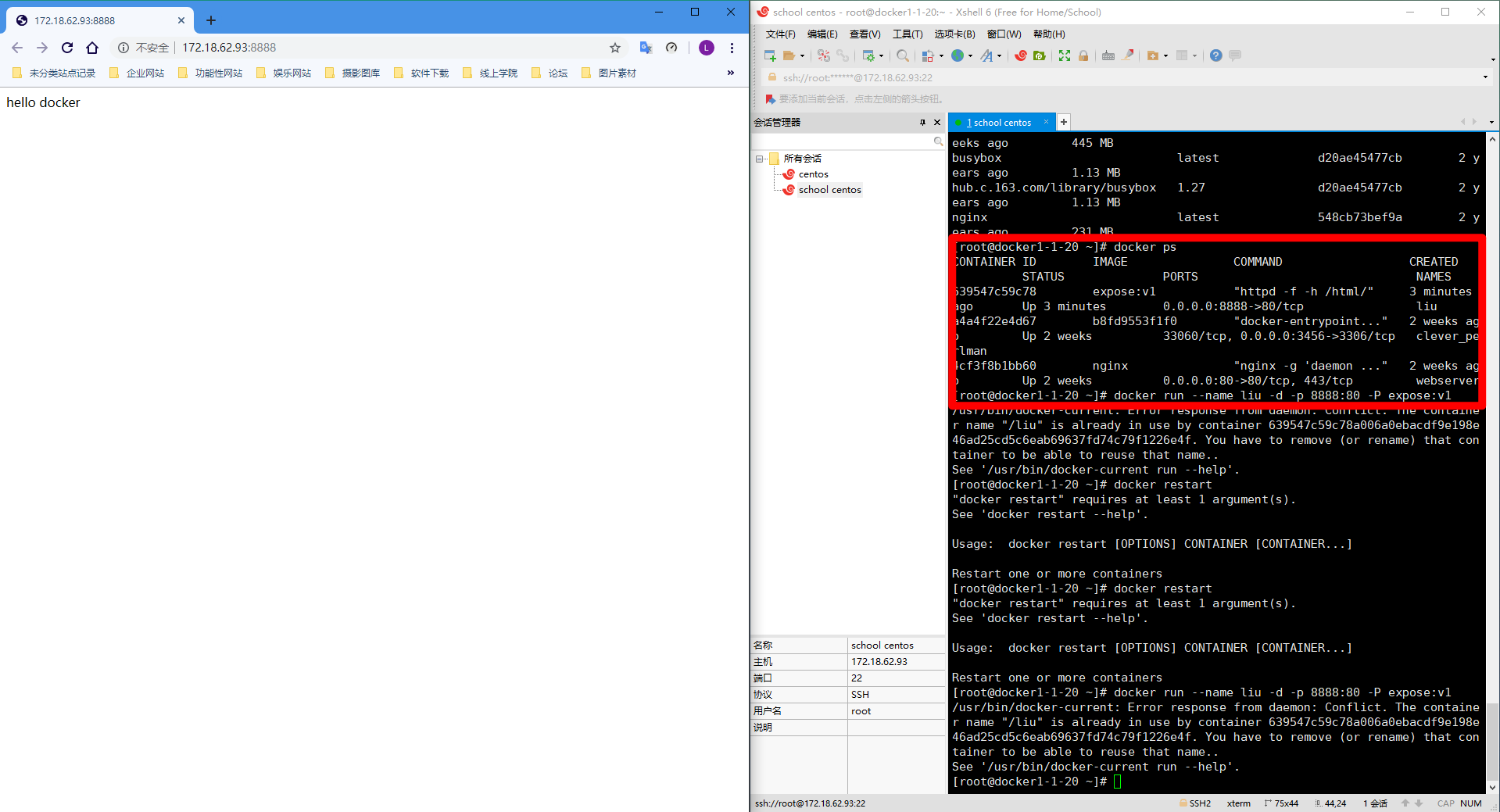
# docker run -d --name e1 -P expose:v1

# docker ps 可以看到容器e1随机生成一个端口绑定到容器内的80端口

docker run在-P随机生成端口的时候，容器内的端口会选择expose暴露的端口



Docker run –name liu -d -p 8888:80 -P expose:v1 原为随机端口，先在启用固定端口的命令。



端口映射 容器互联

**7、WORKDIR**

# cd /root

# mkdir workdir

# cd workdir

# vi Dockerfile

FROM busybox

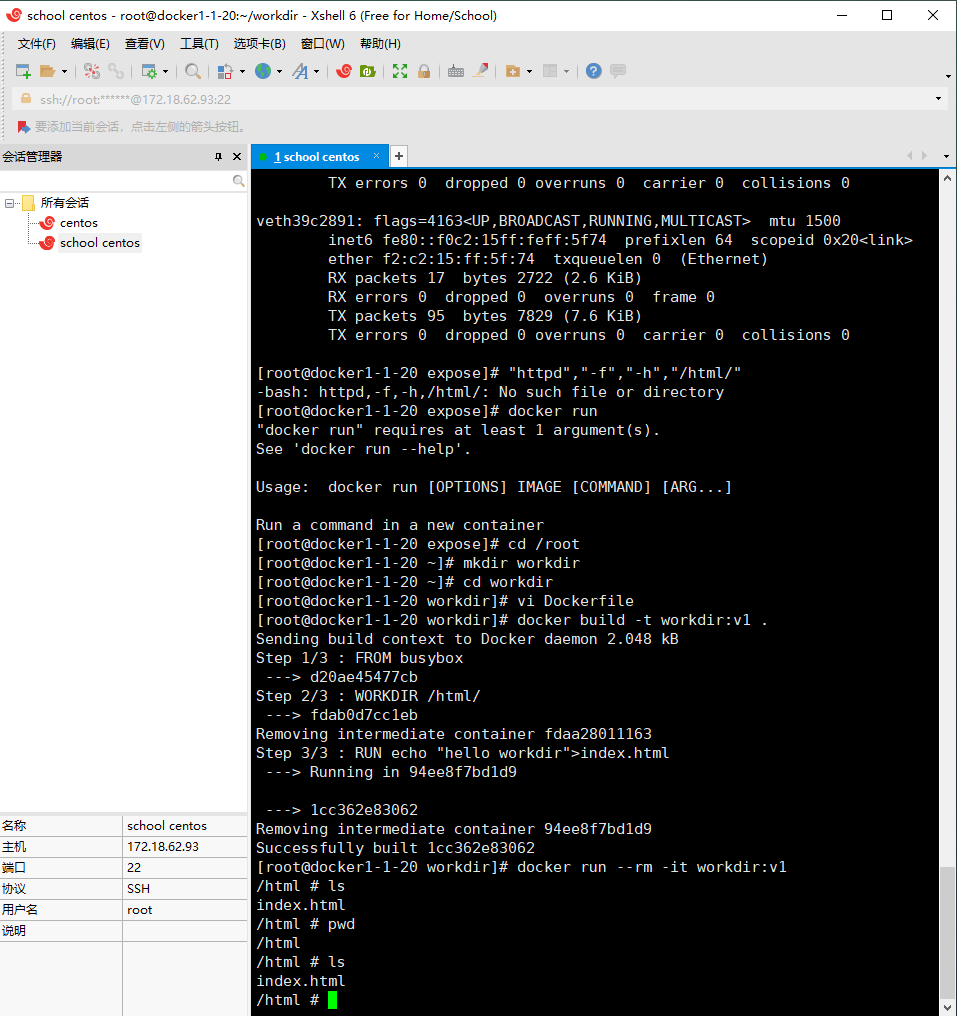
WORKDIR /html/

RUN echo "hello workdir" > index.html

# docker build -t workdir:v1 .

# docker run --rm -it workdir:v1

进入该容器内，执行ls可以看到 /html目录以及创建，执行pwd可以看到当前用户所在目录就是该目录，执行ls可以看到index.html写到该目录下了



用来指定当前工作目录（或者称为当前目录）

当使用相对目录的情况下，采用上一个WORKDIR指定的目录作为基准

相当与cd 命令，但不同的是指定了WORKDIR后，容器启动时执行的命令会在该目录下执行

**8、USER**

# cd /root

# mkdir user

# cd user

# vi Dockerfile

FROM busybox

RUN adduser -D user1

USER user1

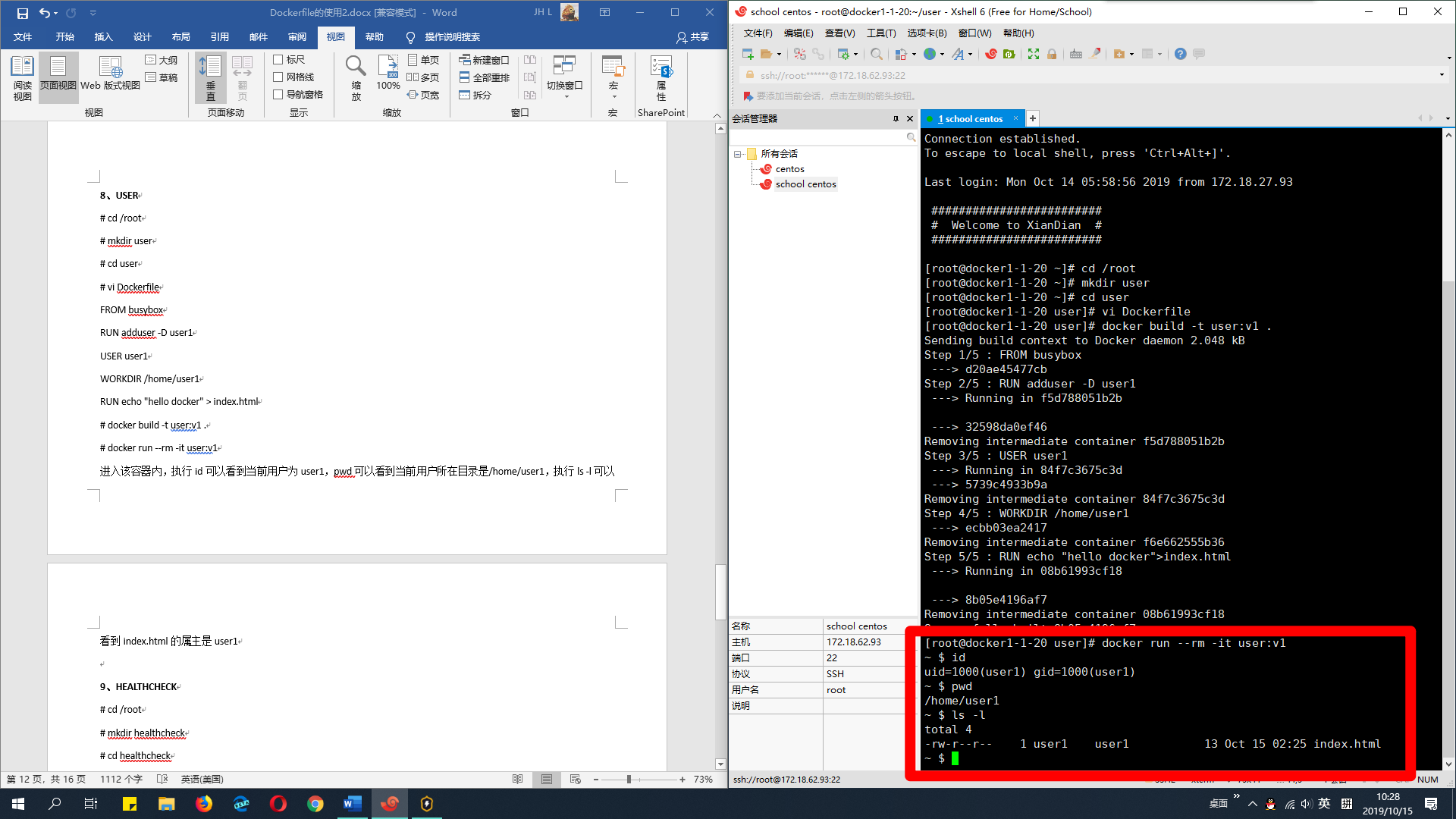
WORKDIR /home/user1

RUN echo "hello docker" > index.html

# docker build -t user:v1 .

# docker run --rm -it user:v1

进入该容器内，执行id可以看到当前用户为user1，pwd可以看到当前用户所在目录是/home/user1，执行ls -l可以看到index.html的属主是user1



指定运行时的用户名或UID，后续的RUN也会使用指定的用户。

当服务不需要管理权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户

**9、HEALTHCHECK**

# cd /root

# mkdir healthcheck

# cd healthcheck

先创建一个shell脚本，该脚本内容为 检测文件夹/test是否存在

# vi test.sh

#!/bin/sh

if [ -d "/test/" ];

then exit 0

else exit 1

fi

# vi Dockerfile

FROM busybox

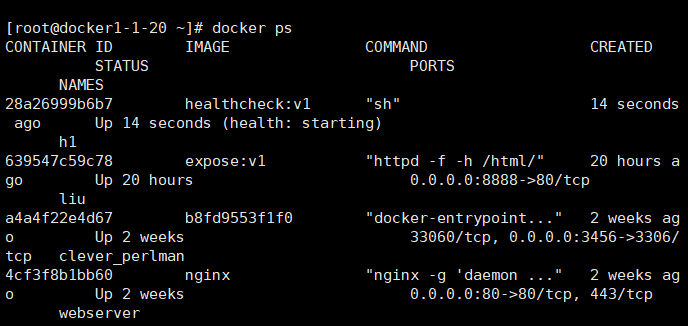
COPY test.sh /opt/

HEALTHCHECK --interval=15s --timeout=5s --retries=2 CMD /bin/sh /opt/test.sh

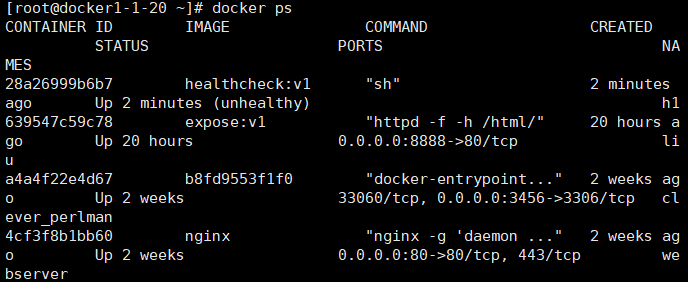
# docker build -t healthcheck:v1 .

# docker run -it --rm --name h1 healthcheck:v1

另开一个Shell终端，执行



# docker ps 可以看到容器h1的STATUS是 health:starting，过30s后，再次执行



# docker ps 可以看到容器h1的STATUS是 health:unhealthy，因为容器内没有检测到/test文件夹，退出h1容器，再次创建一个容器

# docker run -it --rm --name h2 healthcheck:v1

另开一个Shell终端，在30s内执行完

# docker exec -it h2 /bin/sh 进入该容器，再执行

# mkdir /test

此时另开一个Shell终端，30s后，可以执行 docker ps 看到容器h2的STATUS是 health:healthy

**10、ONBUILD**

# cd /root

# mkdir onbuild

# cd onbuild

# vi Dockerfile

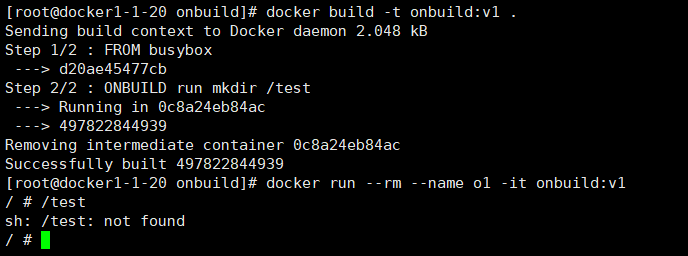
FROM busybox

ONBUILD RUN mkdir /test

# docker build -t onbuild:v1 .

# docker run --rm --name o1 -it onbuild:v1

进入容器可以查看到 /test文件夹没有创建，



再次修改Dockerfile

# vi Dockerfile

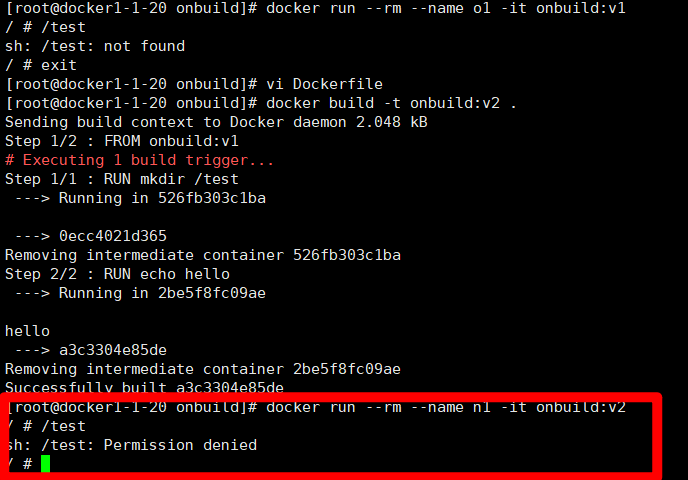
FROM onbuild:v1

RUN echo hello

# docker build -t onbuild:v2 .

# docker run --rm --name n1 -it onbuild:v2

进入容器内可以看到 /tesexit文件夹已经创建



版本号

