**FROM**

功能为指定基础镜像，并且必须是第一条指令。

如果不以任何镜像为基础，那么写法为：FROM scratch。

同时意味着接下来所写的指令将作为镜像的第一层开始

语法：

**FROM <image>**  
**FROM <image>:<tag>**  
**FROM <image>:<digest>**

三种写法，其中<tag>和<digest> 是可选项，如果没有选择，那么默认值为latest

RUN

 功能为运行指定的命令

RUN命令有两种格式

**1. RUN <command>**  
**2. RUN ["executable", "param1", "param2"]**

第一种后边直接跟shell命令

* 在linux操作系统上默认 /bin/sh -c
* 在windows操作系统上默认 cmd /S /C

第二种是类似于函数调用。

可将executable理解成为可执行文件，后面就是两个参数。

两种写法比对：

* **RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME**
* **RUN ["/bin/bash", "-c", "echo hello"]**

注意：多行命令不要写多个RUN，原因是Dockerfile中每一个指令都会建立一层.

 多少个RUN就构建了多少层镜像，会造成镜像的臃肿、多层，不仅仅增加了构件部署的时间，还容易出错。

RUN书写时的换行符是\

CMD

功能为容器启动时要运行的命令

语法有三种写法

**1. CMD ["executable","param1","param2"]**  
**2. CMD ["param1","param2"]**  
**3. CMD command param1 param2**

第三种比较好理解了，就时shell这种执行方式和写法

第一种和第二种其实都是可执行文件加上参数的形式

举例说明两种写法：

* **CMD [ "sh", "-c", "echo $HOME"**
* **CMD [ "echo", "$HOME" ]**

补充细节：这里边包括参数的一定要用双引号，就是",不能是单引号。千万不能写成单引号。

原因是参数传递后，docker解析的是一个JSON array

RUN & CMD

不要把RUN和CMD搞混了。

RUN是构件容器时就运行的命令以及提交运行结果

CMD是容器启动时执行的命令，在构件时并不运行，构件时紧紧指定了这个命令到底是个什么样子

LABEL

功能是为镜像指定标签

语法：

**LABEL <key>=<value> <key>=<value> <key>=<value> ...**

 一个Dockerfile种可以有多个LABEL，如下：

***LABEL "com.example.vendor"="ACME Incorporated"***  
***LABEL com.example.label-with-value="foo"***  
***LABEL version="1.0"***  
***LABEL description="This text illustrates \***  
***that label-values can span multiple lines."***

 但是并不建议这样写，最好就写成一行，如太长需要换行的话则使用\符号

如下：

***LABEL multi.label1="value1" \***  
***multi.label2="value2" \***  
***other="value3"***

说明：LABEL会继承基础镜像种的LABEL，如遇到key相同，则值覆盖

MAINTAINER

指定作者

语法：

**MAINTAINER <name>**

EXPOSE

功能为暴漏容器运行时的监听端口给外部

但是EXPOSE并不会使容器访问主机的端口

如果想使得容器与主机的端口有映射关系，必须在容器启动的时候加上 -P参数

ENV

功能为设置环境变量

语法有两种

**1. ENV <key> <value>**  
**2. ENV <key>=<value> ...**

两者的区别就是第一种是一次设置一个，第二种是一次设置多个

**ADD**

 一个复制命令，把文件复制到景象中。

如果把虚拟机与容器想象成两台linux服务器的话，那么这个命令就类似于scp，只是scp需要加用户名和密码的权限验证，而ADD不用。

语法如下：

**1. ADD <src>... <dest>**  
**2. ADD ["<src>",... "<dest>"]**

<dest>路径的填写可以是容器内的绝对路径，也可以是相对于工作目录的相对路径

<src>可以是一个本地文件或者是一个本地压缩文件，还可以是一个url

如果把<src>写成一个url，那么ADD就类似于wget命令

如以下写法都是可以的：

* ***ADD test relativeDir/***
* ***ADD test /relativeDir***
* ***ADD http://example.com/foobar /***

尽量不要把<scr>写成一个文件夹，如果<src>是一个文件夹了，复制整个目录的内容,包括文件系统元数据

COPY

看这个名字就知道，又是一个复制命令

语法如下：

**1. COPY <src>... <dest>**  
**2. COPY ["<src>",... "<dest>"]**

与ADD的区别

COPY的<src>只能是本地文件，其他用法一致

ENTRYPOINT

功能是启动时的默认命令

语法如下：

**1. ENTRYPOINT ["executable", "param1", "param2"]**  
**2. ENTRYPOINT command param1 param2**

如果从上到下看到这里的话，那么你应该对这两种语法很熟悉啦。

第二种就是写shell

第一种就是可执行文件加参数

与CMD比较说明（这俩命令太像了，而且还可以配合使用）：

1. 相同点：

* 只能写一条，如果写了多条，那么只有最后一条生效
* 容器启动时才运行，运行时机相同

2. 不同点：

* ENTRYPOINT不会被运行的command覆盖，而CMD则会被覆盖
* 如果我们在Dockerfile种同时写了ENTRYPOINT和CMD，并且CMD指令不是一个完整的可执行命令，那么CMD指定的内容将会作为ENTRYPOINT的参数

如下：

***FROM ubuntu***  
***ENTRYPOINT ["top", "-b"]***  
***CMD ["-c"]***

* 如果我们在Dockerfile种同时写了ENTRYPOINT和CMD，并且CMD是一个完整的指令，那么它们两个会互相覆盖，谁在最后谁生效

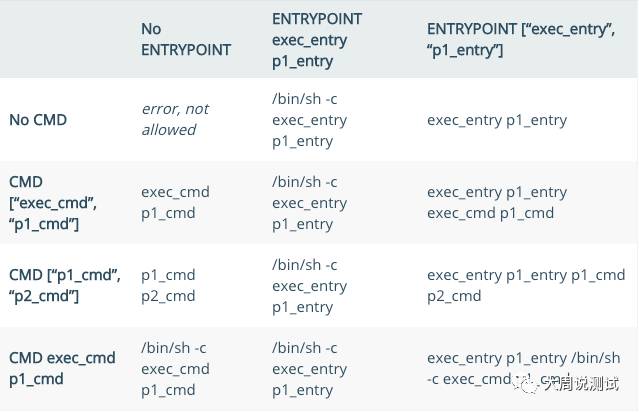
如下：

***FROM ubuntu***  
***ENTRYPOINT ["top", "-b"]***  
***CMD ls -al***

那么将执行ls -al ,top -b不会执行。

Docker官方使用一张表格来展示了ENTRYPOINT 和CMD不同组合的执行情况

（下方表格来自docker官网）



VOLUME

可实现挂载功能，可以将内地文件夹或者其他容器种得文件夹挂在到这个容器种

语法为：

**VOLUME ["/data"]**

说明：

   ["/data"]可以是一个JsonArray ，也可以是多个值。所以如下几种写法都是正确的

***VOLUME ["/var/log/"]***

***VOLUME /var/log***

***VOLUME /var/log /var/db***

一般的使用场景为需要持久化存储数据时

容器使用的是AUFS，这种文件系统不能持久化数据，当容器关闭后，所有的更改都会丢失。

所以当数据需要持久化时用这个命令。

USER

设置启动容器的用户，可以是用户名或UID，所以，只有下面的两种写法是正确的

* **USER daemo**
* **USER UID**

注意：如果设置了容器以daemon用户去运行，那么RUN, CMD 和 ENTRYPOINT 都会以这个用户去运行

WORKDIR

语法：

**WORKDIR /path/to/workdir**

设置工作目录，对RUN,CMD,ENTRYPOINT,COPY,ADD生效。如果不存在则会创建，也可以设置多次。

如：

***WORKDIR /a***  
***WORKDIR b***  
***WORKDIR c***  
***RUN pwd***

pwd执行的结果是/a/b/c

WORKDIR也可以解析环境变量

如：

***ENV DIRPATH /path***  
***WORKDIR $DIRPATH/$DIRNAME***  
***RUN pwd***

pwd的执行结果是/path/$DIRNAME

ARG

语法：

**ARG <name>[=<default value>]**

设置变量命令，ARG命令定义了一个变量，在docker build创建镜像的时候，使用 --build-arg <varname>=<value>来指定参数

如果用户在build镜像时指定了一个参数没有定义在Dockerfile种，那么将有一个Warning

提示如下：

**[Warning] One or more build-args [foo] were not consumed.**

我们可以定义一个或多个参数，如下：

**FROM busybox**  
**ARG user1**  
**ARG buildno**  
**...**

也可以给参数一个默认值：

**FROM busybox**  
**ARG user1=someuser**  
**ARG buildno=1**  
**...**

如果我们给了ARG定义的参数默认值，那么当build镜像时没有指定参数值，将会使用这个默认值

ONBUILD

语法：

**ONBUILD [INSTRUCTION]**

这个命令只对当前镜像的子镜像生效。

比如当前镜像为A，在Dockerfile种添加：

**ONBUILD RUN ls -al**

这个 ls -al 命令不会在A镜像构建或启动的时候执行

此时有一个镜像B是基于A镜像构建的，那么这个ls -al 命令会在B镜像构建的时候被执行。

STOPSIGNAL

语法：

**STOPSIGNAL signal**

STOPSIGNAL命令是的作用是当容器推出时给系统发送什么样的指令

HEALTHCHECK

 容器健康状况检查命令

语法有两种：

**1. HEALTHCHECK [OPTIONS] CMD command**  
**2. HEALTHCHECK NONE**

第一个的功能是在容器内部运行一个命令来检查容器的健康状况

第二个的功能是在基础镜像中取消健康检查命令

[OPTIONS]的选项支持以下三中选项：

***--interval=DURATION 两次检查默认的时间间隔为30秒***

***--timeout=DURATION 健康检查命令运行超时时长，默认30秒***

***--retries=N 当连续失败指定次数后，则容器被认为是不健康的，状态为unhealthy，默认次数是3***

注意：

HEALTHCHECK命令只能出现一次，如果出现了多次，只有最后一个生效。

CMD后边的命令的返回值决定了本次健康检查是否成功，具体的返回值如下：

***0: success - 表示容器是健康的***

***1: unhealthy - 表示容器已经不能工作了***

***2: reserved - 保留值***

例子：

**HEALTHCHECK --interval=5m --timeout=3s \**  
**CMD curl -f http://localhost/ || exit 1**

健康检查命令是：curl -f http://localhost/ || exit 1

两次检查的间隔时间是5秒

命令超时时间为3秒