<https://docs.docker.com/develop/develop-images/dockerfile_best-practices/>

只有RUN、COPY、ADD create layers。其他的结构创建临时中间images，不会增加build的大小。

使用 multi-stage build 只copy你需要的文件到最终的image中。可以允许你在中间build stage包含tools和debug信息，而不会增加image的大小。

multi-stage build可以减少最终image的大小，而不用极力去减少中间层和文件。

按字母排序mutil-line 参数，这样可以避免重复参数，也容易更新，也容易review。在backslash(\)前面加一个空格是很好的。

例如：

RUN apt-get update && apt-get install -y \

bzr \

cvs \

git \

mercurial \

subversion

利用构建缓存

Docker构建镜像的时候，docker按照在Dockerfile的顺序逐步执行instructions 。当每个instruction检查后，docker 在缓存中寻找一个已经存在可以reuse的镜像，而不是创建一个新的。

如果不想使用cache，可以使用 --no-cache=true 选项在docker build的命令上，如果想让docker使用cache，明白什么时候可以，什么时候不可以找到匹配的镜像，是很重要的。Docker遵循的基本规则归纳如下：

开始就将父镜像放入cache中，下一个instruction 和源于基本镜像的所有子镜像比较，看看有没有一个就是用 这个 exact same instruction 构建的。如果没有，这个cache就无效了。

大多数情况下，简单的用Dockerfile中的instruction和一个子image比较是sufficient。然而，某些instructions 需要跟多的检查和解释。

对于ADD和COPY instruction，镜像中的所有文件内容被检查，每个文件会计算出一个checksum， 文件的last-modified 和 last-accessed 时间不会被计入 checksum， 在cache搜寻中，用这个镜像的checksum和已经存在镜像的checksum比较。如果当前文件发生了一些变化，比如内容，元数据，cache也会失效

撇开ADD和COPY 命令， cache检查不会看container中的文件来决定cache匹配。例如，执行 RUN apt-get –y update 命令，容器中的文件更新了，不会检查来决定一个存在cache是否命中。在这种情况下，仅仅命令本身字符串被用来查找一个匹配

一旦cache失效，Dockerfile文件中后续的命令产生新的镜像 且cache不会被用到。

Dockerfile instruction——Dockerfile中的指令

FROM

任何时候可能，使用当前官方镜像作为自己镜像的基础，推荐Alpine image 因为它是严格控制的而且size很小（一般小于5m），而且任然是一个完整的linux 发行版。

LABEL

给image添加label可以帮助你通过project组织管理image，记录licensing信息，对自动化有帮助，或者其他原因。每个label都用LABEL开始，然后跟多个键值对，下面的例子给出了不同的正确格式。解释说明在一行内

如果字符串中有空格必须quote引起来，要么就不要有空格。内部的引号也不行。

LABEL com.example.version="0.0.1-beta"

LABEL vendor1="ACME Incorporated"

LABEL vendor2=ZENITH\ Incorporated

LABEL com.example.release-date="2015-02-12"

LABEL com.example.version.is-production=""

一个镜像可以有多个label，docker1.10版本，推荐奖多个label写在一行，用一个LABEL指令，来防止额外的layer被创建。现在不是必须的了，但是这种写在一行的语法还是支持的。

写在一行里面

# Set multiple labels on one line

LABEL com.example.version="0.0.1-beta" com.example.release-date="2015-02-12"

# Set multiple labels at once, using line-continuation characters to break long lines

LABEL vendor=ACME\ Incorporated \

com.example.is-beta= \

com.example.is-production="" \

com.example.version="0.0.1-beta" \

com.example.release-date="2015-02-12"

RUN

用反斜杠将长的或者复杂的RUN指令拆分成多行，来时Dockerfile更可读，更易理解，和更好维护。

RUN最常用的场景是apt-get的应用。因为安装包，RUN apt-get命令有几个gotcha需要注意

避免使用RUN apt-get 和dist-upgrade，因为父image中许多essential的包不能更新，在一个无特权的容器中。如果遇到了父镜像中的包过时了，联系他的维护者。如果某个特定的包需要更新，比如foo， 可以用 apt-get install –y foo 来自动更新。

通常联合RUN apt-get update 和apt-get install 在在同一个statement（声明）中，如下：

RUN apt-get update && apt-get install -y \

package-bar \

package-baz \

package-foo

在RUN声明中单独使用apt-get update 会引起缓存问题，并且后面的apt-get install 指令会失败。例如，假设（say）你有一个Dockerfile：

FROM ubuntu:18.04

RUN apt-get update

RUN apt-get install -y curl

Build 这个image之后，所有的层都在docker 缓存中。假设你后来通过增加一个额外的package修改了apt-get install

FROM ubuntu:18.04

RUN apt-get update

RUN apt-get install -y curl nginx

Docker 看见最初的和修改后的指令是identical的然后就用了前面step的cache，结果apt-get update 就没有执行，因为这个build用了缓存的版本，apt-get install就是用过时的curl和nginx包。

使用RUN apt-get update && apt-get install –y 保证Dockerfile安装最新的版本，而不需要进一步编码和人为的干预。这就是著名的cache busting技术。你也可以指定版本来实现cache-busting技术，就是version pinning

RUN apt-get update && apt-get install -y \

package-bar \

package-baz \

package-foo=1.3.\*

version pinning 强制构建检索指定的版本，忽略缓存中的内容。这个技术可以减少失败因为不可预期的（unanticipateed）变化在必须的包里。

下面是一个很好的RUN指令，说明了apt-get所有的推荐使用。

RUN apt-get update && apt-get install -y \

aufs-tools \

automake \

build-essential \

curl \

dpkg-sig \

libcap-dev \

libsqlite3-dev \

mercurial \

reprepro \

ruby1.9.1 \

ruby1.9.1-dev \

s3cmd=1.1.\* \

&& rm -rf /var/lib/apt/lists/\*

s3cmd参数制定了版本1.1.\*。如果这个image先前用了比较老的版本，指定新的版本会使apt-get update 的缓存破裂（就是失效），从而保证安装最新的版本，将包分行写也可以防止包重复的错误。

而且，当你通过删除/var/lib/apt/lists来删除apt缓存，可以减小image的size，因为apt缓存不是存储在layer中的。因为RUN声明以apt-get update 开始，包刷新总是在apt-get install前面。

官方的debian和Ubuntu镜像自动的执行 apt-get clean， 所以显示的调用就没必要了。

使用pipes

一些RUN命令依赖管道将一个命令的结果传入另一个命令的能力。Pipe is |

RUN wget -O - https://some.site | wc -l > /number

Docker 执行这些命令使用 /bin/sh –c 解释器，它只通过评估管道中的最后一条命令的退出码来决定是否成功。

如果你想管道中不论哪一步失败都使命令失败，可以提前 set –o pipefail && 来保证错误终止构建

RUN set -o pipefail && wget -O - https://some.site | wc -l > /number

不是所有的shell 都支持 –o pipefail 选项。比如基于debian镜像的dash 脚本就不支持。可以使用RUN的exec 形式来明确的指定一个支持-o pipefail选项的shell。例如

RUN ["/bin/bash", "-c", "set -o pipefail && wget -O - https://some.site | wc -l > /number"]

ARG指令定义了用户可以在编译时传递的变量，如使用如下命令：

--build-arg <varname>=<value>

ENV指令是在dockerfile里面设置环境变量，运行时传递。

结合使用

1. ARG var
2. ENV var=${var}

You can then either build an image with a specific var value at build-time

docker build --build-arg var=xxx

 or run a container with a specific runtime value

docker run -e var=yyy