Dockerfile最佳实践

使用缓存

Dockerfile的每条指令都会将结果提交为新的镜像。下一条指令基于上一条指令的镜像进行构建。如果一个镜像拥有相同的父镜像和指令（除了 `ADD`），Docker将会使用镜像而不是执行该指令，即缓存。

因此，为了有效的利用缓存，尽量保持Dockerfile一致，并且尽量在末尾修改:

```
FROM ubuntu

MAINTAINER author <somebody@company.com>

RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe"

RUN apt-get update

RUN apt-get upgrade -y
```

更改 `MAINTAINER` 指令会使Docker强制执行 `run` 指令来更新apt，而不是使用缓存。

如不希望使用缓存，在执行 `docker build` 的时候加上参数 `--no-cache=true` 。

Docker匹配镜像决定是否使用缓存的规则如下:

1. 从缓存中存在的基础镜像开始，比较所有子镜像，检查它们构建的指令是否和当前的是否完全一致。如果不一致则缓存不匹配。
2. 多数情况中，使用其中一个子镜像来比较Dockerfile中的指令是足够的。然而，特定的指令需要做更多的判断。
3. `ADD` `COPY` 指令中，将要添加到镜像中的文件也要被检查。通常是检查文件的校验和(checksum)。
4. 缓存匹配检查并不检查容器中的文件。例如，当使用 `RUN apt-get -y update` 命令更新了容器中的文件，并不会被缓存检查策略作为缓存匹配的依据。

使用标签

除非是在用Docker做实验，否则你应当通过 `-t` 选项来 `docker build` 新的镜像以便于标记构建的镜像。一个简单可读的标签可以帮助管理每个创建的镜像。

```
docker build -t="tuxknight/luckypython"
```

Note

始终通过 `-t` 标记来构建镜像。

公开端口

Docker的核心概念是可重复和可移植，镜像应该可以运行在任何主机上并运行尽可能多的次数。在Dockerfile中你可以映射私有和公有端口，但永远不要通过Dockerfile映射公有端口。这样运行多个镜像的情况下会出现端口冲突的问题。

```
EXPOSE 80:8080  # 80映射到host的8080，不提倡这种用法

EXPOSE 80 # 80会被docker随机映射一个端口
```

CMD ENTRYPOINT语法

`CMD` 和 `ENTRYPOINT` 支持两种语法:

24.03%

```
CMD /bin/echo

CMD ["/bin/echo"]
```

在第一种方式下，Docker会在命令前加上 `/bin/sh -c` ,可能会导致一些意想不到的问题。第二种方式下 `CMD` `ENTRYPOINT` 是一个数组，执行的命令完全和你期待的一样。

容器是短暂的

容器模型是进程而不是机器，不需要开机初始化。在需要时运行，不需要是停止，能够删除后重建，并且配置和启动的最小化。

.dockerignore 文件

在 `docker build` 的时候，忽略部分无用的文件和目录可以提高构建的速度。

不要在构建中升级版本

不在容器中更新，更新交给基础镜像来处理。

每个容器只运行一个进程

一个容器只运行一个进程。容器起到了隔离应用隔离数据的作用，不同的应用运行在不同的容器让集群的纵向扩展以及容器的复用都变的更加简单。

最小化层数

需要掌握号Dockerfile的可读性和文件系统层数之间的平衡。控制文件系统层数的时候会降低Dockerfile的可读性。而Dockerfile可读性高的时候，往往会导致更多的文件系统层数。

使用小型基础镜像

`debian:jessie` 基础镜像足够小并且没有包含任何不需要的包。

使用特定标签

Dockerfile中 `FROM` 应始终包含依赖的基础镜像的完整仓库名和标签，如使用 `FROM debian:jessie` 而不是 `FROM debian` 。

多行参数顺序

`apt-get update` 应与 `apt-get install` 组合，采取 `\` 进行多行安装。同时需要注意按照字母表顺序排序避免出现重复。


```
FROM debian:jessie

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

build-essential\

git \

make \

python \

RUN dpkg-reconfigure locales && \

locale-gen C.UTF-8 && \

/usr/sbin/update-locale LANG=C.UTF-8

ENV LC_ALL C.UTF-8
```

Dockerfile指令最佳实践

FROM

使用官方仓库中的镜像作为基础镜像，推荐使用 `Debian image` ,大小保持在100mb上下，且仍是完整的发行版。

RUN

把复杂的或过长的 RUN 语句写成以 `\` 结尾的多行的形式，以提高可读性和可维护性。

`apt-get update` 和 `apt-get install` 一起执行，否则 `apt-get install` 会出现异常。

避免运行 `apt-get upgrade` 或 `dist-upgrade` ，在无特权的容器中，很多 *必要* 的包不能正常升级。如果基础镜像过时了，应当联系维护者。 推荐 `apt-get update && apt-get install -y package-a package-b` 这种方式，先更新，之后安装最新的软件包。

```
RUN apt-get update && apt-get install -y \
    aufs-tools \
    automake \
    btrfs-tools \
    build-essential \
    curl \
    dpkg-sig \
    git \
    iptables \
    libapparmor-dev \
    libcap-dev \
    libsqlite3-dev \
    lxc=1.0* \
    mercurial \
    parallel \
    reprepro \
    ruby1.9.1 \
    ruby1.9.1-dev \
    s3cmd=1.1.0*
```

24.03%

CMD

CMD 推荐使用 `CMD ["executable","param1","param2"]` 这样的格式。如果镜像是用来运行服务，需要使用 `CMD["apache2","-DFOREGROUND"]` ,这种格式的指令适用于任何服务性质的镜像。

ADD COPY

虽然 `ADD` 与 `COPY` 功能类似，但推荐使用 `COPY` 。 `COPY` 只支持基本的文件拷贝功能，更加的可控。而 `ADD` 具有更多特定，比如tar文件自动提取，支持URL。 通常需要提取tarball中的文件到容器的时候才会用到 `ADD` 。

如果在Dockerfile中使用多个文件，每个文件应使用单独的 `COPY` 指令。这样，只有出现文件变化的指令才会不使用缓存。

为了控制镜像的大小，不建议使用 `ADD` 指令获取URL文件。正确的做法是在 `RUN` 指令中使用 `wget` 或 `curl` 来获取文件，并且在文件不需要的时候删除文件。

```
RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.gz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

VOLUME

`VOLUME` 指令应当暴露出数据库的存储位置，配置文件的存储以及容器中创建的文件或目录。由于容器结束后并不保存任何更改，你应该把所有数据通过 `VOLUME` 保存到host中。

USER

如果服务不需要特权来运行，使用 `USER` 指令切换到非root用户。使用 `RUN groupadd -r mysql && useradd -r -g mysql mysql` 之后用 `USER mysql` 切换用户

要避免使用 `sudo` 来提升权限，因为它带来的问题远比它能解决的问题要多。如果你确实需要这样的特性，那么可以选择使用 `gosu` 。

最后，不要反复的切换用户。减少不必要的layers。

WORKDIR

`WORKDIR` 的路径始终使用绝对路径可以保证指令的准确和可靠。 同时，使用 `WORKDIR` 来替代 `RUN cd ... && do-something` 这样难以维护的指令。