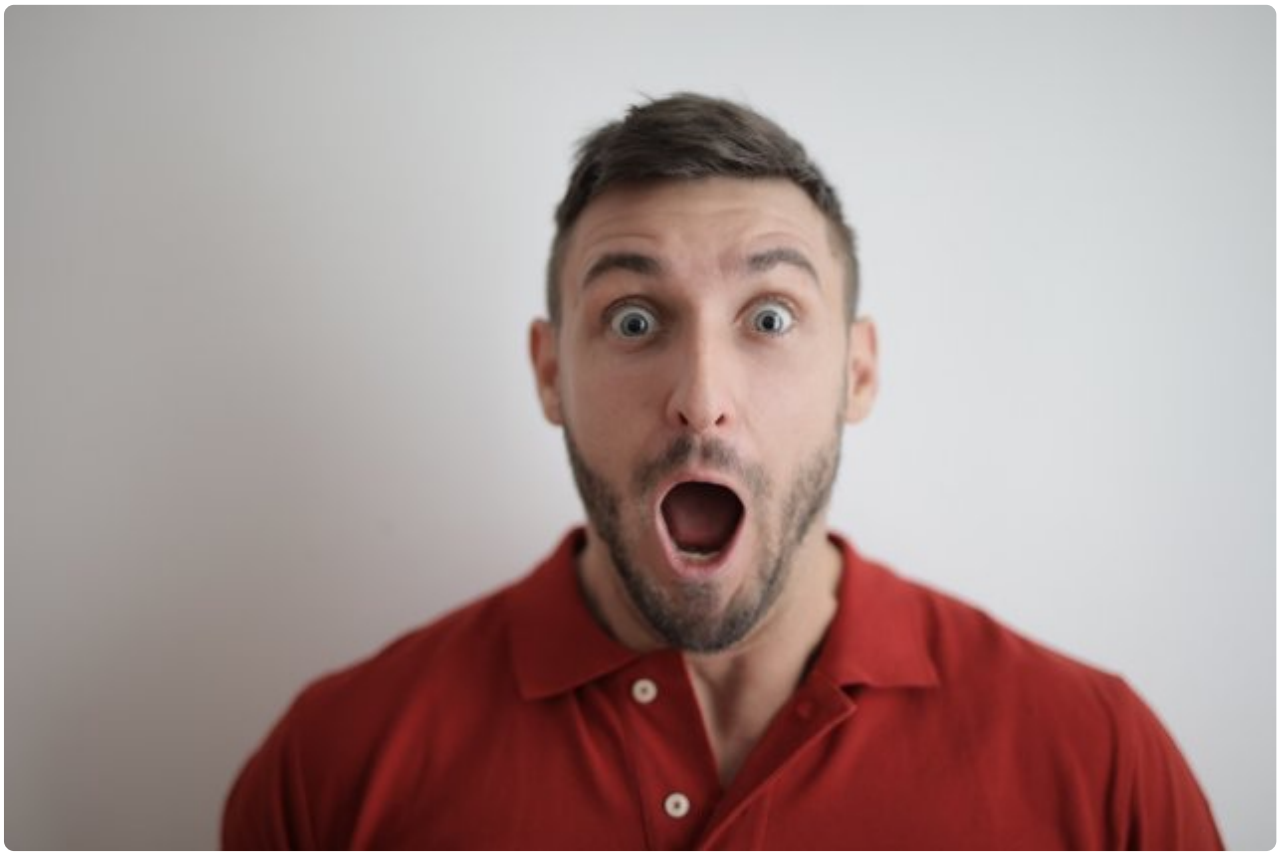
22 Dockerfile 你真的会用吗？

更新时间：2020-09-09 09:55:26



> 没有引发任何行动的思想都不是思想，而是梦想。—— 马丁

Dockerfile 最佳实践

我们前面了解到 Docker 会根据 Dockerfile 中指令构建出镜像，关于 Dockerfile 的指令规范可以参考我们之前的文章。Docker 镜像是由多个只读的文件层（layer）组成的，Dockerfile 中的每个指令会生成一个层，而且层是以 delta 增量的形式组织的。下面我们就来介绍一下在使用 Dockerfile 过程中的最佳实践。

FROM

任何时候，base 镜像尽量使用官方的镜像，比如 Alpine 镜像，作为一个完整的 Linux 发行版，大小不足 5MB。

LABEL

我们可以通过给镜像添加 label 来管理我们的镜像，比如记录 license 信息等。下面是集中比较好的 LABEL 编写格式。

```
# Set one or more individual labels
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=""
```

```
# 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

RUN 指令后面可以接任何命令，当 RUN 后面接的命令太长时，我们可以将命令拆成多行，从而使我们的 dockerfile 可读性更好。

apt-get

RUN 指令的一个典型应用就是和 apt-get 结合起来使用，我们这里看一下 apt-get 的使用注意事项。

不要在 dockerfile 中使用 RUN apt-get upgrade 或者 dist-upgrade ，因为 upgrade 会升级镜像中安装的所有包（如果包有更新的话）。取而代之的是，我们可以使用 apt-get update 获取更新的软件包列表，然后如果确定要升级的话再使用 apt-get install -y foo 去自动更新。

将 apt-get udpate 和 apt-get install 写到一条 RUN 的指令中，也就是像下面这样。

```
RUN apt-get update && apt-get install -y \
    package-bar \
    package-baz \
    package-foo
```

如果把 apt-get update 和 apt-get install 分开编写的话可能会因为 docker build cache 的问题导致没有安装最新的包，举个例子。

```
FROM ubuntu:18.04
RUN apt-get update
RUN apt-get install -y curl
```

通过 docker build 之后，上面 dockerfile 生成的所有文件层都在 Docker cache 中。如果你之后想安装其他的软件包，比如 nginx，然后将 dockerfile 修改成如下的样子。

```
FROM ubuntu:18.04
RUN apt-get update
RUN apt-get install -y curl nginx
```

重新执行 docker build 的时候，由于 cache 的原因，RUN apt-get update 这一行并不会被重新执行，也就是说我们可能会 apt-get install 安装的不是最新版本软件包。

将 apt-get update 和 apt-get install 写在一行就是典型的 ***cache-busting*** 技术。

使用 pipes

有些 RUN 指令后面的命令涉及的 Linux 的管道（pipe），比如将一个命令执行的输出作为下一个命令的输入。比如下面这个例子：先 wget 下载一个文件，然后使用 wc 统计行数。

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

Docker 执行 `RUN` 后面的指令是使用 `/bin/sh -c` 来执行的，对于上面的管道情况，只会把最后一个命令的返回值来作为整个管道链接起来的这条命令的返回值。也就是说上面这条 dockerfile 的指令，只要 `wc -l` 执行成功 Docker 就认为这条指令 `docker build` 成功了。但是这个不是符合预期的，比如前面的 `wget` 执行失败，应该导致 build 失败才是预期的。

为了解决这个问题，或者说解决此类问题：对于管道中的任何阶段的命令失败都导致 build 失败，我们可以使用 `set -o pipefail` 来解决。

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

CMD

`CMD` 指令用于执行镜像中包含的软件，可以带参数。`CMD` 大多数情况都应该以 `CMD ["executable", "param1", "param2"]` 的形式使用。比如说 Apache 服务镜像，我们可以执行类似于 `CMD ["apache2", "DFOREGROUND"]` 形式的命令。

对于一些其他的 case，`CMD` 应该提供一个交互式的 shell，比如 Linux 的 bash，python 或者 perl。比如 `CMD ["perl", "-de0"]` ，`CMD ["python"]` 和 `CMD ["php", "-a"]` 。一旦设置这种形式的 `CMD`，当我们以类似 `docker run -ti <image>` 的形式启动容器时，容器启动之后会自动进入一个可用的 shell。

`CMD` 的另外一种形式 `CMD ["param", "param"]` 只有和 `ENTRYPOINT` 结合使用的情况下才会用这种形式。

EXPOSE

`EXPOSE` 指令用来指定容器暴露的端口。对于一些默认的服务镜像，我们应该尽量使用这些服务的默认端口。比如 Apache web server 使用 80 端口：`EXPOSE 80` ，MongoDB 使用 `EXPOSE 27017` 。

我们在容器中暴露了端口之后，`docker run` 的时候就可以在参数中通过端口隐射的方式将端口暴露到宿主机上。

ENV

`ENV` 指令用来指定镜像中的环境变量。比如对于 nginx 镜像我们可以将 nginx 的 bin 加到环境变量 PATH 中，然后 CMD 指定 nginx 就可以直接使用了。

```
ENV PATH /usr/local/nginx/bin:$PATH
...
CMD ["nginx"]
```

`ENV` 除了设置用户的自定义环境变量，有时候还可以用来设置版本号，类似于我们编程中的常量。

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC /usr/src/postgress && …
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

ADD 和 COPY

`ADD` 和 `COPY` 的功能非常类似，但是一般优先使用 `COPY` ，因为 `COPY` 的功能更单一，只是将本地文件拷贝到容器中，而 `ADD` 还包括压缩文件解压和以 URL 指定的远程文件支持。

`ADD` 的最佳实践是将本地的 tar 文件提取到镜像中，例如 `ADD rootfs.tar.xz` ，这里所说的提取包括拷贝和解压。

如果需要拷贝多个文件，那么在 dockerfile 文件中最好每次拷贝一个单独的文件，这样的好处是我们可以利用 Docker 的 build cache，每次一个文件变化只会影响单个层的 build cache 失效。举个例子。

```
COPY . /tmp/
RUN ...
```

上面的 dockerfile 只要当前目录的任何一个文件变化都会导致 `COPY . /tmp` 层重新构建，导致后面的指令的 build cache 缓存失效。

为了让镜像尽量小，最好不要使用 `ADD` 指令从远程 URL 获取包，而是使用 `curl` 或者 `wget` 先下载包，使用完之后将包删除掉。

ENTRYPOINT

`ENTRYPOINT` 的最佳实践是设置镜像的主命令，使用该镜像启动容器的时候将会执行 `ENTRYPOINT` 中指定的命令。`CMD` 可以作为 `ENTRYPOINT` 的补充，指定主命令的默认参数。

```
ENTRYPOINT ["s3cmd"]
CMD ["--help"]
```

`ENTRYPOINT` 还可以结合一个辅助脚本使用，下面是 Postgres 官方镜像使用的脚本和 `ENTRYPOINT` 设置。

```
#!/bin/bash
set -e

if [ "$1" = 'postgres' ]; then
    chown -R postgres "$PGDATA"

    if [ -z "$(ls -A "$PGDATA")" ]; then
        gosu postgres initdb
    fi

    exec gosu postgres "$@"
fi

exec "$@"
```

```
COPY ./docker-entrypoint.sh /

ENTRYPOINT ["/docker-entrypoint.sh"]
```

shell 脚本的意思当启动参数中的第一个参数是 postgres 时，会做一些和 $PGDATA 相关的工作，最后再调用 Linux 的系统命令 exec 执行所有参数。借助于这个 `ENTRYPOINT` ，我们就可以以多种方式启动容器。

```
$ docker run postgres
# 或者
$ docker run postgres postgres --help
# 或者
$ docker run --rm -it postgres bash
```

VOLUME

VOLUME 使用用来存储任何数据库存储文件、配置文件和 Docker 容器创建的文件。强烈建议使用 VOLUME 来管理镜像中的可变数据。

USER

如果某个服务不需要使用 root 用户执行时，建议使用 USER 指令切换到非 root 用户。使用 USER 之前，要先创建用户或者用户组，类似如下命令。

```
RUN groupadd -r postgres && useradd -r -g postgres postgres
```

最后，为了减少层数和复杂度，避免频繁使用 USER 来回切换用户。

WORKDIR

为了 dockerfile 的可读性， WORKDIR 应该尽量使用绝对路径。对于类似 RUN cd ... &&& do-something 的指令应该尽量避免，取而代之用 WORKDIR 来代替。

总结

本篇文章系统地总结了在 Dockerfile 编写中的注意事项和最佳实践，希望大家在日常使用过程中可以作为参考使用。

}

21 数据共享：volume 的使用指南

23 Docker 最佳实践：如何构建最小的镜像