Cizixs Writes Here

Tech, Python, Reading, Life, And Love.

DOCKER • CONTAINER • DOCKERFILE

编写 Dockerfile 的最佳实践

BY CIZIXS MARCH 28, 2017 0 COMMENTS TWEET LIKE +1

简介

我之前写过一篇文章介绍 docker 的镜像基本原理和概念，这篇文章是公司内部培训的一个总结文字，主要介绍在编写 docker 镜像的时候一些需要注意的事项和推荐的做法。

虽然 Dockerfile 简化了镜像构建的过程，并且把这个过程可以进行版本控制，但是不正当的 Dockerfile 使用也会导致很多问题：

- docker 镜像太大。如果你经常使用镜像或者构建镜像，一定会遇到那种很大的镜像，甚至有些能达到 2G 以上
- docker 镜像的构建时间过长。每个 build 都会耗费很长时间，对于需要经常构建镜

像（比如单元测试）的地方这可能是个大问题

- 重复劳动。多次镜像构建之间大部分内容都是完全一样而且重复的，但是每次都要做一遍，浪费时间和资源

这篇文章会讲述一些做法，希望能解决这些问题。

希望读者能够对 docker 镜像有一定的了解，阅读这篇文章至少需要一下前提知识：

- 了解 docker 的基础概念，运行过容器
- 熟悉 docker 镜像的基础知识，知道镜像的分层结构
- 最好是负责过某个 docker 镜像的构建（使用 docker build 命令创建过自己的镜像）

Dockerfile 和镜像构建

Dockerfile 是由一个个指令组成的，每个指令都对应着最终镜像的一层。每行的第一个单词就是命令，后面所有的字符串是这个命令的参数，关于 Dockerfile 支持的命令以及它们的用法，可以参考官方文档，这里不再赘述。

当运行 `docker build` 命令的时候，整个的构建过程是这样的：

1. 读取 Dockerfile 文件发送到 docker daemon
2. 读取当前目录的所有文件（context），发送到 docker daemon
3. 对 Dockerfile 进行解析，处理成命令加上对应参数的结构
4. 按照顺序循环遍历所有的命令，对每个命令调用对应的处理函数进行处理
5. 每个命令（除了 FROM）都会在一个容器执行，执行的结果会生成一个新的镜像
6. 为最后生成的镜像打上标签

编写 Dockerfile 的一些最佳实践

1. 使用统一的 base 镜像

有些文章讲优化镜像会提倡使用尽量小的基础镜像，比如 busybox 或者 alpine 等。我更推荐使用统一的大家比较熟悉的基础镜像，比如 ubuntu，centos 等，因为基础镜像只需要下载一次可以共享，并不会造成太多的存储空间浪费。它的好处是这些镜像的生态比较完整，方便我们安装软件，除了问题进行调试。

2. 动静分离

经常变化的内容和基本不会变化的内容要分开，把不怎么变化的内容放在下层，创建出来不同基础镜像供上层使用。比如可以创建各种语言的基础镜像，python2.7、python3.4、go1.7、java7等等，这些镜像包含了最基本的语言库，每个组可以在上面继续构建应用级别的镜像。

3. 最小原则：只安装必需的东西

很多人构建镜像的时候，都有一种冲动——把可能用到的东西都打包到镜像中。要遏制这种想法，镜像中应该只包含必需的东西，任何可以有也可以没有的东西都不要放到里面。因为镜像的扩展很容易，而且运行容器的时候也很方便地对其进行修改。这样可以保证镜像尽可能小，构建的时候尽可能快，也保证未来的更快传输、更省网络资源。

4. 一个原则：每个镜像只有一个功能

不要在容器里运行多个不同功能的进程，每个镜像中只安装一个应用的软件包

和文件，需要交互的程序通过 pod（kubernetes 提供的特性）或者容器之间的网络进行交流。这样可以保证模块化，不同的应用可以分开维护和升级，也能减小单个镜像的大小。

5. 使用更少的层

虽然看起来把不同的命令尽量分开来，写在多个命令中容易阅读和理解。但是这样会导致出现太多的镜像层，而不好管理和分析镜像，而且镜像的层是有限的。尽量把相关的内容放到同一个层，使用换行符进行分割，这样可以进一步减小镜像大小，并且方便查看镜像历史。

6. 减少每层的内容

尽管只安装必须的内容，在这个过程中也可能会产生额外的内容或者临时文件，我们要尽量让每层安装的东西保持最小。

- 比如使用 `--no-install-recommends` 参数告诉 `apt-get` 不要安装推荐的软件包
- 安装完软件包，清楚 `/var/lib/apt/list/` 缓存
- 删除中间文件：比如下载的压缩包
- 删除临时文件：如果命令产生了临时文件，也要及时删除

7. 不要在 Dockerfile 中单独修改文件的权限

因为 docker 镜像是分层的，任何修改都会新增一个层，修改文件或者目录权限也是如此。如果有一个命令单独修改大文件或者目录的权限，会把这些文件复制一份，这样很容易导致镜像很大。

解决方案也很简单，要么在添加到 Dockerfile 之前就把文件的权限和用户设置好，要么在容器启动脚本（entrypoint）做这些修改，或者拷贝文件和修改权

限放在一起做（这样最终也只是增加一层）。

8. 利用 cache 来加快构建速度

如果 Docker 发现某个层已经存在了，它会直接使用已经存在的层，而不会重新运行一次。如果你连续运行 `docker build` 多次，会发现第二次运行很快就结束了。

不过从 1.10 版本开始，Content Addressable Storage 的引入导致缓存功能的实效，目前引入了 `--cache-from` 参数可以手动指定一个镜像来使用它的缓存。

9. 版本控制和自动构建

最好把 Dockerfile 和对应的应用代码一起放到版本控制中，然后能够自动构建镜像。这样的好处是可以追踪各个版本镜像的内容，方便了解不同镜像有什么区别，对于调试和回滚都有好处。

另外，如果运行镜像的参数或者环境变量很多，也要有对应的文档给予说明，并且文档要随着 Dockerfile 变化而更新，这样任何人都能参考着文档很容易地使用镜像，而不是下载了镜像不知道怎么用。

参考资料

- Best Practices for writing Dockerfiles
- Refactoring a Dockerfile for Image Size
- How to Not Be the Engineer Running 3.5GB Docker Images

Previous Next