上一篇文章介绍了如何构建最小的镜像文件，这篇文章我们看一下其他的最佳实践，包括 tag 如何使用等。

1. 镜像 tag 的使用

慎用 latest 标签

准确来说除了 demo 或者测试使用，正式环境应该禁止使用 latest 标签。因为 latest 标签表示的是镜像的最新版本，也就是说是一直变化的。这是慎用 latest 标签的根本原因。

举个例子，线上应用依赖的一个基础镜像，比如 ubuntu，使用了 latest 的标签，那么随着基础镜像的不断发布（ubuntu 系统官方是 6 个月发布一个系统，一般都是类似 19.04、19.10，表示 19 年 4 月和 10 月的版本），我们的线上应用在不同时间 build 的时候就可能依赖的是不同的环境，很有可能引入问题。

tag 尽量指定到具体的版本

什么叫具体的版本，比如很多系统的版本都是通过三个数字来表示的，比如 x.y.z。其中 x 是主版本号，一般当软件整体重写时，或出现不向后兼容的改变等重大更新时，增加 X，同时重置 Y、Z 为 0，X 为 0 时表示软件还在开发阶段；y 是次版本号，一般我们说的系统 release 都是保持 x 不变，增加 y，同时重置 z 为 0；z 是修订号，一般主要用于 bugfix。

我们这里说的指定到具体的版本就是指定到 z，很多系统的镜像 tag 可能同时存在 x，x.y，x.y.z 这三种形式的 tag，我们使用的时候在情况允许的情况下尽量指定到 x.y.z 版本号。

2. 应用程序数据持久化

尽量不要将应用程序的数据存储在镜像中。因为这样会增加镜像的大小，并且从 IO 的角度看也显得不够高效。

怎么理解呢？比如我们应用程序依赖一些数据，我们可以把数据打到镜像里面，这种是最省事的，但是这种方式就会带来前面说的问题。

一种推荐的方式是使用 Docker 的数据卷或者目录挂载的方式。其中目录挂载的方式是将容器内的某个目录和宿主机做映射，这种方式一般推荐在应用开发的时候使用，并不建议线上使用。

比如我们要快速调试我们的应用程序，那么我们可以将可执行文件存储在宿主机中，通过目录挂载的方式映射到容器中，这样每次本地开发出一个新的版本就可以快速在容器中进行测试。

另外我们根据应用程序数据的不同，我们可以再细化我们的技术方案。像数据库连接密码这种比较敏感的数据可以存储在 secret 中，而不敏感的数据，比如配置文件，可以使用 config 。

关于 Docker 的 secret 和 config 这里就不再展开叙述了。实际上现在很少有线上单独使用 Docker 的场景了，更多是结合 Kubernetes 来部署和管理我们的容器化应用，secret 和 config 对应到 Kubernetes 中就是 Secret 和 ConfigMap，我们在后面一章专门讨论 Kubernetes 相关技术的时候再进行详述。

3. 通过 CI/CD 的方式进行开发和测试

当我们使用 Docker 技术来开发和部署我们的应用的时候，我们不仅要测试程序本身的正确性，还要测试应用在 Docker 容器化之后的正确性。而应用容器化的过程毫无疑问将整个开发测试链路拉长了，我们一般可以像下面这么几步来做：

1. 本地 build 应用或者拷贝代码到容器中进行 build 生成可执行文件；
2. 使用 docker build 编译生成我们的镜像；
3. 通过 docker push 将镜像 push 到私有或者公有的镜像中心；
4. 测试环境拉取新的镜像进行测试。

我这里简化了操作流程和步骤，还是有 4 个步骤，在实际操作的时候很容易问题。所以更建议以一种自动化的 CI/CD 的方式来进行部署和测试，比如 Jenkins。

Jenkins 是一个广泛用于持续构建的工具，展开来说，就是各种项目的 "自动化" 编译、打包、分发部署等。Jenkins 可以很好的支持各种语言（比如：java, c#, php 等）的项目构建，也完全兼容 ant、maven、gradle 等多种第三方构建工具，同时跟 svn、git 能无缝集成，也支持直接与知名源代码托管网站，比如 github、bitbucket 直接集成。

4. 安全性

下面是几条和安全性相关的最佳实践。

base 镜像最小原则

Docker 安全问题一个常见的原因就是使用的 base 镜像有安全问题。就像程序员开发中的那句话，代码写的越多，bug 越多一样，base 镜像越大则越容易暴露安全问题。针对这个问题，我们可以考虑在 base 镜像中只安装必要的依赖。

最小用户权限原则

如果 Dockerfile 中没有指定 USER，则默认是以 root 用户运行，但是很多情况下我们的 Docker 应用并不是真正需要 root 权限。Docker 以 root 权限启动，映射到宿主机上也具有 root 权限，而 root 权限很容易带来更多的安全问题。

为了解决这个问题，一个比较好的解决方案是在 Dockerfile 中指定特定的用户，先添加用户，然后再使用 USER 指令显示指定，下面是一个例子：

1. 创建一个没有密码、没有 home 目录、没有 shell 的系统用户；
2. 将该系统用户加到一个已经存在的用户组里面；
3. 通过 `USER` 指令显示设置我们的 Docker 应用的启动用户。

```
FROM ubuntu
RUN mkdir /app
RUN groupadd -r lirantal && useradd -r -s /bin/false -g lirantal lirantal
WORKDIR /app
COPY . /app
RUN chown -R lirantal:lirantal /app
USER lirantal
CMD node index.js
```

有些 Base 镜像考虑到了这个用户的问题，已经帮我们把这件事情做了，比如 node 镜像就帮我们提前创建了一个叫 node 的用户。

```
FROM node:10-alpine
RUN mkdir /app
COPY . /app
RUN chown -R node:node /app
USER node
CMD [“node”, “index.js”]
```

不要在 Docker 镜像中泄露敏感信息

我们有时间将应用打包成 Docker 镜像时，会需要一些 secret，比如 SSH private key 用来从私有的代码托管仓库上拉取代码、或者其他需要进行认证的情况。有些同学会直接将这些 secret 拷贝到镜像中，这是一个非常错误的做法。

使用 Docker secret 命令

secret 是 Docker 中的比较新的功能，专门用来处理敏感数据。但是使用起来也是比较简单的，下面是一个例子。

```
FROM alpine


RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecre


RUN --mount=type=secret,id=mysecret,dst=/foobar cat /foobar
```

5. 使用 linter

---

linter 是一种代码规范检测工具，比如写过 Go 语言的应该都只有有一个小工具叫 golint。同样的，对于 Dockerfile 的编写我们也可以使用 linter。这里要推荐的是 hadolint，下面是 hadolint 的简介。

> A smarter Dockerfile linter that helps you build best practice Docker images. The linter is parsing the Dockerfile into an AST and performs rules on top of the AST. It is standing on the shoulders of ShellCheck to lint the Bash code inside `RUN` instructions.

简单翻译一下。

> hadolint 是一款智能的 Dockerfile 检测工具，可以帮助我们在构建 Docker 镜像的时候遵从最佳实践。工作原理是将 Dockerfile 解析成一个抽象语法树，然后对该抽象语法树应用一系列的规则。hadolint 参考自 ShellCheck。

hadolint 使用起来也比较简单，需要先安装。

```
$ brew install hadolint
$ docker run --rm -i hadolint/hadolint < Dockerfile
```

下面是 hadolint 的检测结果示例。

```
DL4000 Specify a maintainer of the Dockerfile
DL3006 Always tag the version of an image explicitely.
FROM debian
SC1007 Remove space after = if trying to assign a value (for empty string, use var='' ... ).
SC2154 node_verson is referenced but not assigned.
DL3009 Delete the apt-get lists after installing something
RUN node_version= "0.10" \
 && apt-get update && apt-get -y install nodejs="$node_verson"
COPY package.json usr/src/app
DL3003 Use WORKDIR to switch to a directory
RUN cd /usr/src/app \
 && npm install node-static

DL3011 Valid UNIX ports range from 0 to 65535
EXPOSE 80000
CMD ["npm", "start"]
```

6. 总结

本文总结了 Docker 最佳实践的几条规则，希望大家在 Docker 使用过程中，可以举一反三。

}