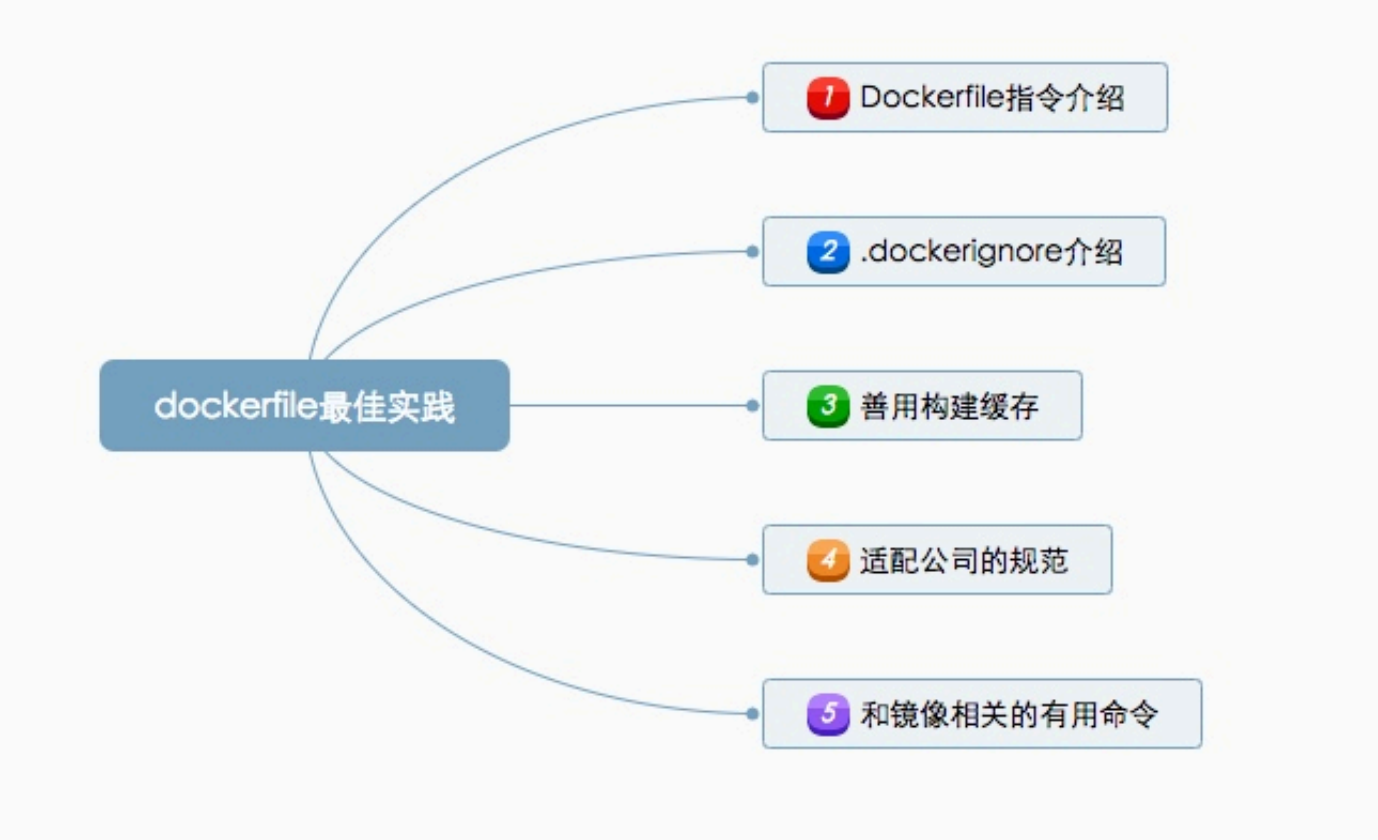
Dockerfile最佳实践

本次分享大纲如下：



Dcokerfile涉及到指令介绍

在Dockerfile中最常用的指令有以下几个：

- FROM指令，FROM就是指定基础镜像，因此一个 Dockerfile 中 FROM 是必备的指令，并且必须是第一条指令。比如说，我要基于ubuntu 14.04的镜像的基础上进行构建应用，因此第一行我会写

  ```
  FROM ubuntu:14.04
  ```

- ADD指令，将本地文件添加到镜像中，比如我需要添加一个python的requirements.txt用来初始化环境，我会写

  ```
  ADD requirements.txt /home/root
  ```

- COPY指令，和ADD类似，就是拷贝本地路径的文件到镜像中，ADD 指令和 COPY 的格式

和性质基本一致。但是在 COPY 基础上增加了一些功能。比如如果ADD的是一个URL路径ADD会自动进行下载等，目前ADD的这些高级功能比较隐晦不建议大家使用。我这里再举例COPY的用法，比如我想把app.py的文件拷贝到/usr/local/bin下，我会这么写

```
COPY app.py /usr/local/bin/
```

- RUN指令，执行一个命令行操作，类似Linux的shell，举例，我要在镜像中安装一些依赖环境，想通过yum安装python3，我会这么写，注意不要执行交互式命令，注意yum的-y参数

```
RUN yum install -y python3
```

- CMD指令，这个指令是运行容器时启动服务的入口配置，比如，我想基于supervisord启动容器，我会如下的写法

```
CMD ["/usr/local/bin/supervisord", "-c", "/path/to/app/supervisord.conf"
]
```

- ENTRYPOINT指令，启动容器的进程，经常和CMD配合使用，比如经常在Linux下执行命令的时候其实默认的是用shell的父进程在执行当前进程，因此这个入口程序我们可以拿shell在举例如下

```
ENTRYPOINT ["/bin/bash"]
```

- WORKDIR指令，指定代码的默认工作目录，也就是你进入容器的时候默认的路径，就好比我们Linux下分配完用户后，默认进入系统的路径是/home/用户名的路径。比如，我想我的工作路径为/data/www，那我可以设置如下

```
WORKDIR /data/www
```

- ENV指令，设定容器运行的时候的环境变量，比如我想设置一个环境变量ENV_APP_PORT，我启动我的进程的时候读取环境变量，如果存在就以这个环境变量设置的端口启动，我可以这样写

```
ENV ENV_APP_PORT 8899
```

- LABEL指令，给启动的容器打上一些标签，主要用来表示这个容器的一些特征，比如我启动的这个容器是掌阅用户组的，那么我可以打一个如下的标签，举例：

```
LABEL group="zhangyue.user"
LABEL version="1.0"
```

以上就是Dockerfile中最常用的一些指令，在开发过程中基本涉及到的大多数都可以用以上指令配合完成工作。

善用.dockerignore

和`.gitignore`一样，如果不想某些文件被打包到镜像可以在根目录设置`.dockerignore`
以下是一个内部的`.dockerignore`举例，大家看看语法结构

```
# git
.git
.gitignore
# python
*.pyc
__pycache__/
*/__pycache__/
*/*/__pycache__/
*/*/*/__pycache__/
*.py[cod]
*/*.py[cod]
*/*/*.py[cod]
*/*/*/*.py[cod]
# project
script
test
tpl
readme.txt
requirements
*.log
# Docker
docker-compose.yml
.docker
Dockerfile
# Vim swap files
*.swp
*/*.swp
*/*/*.swp
*/*/*/*.swp
# Python mode for VIM
.ropeproject
*/.ropeproject
*/*/.ropeproject
```

```
*/*/*/.ropeproject
# PyCharm
.idea
# Virtual environment
.env
# Installer logs
pip-log.txt
pip-delete-this-directory.txt
# CI
.codeclimate.yml
.travis.yml
```

`.dockerignore`的作用就是避免一些与应用无关的文件加载到镜像中，毕竟构建镜像的时候任何多余的文件都会使得镜像的体积变大。因此我们要善于利用`.dockerignore`文件把不必要的临时文件进行屏蔽掉。

善用构建缓存

构建docker镜像的时候，大家需要去理解镜像有一个分层的概念，什么是分层，如何判断我的镜像有多少层？我告诉大家一个简单的办法，你在Dockerfile里面使用了多少指令大概就是增加了多少个分层，为什么说是增加了多少个分层呢，是因为FROM指令导入的镜像也有自己的分层，因此总共的层数就是FROM导入镜像的层数加上新增的层数就是这个镜像总的层数。

减少镜像的分层有利于降低镜像构建的大小，降低镜像的大小有利于镜像分发和加快启动速度。因此我们要尽可能减少镜像分层。接下来我就给大家分享一些小技巧用来降低镜像的分层。

- 在docker1.10及更高版本RUN,COPY,ADD会创建新的分层，其他指令只创建中间临时镜像并不会直接影响最终的镜像大小。
- 合并多行参数比如安装软件包不要每个软件包都写一个RUN，而是合并起来

```
RUN apt-get update && apt-get install -y \
     bzr \
     cvs \
     git \
     mercurial \
     subversion
```

- 合并多行LABEL

```
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"
```

- 及时清理垃圾文件

```
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/*
```

编写Dockerfile的注意事项

- 善于使用管道PIPES避免环境构建无效，比如执行一个命令的时候，Docker判断这个命令是否成功执行是基于这个命令的返回值来进行，如果返回值是0就认为成功，否则就认为失败，而管道符号很容易把指令的正确返回值给屏蔽掉，导致后续的命令在前面命令执行失败的情况下继续执行而没有发生中断。

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

  如何来避免这个情况呢？我来给大家一个推荐的写法，就是增加set -o pipefail，举例如下：

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

- 默认的shell执行环境是/bin/sh，而/bin/bash和/bin/sh是有一些区别的，如果想指定shell执行命令可以按照如下方法：

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

- 合理的选择ADD还是COPY

  ADD和COPY指令很相似，不过COPY指令仅仅把文件拷贝到镜像层而ADD指令还有一些额外的功能，比如把文件解压缩`ADD rootfs.tar.xz /`，执行下载`ADD http://example.com/big.tar.xz /usr/src/things/`，不过官方不建议使用ADD的方式来下载文件，因为如果用curl或者wget的话可以在一个层内下载完文件并解压后删除不必要的tar包，而ADD指令下载需要有两个分层执行这些逻辑。

  如果镜像有多个层次，执行COPY时可以把COPY分成多次，避免一次COPY完成，因为文件发生变更的话会导致cache失效，如果COPY一个没有修改的文件的话是会有构建缓存的，如果COPY的是一大堆文件的话，任何一个文件变更都会导致cache失效

- 明确指定容器内的用户和组权限

  为了安全考虑如果不指定用户和组权限默认用的root账户，因此可能有一定风险，如果一个容器可以在非root用户下启动的话建议都指定下用户，比如：

  ```
  RUN groupadd -r postgres && useradd --no-log-init -r -g postgres postgres
  ```

  其中--no-log-init参数是为了解决一个历史遗留的bug，这个bug发生在go 打包tar时会产生一个日志到/var/log/faillog，如果没有设置这个参数日志文件会越来越多，不过Debian/Ubuntu系统目前不支持这个指令。

- 避免使用sudo指令，可以考虑用gosu指令替代

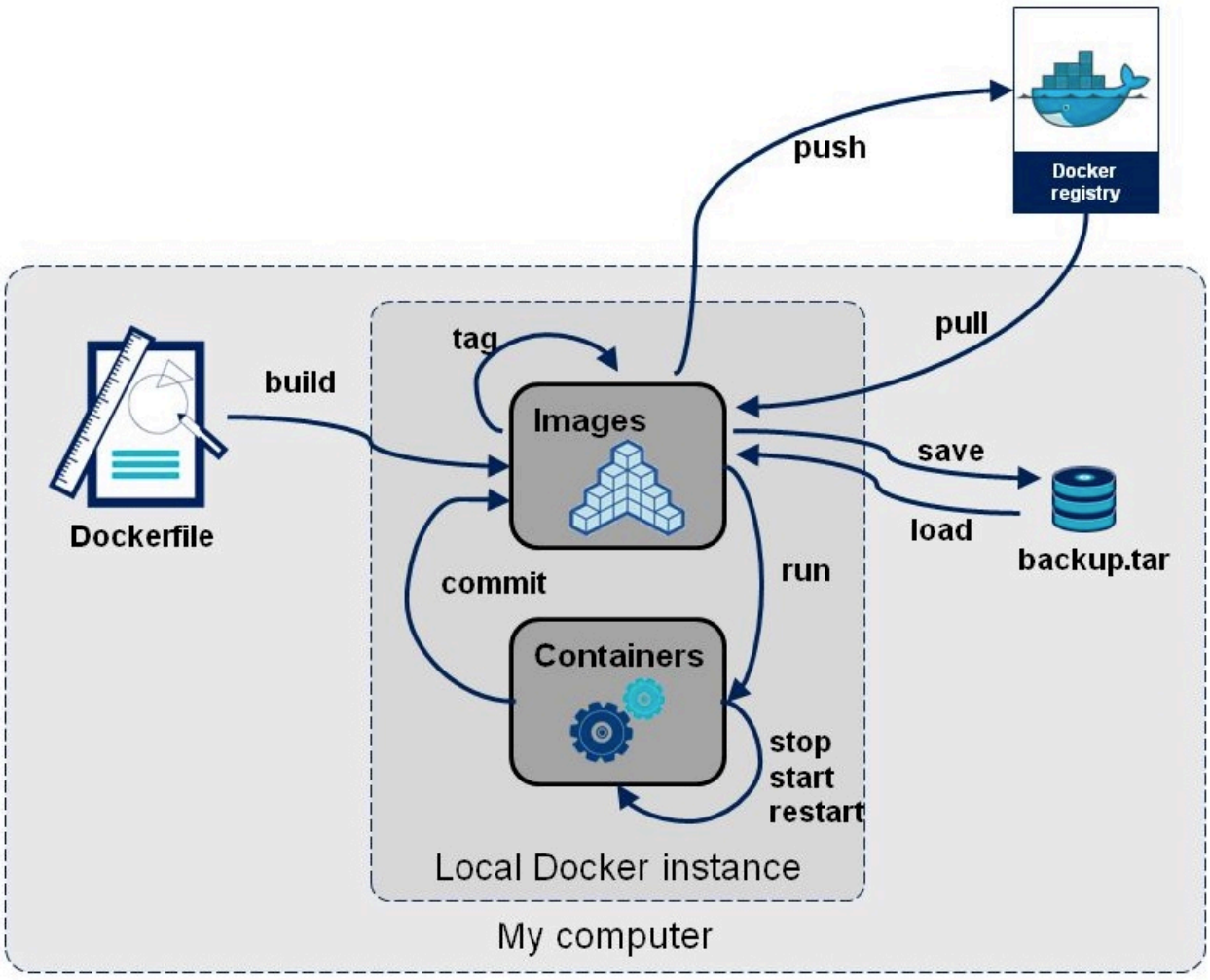
适配公司的规范

每个公司内部都会有一些公司内的约定，比如：

- 如何给Dockerfile打造出来的镜像命名，命名的方式，版本号如何约定
- 是否允许打latest标签的镜像推送到仓库
- 是否需要删除镜像构建的中间层
- 镜像中哪些环境变量和标签不能随意使用，比如，内部服务发现或者docker-compose会基于容器打一些标签是否会冲突等都需要命名上进行规范

以上规范每个公司可能都会有差异，我提出一些点了方便大家扩展思路，就不在一一列举出来所有场景。

和镜像有关的命令



- 和docker仓库相关的命令有docker push，docker pull
- 和外部tar文件相关的命令有docker save，docker load
- 基于Dockerfile构建镜像相关的命令有docker build
- 构建完镜像后给镜像打tag的命令docker tag
- 基于运行时的容器生成镜像的命令docker commit
- 基于镜像启动容器的命令docker run

每个命令的用法都可以通过docker <对应命令> -h 查看其帮助信息，由于命令较多，我不再挨个介绍其功能，我以一个命令的帮助信息给大家展示下，其余的都是类似的，举例如下：

```
➜  sh git:(master) ✗ docker build -h

Usage:  docker build [OPTIONS] PATH | URL | -

Build an image from a Dockerfile

  --build-arg=[]                    Set build-time variables
  --cpu-shares                      CPU shares (relative weight)
  --cgroup-parent                   Optional parent cgroup for the container
  --cpu-period                      Limit the CPU CFS (Completely Fair Scheduler) period
  --cpu-quota                       Limit the CPU CFS (Completely Fair Scheduler) quota
  --cpuset-cpus                     CPUs in which to allow execution (0-3, 0,1)
  --cpuset-mems                     MEMs in which to allow execution (0-3, 0,1)
  --disable-content-trust=true      Skip image verification
  -f, --file                        Name of the Dockerfile (Default is 'PATH/Dockerfile')
  --force-rm                        Always remove intermediate containers
  --help                            Print usage
  --isolation                       Container isolation level
  -m, --memory                      Memory limit
  --memory-swap                     Swap limit equal to memory plus swap: '-1' to enable unlimited swap
  --no-cache                        Do not use cache when building the image
  --pull                            Always attempt to pull a newer version of the image
  -q, --quiet                       Suppress the build output and print image ID on success
  --rm=true                         Remove intermediate containers after a successful build
  --shm-size                        Size of /dev/shm, default value is 64MB
  -t, --tag=[]                      Name and optionally a tag in the 'name:tag' format
  --ulimit=[]                       Ulimit options
  -v, --volume=[]                   Set build-time bind mounts
```

声明

- 本文章出自头条号：《docker进击之路之Dockerfile最佳实践》，更多关于docker相关技术文章请在今日头条搜索：docker进击之路
- 作者：MagickKing
- docker进击之路官方QQ交流群