Dockerfile 文件全面详解

mp.weixin.qq.com/s/o62WE6dG7GrOKp7JDPk3SA

原文链接：https://zhuanlan.zhihu.com/p/387855002

DevOps技术栈
专注于分享DevOps工具链及经验总结，例如：Docker、K8s、ServiceMesh、Jenkins等技术栈。

66篇原创内容

公众号

Docker 可以通过读取 Dockerfile 中的指令自动构建镜像。Dockerfile 是一个文本文档，其中包含了用户创建镜像的所有命令和说明。

一、 变量

变量用 $variable_name 或者 ${variable_name} 表示。

- ${variable:-word} 表示如果 variable 设置，则结果将是该值。如果 variable 未设置，word 则将是结果。
- ${variable:+word} 表示如果 variable 设置则为 word 结果，否则为空字符串。

变量前加 \ 可以转义成普通字符串：\$foo or \${foo}，表示转换为 $foo 和 ${foo} 文字。

二、FROM

初始化一个新的构建阶段，并设置基础镜像：

```
FROM [--platform=<platform>] <image> [AS <name>]
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

- 单个 Dockfile 可以多次出现 FROM，以使用之前的构建阶段作为另一个构建阶段的依赖项
- AS name 表示为构建阶段命名，在后续 FROM 和 COPY --from=<name> 说明中可以使用这个名词，引用此阶段构建的映像
- digest 其实就是就是根据镜像内容产生的一个 ID，只要镜像的内容不变 digest 也不会变
- tag 或 digest 值是可选的。如果您省略其中任何一个，构建器默认使用一个 latest 标签。如果找不到该 tag 值，构建器将返回错误。
- --platform 标志可用于在 FROM 引用多平台镜像的情况下指定平台。例如，linux/amd64、linux/arm64、 或 windows/amd64。

三、RUN

将在当前镜像之上的新层中执行命令，在 docker build时运行。

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
```

RUN 有两种形式：

- RUN<command>（shell 形式，命令在 shell 中运行，默认 /bin/sh -c 在 Linux 或 cmd /S /CWindows 上）
- RUN ["executable", "param1", "param2"]（执行形式）

说明：

- 可以使用 \（反斜杠）将单个 RUN 指令延续到下一行
- RUN 在下一次构建期间，指令缓存不会自动失效。可以使用 --no-cache 标志使指令缓存无效
- Dockerfile 的指令每执行一次都会在 Docker 上新建一层。所以过多无意义的层，会造成镜像膨胀过大，可以使用 && 符号连接命令，这样执行后，只会创建 1 层镜像

四、CMD

运行程序，在 docker run 时运行，但是和 run 命令不同，RUN 是在 docker build 时运行。

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

支持三种格式：

- CMD ["executable","param1","param2"] 使用 exec 执行，推荐方式；
- CMD command param1 param2 在 /bin/sh 中执行，提供给需要交互的应用；
- CMD ["param1","param2"] 提供给 ENTRYPOINT 的默认参数。

指定启动容器时执行的命令，每个 Dockerfile 只能有一条 CMD 命令。如果指定了多条命令，只有最后一条会被执行。

如果用户启动容器时候指定了运行的命令，则会覆盖掉 CMD 指定的命令。

五、LABEL

添加元数据：

```
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

六、EXPOSE

```
EXPOSE <port> [<port>/<protocol>...]
```

Docker 容器在运行时侦听指定的网络端口。可以指定端口是监听TCP还是UDP，如果不指定协议，默认为TCP。

该 EXPOSE 指令实际上并未发布端口。要在运行容器时实际发布端口，docker run -P 来发布和映射一个或多个端口。

默认情况下，EXPOSE 假定 TCP。您还可以指定 UDP：

```
EXPOSE 80/udp
```

七、ENV

设置环境变量：

```
ENV <key>=<value> ...
```

设置的环境变量将持续存在，您可以使用 docker inspect 来查看。使用 docker run --env <key>=<value> 来更改环境变量的值。

如果环境变量只在构建期间需要，请考虑为单个命令设置一个值：

```
RUN DEBIAN_FRONTEND=noninteractive apt-get update && apt-get install -y ...
```

或者使用 ARG，它不会保留在最终镜像中：

```
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y ...
```

八、ADD

复制新文件、目录或远程文件 URL <src> ，并将它们添加到 <dest> 中。

<src> 可以指定多个资源，但如果它们是文件或目录，则它们的路径被解释为相对于构建上下文的源，也就是 WORKDIR。

每个都 <src> 可能包含通配符，匹配将使用 Go 的 filepath.Match 规则。例如：

添加所有以“hom”开头的文件：

```
ADD hom* /mydir/
```

在下面的示例中，? 被替换为任何单个字符，例如“home.txt”。

```
ADD hom?.txt /mydir/
```

<dest> 是一个绝对路径，或相对 WORKDIR 的相对路径。

九、COPY

语法同ADD一致，复制拷贝文件。

COPY 指令和 ADD 指令的唯一区别在于：是否支持从远程URL获取资源。COPY 指令只能从执行 docker build 所在的主机上读取资源并复制到镜像中。而 ADD 指令还支持通过 URL 从远程服务器读取资源并复制到镜像中。

相同需求时，推荐使用 COPY 指令。ADD 指令更擅长读取本地tar文件并解压缩。

十、ENTRYPOINT

ENTRYPOINT 和 CMD 一样，都是在指定容器启动程序及参数，不过它不会被 docker run 的命令行参数指定的指令所覆盖。如果要覆盖的话，需要通过 docker run --entrypoint 来指定。

它有2种格式：

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

指定了 ENTRYPOINT 后， CMD 的内容作为参数传给 ENTRYPOINT 指令，实际执行时，将变为：

```
<ENTRYPOINT> <CMD>
```

十一、VOLUME

创建一个具有指定名称的挂载数据卷。

```
VOLUME ["/var/log/"]
VOLUME /var/log
```

它的主要作用是：

- 避免重要的数据，因容器重启而丢失
- 避免容器不断变大

十二、ARG

定义变量，与 ENV 作用相同，不过 ARG 变量不会像 ENV 变量那样持久化到构建好的镜像中。

```
ARG <name>[=<default value>]
```

Docker 有一组预定义的 ARG 变量，您可以在 Dockerfile 中没有相应指令的情况下使用这些变量。

- HTTP_PROXY
- http_proxy
- HTTPS_PROXY
- https_proxy
- FTP_PROXY
- ftp_proxy
- NO_PROXY
- no_proxy

要使用这些，请使用 --build-arg 标志在命令行上传递它们，例如：

```
docker build --build-arg HTTPS_PROXY=https://my-proxy.example.com .
```

十三、ONBUILD

将一个触发指令添加到镜像中，以便稍后在该镜像用作另一个构建的基础时执行。也就是另外一个 dockerfile FROM 了这个镜像的时候执行。

```
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
```

十四、STOPSIGNAL

设置将发送到容器退出的系统调用信号。该信号可以是与内核系统调用表中的位置匹配的有效无符号数，例如 9，或格式为 SIGNAME 的信号名称，例如 SIGKILL。

```
STOPSIGNAL signal
```

默认的 stop-signal 是 SIGTERM，在 docker stop 的时候会给容器内 PID 为 1 的进程发送这个 signal，通过 --stop-signal 可以设置自己需要的 signal，主要目的是为了让容器内的应用程序在接收到 signal 之后可以先处理一些事物，实现容器的平滑退出，如果不做任何处理，容器将在一段时间之后强制退出，会造成业务的强制中断，默认时间是 10s。

十五、HEALTHCHECK

用于指定某个程序或者指令来监控 Docker 容器服务的运行状态。该 HEALTHCHECK 指令有两种形式：

- HEALTHCHECK [OPTIONS] CMD command（通过在容器内运行命令来检查容器健康状况）
- HEALTHCHECK NONE（禁用从基础镜像继承的任何健康检查）

十六、SHELL

覆盖用于命令的 shell 形式的默认 shell。Linux 上的默认 shell 是 ["/bin/sh", "-c"]，Windows 上是 ["cmd", "/S", "/C"]。

```
SHELL ["executable", "parameters"]
```

该 SHELL 指令在 Windows 上特别有用，因为 Windows 有两种常用且截然不同的本机 SHELL：cmd 和 powershell，以及可用的备用 shell，包括 sh。该 SHELL 指令可以出现多次。每条 SHELL 指令都会覆盖所有先前的 SHELL 指令，并影响所有后续指令。

十七、WORKDIR

工作目录，如果 WORKDIR 不存在，即使它没有在后续 Dockerfile 指令中使用，它也会被创建。

docker build 构建镜像过程中，每一个 RUN 命令都会新建一层。只有通过 WORKDIR 创建的目录才会一直存在。

可以设置多个 WORKDIR，如果提供了相对路径，它将相对于前一条 WORKDIR 指令的路径。例如：

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

最终 pwd 命令的输出是 /a/b/c。

该 WORKDIR 指令可以解析先前使用 ENV，例如：

```
ENV DIRPATH=/path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

最终 pwd 命令的输出是 /path/$DIRNAME。

十八、USER

设置用户名（或 UID）和可选的用户组（或 GID）。

```
USER <user>[:<group>]
USER <UID>[:<GID>]
```