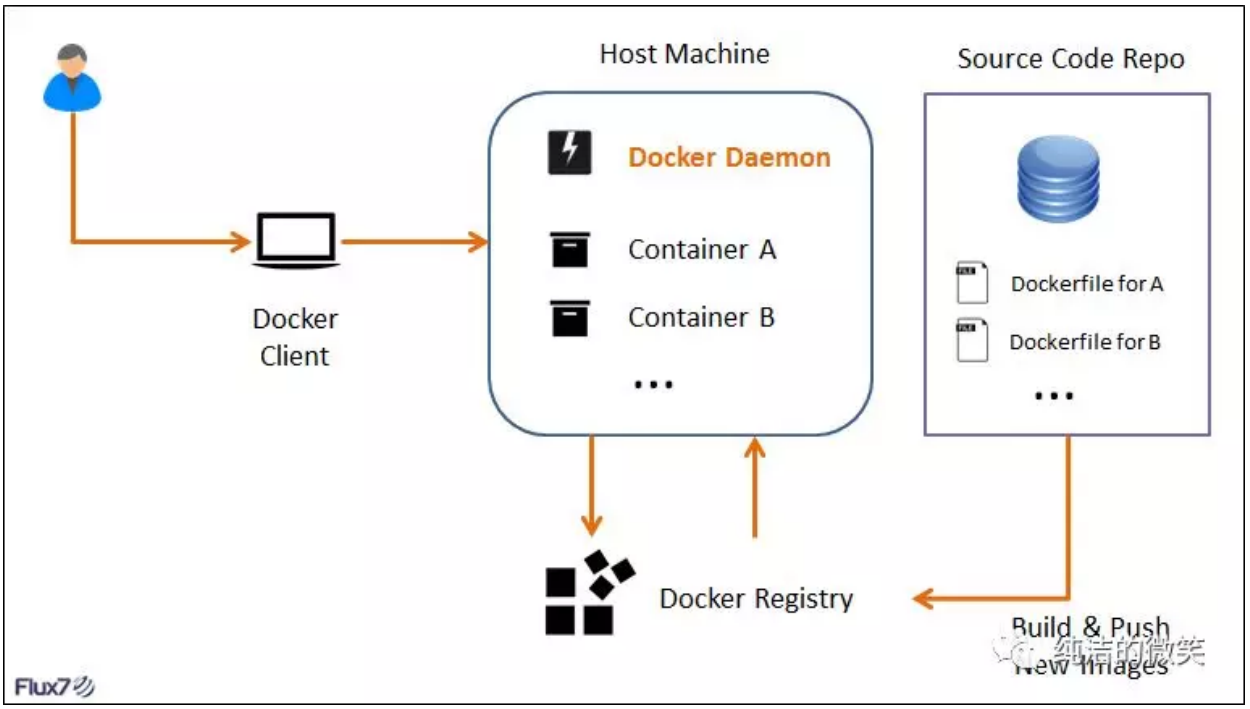
Docker(二)：Dockerfile 使用介绍

原创： 纯洁的微笑 纯洁的微笑 2018-03-13

上一篇文章Docker(一)：Docker入门教程介绍了 Docker 基本概念，其中镜像、容器和 Dockerfile 。我们使用 Dockerfile 定义镜像，依赖镜像来运行容器，因此 Dockerfile 是镜像和容器的关键，Dockerfile 可以非常容易的定义镜像内容，同时在我们后期的微服务实践中，Dockerfile 也是重点关注的内容，今天我们就来一起学习它。

首先通过一张图来了解 Docker 镜像、容器和 Dockerfile 三者之间的关系。



通过上图可以看出使用 Dockerfile 定义镜像，运行镜像启动容器。

Dockerfile 概念

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像的定制实际上就是定制每一层所添加的配置、文件。如果我们可以把每一层修改、安装、构建、操作的命令都写入一个脚本，用这个脚本来构建、

定制镜像，那么之前提及的无法重复的问题、镜像构建透明性的问题、体积的问题就都会解决。这个脚本就是 Dockerfile。

Dockerfile 是一个文本文件，其内包含了一条条的指令(Instruction)，每一条指令构建一层，因此每一条指令的内容，就是描述该层应当如何构建。

有了 Dockerfile，当我们需要定制自己额外的需求时，只需在 Dockerfile 上添加或者修改指令，重新生成 image 即可，省去了敲命令的麻烦。

Dockerfile 文件格式

Dockerfile文件格式如下：

```
## Dockerfile文件格式
# This dockerfile uses the ubuntu image
# VERSION 2 - EDITION 1
# Author: docker_user
# Command format: Instruction [arguments / command] ..
# 1、第一行必须指定 基础镜像信息
FROM ubuntu
# 2、维护者信息
MAINTAINER docker_user docker_user@email.com
# 3、镜像操作指令
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/sources.list
RUN apt-get update && apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf
# 4、容器启动执行指令
CMD /usr/sbin/nginx
```

Dockerfile 分为四部分：**基础镜像信息、维护者信息、镜像操作指令、容器启动执行指令**。一开始必须要指明所基于的镜像名称，接下来一般会说明维护者信息；后面则是镜像操作指令，例如 RUN 指令。每执行一条RUN指令，镜像添加新的一层，并提交；最后是 CMD 指令，来指明运行容器时的操作命令。

构建镜像

docker build 命令会根据 Dockerfile 文件及上下文构建新 Docker 镜像。构建上下文是指 Dockerfile 所在的本地路径或一个URL（Git仓库地址）。构建上下文环境会被递归处理，所以构建所指定的路径还包括了子目录，而URL还包括了其中指定的子模块。

将当前目录做为构建上下文时，可以像下面这样使用docker build命令构建镜像：

```
1. docker build .
2. Sending build context to Docker daemon 6.51 MB
3. ...
```

说明：构建会在 Docker 后台守护进程（daemon）中执行，而不是 CLI 中。构建前，构建进程会将全部内容（递归）发送到守护进程。大多情况下，应该将一个空目录作为构建上下文环境，并将 Dockerfile 文件放在该目录下。

在构建上下文中使用的 Dockerfile 文件，是一个构建指令文件。为了提高构建性能，可以通过 .dockerignore文件排除上下文目录下不需要的文件和目录。

在 Docker 构建镜像的第一步，docker CLI 会先在上下文目录中寻找 .dockerignore文件，根据 .dockerignore 文件排除上下文目录中的部分文件和目录，然后把剩下的文件和目录传递给 Docker 服务。

Dockerfile 一般位于构建上下文的根目录下，也可以通过 -f指定该文件的位置：

```
1. docker build -f /path/to/a/Dockerfile .
```

构建时，还可以通过 -t参数指定构建成镜像的仓库、标签。

镜像标签

```
1. docker build -t nginx/v3 .
```

如果存在多个仓库下，或使用多个镜像标签，就可以使用多个 -t参数：

```
1. docker build -t nginx/v3:1.0.2 -t nginx/v3:latest .
```

在 Docker 守护进程执行 Dockerfile 中的指令前，首先会对 Dockerfile 进行语法检查，有语法错误时会返回：

```
1. docker build -t nginx/v3 .
2. Sending build context to Docker daemon 2.048 kB
3. Error response from daemon: Unknown instruction: RUNCMD
```

缓存

Docker 守护进程会一条一条的执行 Dockerfile 中的指令，而且会在每一步提交并生成一个新镜像，最后会输出最终镜像的ID。生成完成后，Docker 守护进程会自动清理你发送的上下文。 Dockerfile文件中的每条指令会被

独立执行，并会创建一个新镜像，RUN cd /tmp等命令不会对下条指令产生影响。 Docker 会重用已生成的中间镜像，以加速docker build的构建速度。以下是一个使用了缓存镜像的执行过程：

```
$ docker build -t svendowideit/ambassador .
Sending build context to Docker daemon 15.36 kB
Step 1/4 : FROM alpine:3.2
 ---> 31f630c65071
Step 2/4 : MAINTAINER SvenDowideit@home.org.au
 ---> Using cache
 ---> 2a1c91448f5f
Step 3/4 : RUN apk update && apk add socat && rm -r /var/cache/
 ---> Using cache
 ---> 21ed6e7fbb73
Step 4/4 : CMD env | grep _TCP= | (sed 's/.*_PORT_\([0-9]*\)_TCP=tcp:\/\/\(.*\):\
(.*\)/socat -t 100000000 TCP4-LISTEN:\1,fork,reuseaddr TCP4:\2:\3 \&' && echo wait) | sh
 ---> Using cache
 ---> 7ea8aef582cc
Successfully built 7ea8aef582cc
```

构建缓存仅会使用本地父生成链上的镜像，如果不想使用本地缓存的镜像，也可以通过 `--cache-from`指定缓存。指定后将不再使用本地生成的镜像链，而是从镜像仓库中下载。

寻找缓存的逻辑

Docker 寻找缓存的逻辑其实就是树型结构根据 Dockerfile 指令遍历子节点的过程。下图可以说明这个逻辑。

```
 FROM base_image:version Dockerfile:
 +----------+ FROM base_image:version
 |base image| RUN cmd1 --> use cache because we found base image
 +-----X----+ RUN cmd11 --> use cache because we found cmd1
 / \
 / \
 RUN cmd1 RUN cmd2 Dockerfile:
 +------+ +------+ FROM base_image:version
 |image1| |image2| RUN cmd2 --> use cache because we found base image
 +---X--+ +------+ RUN cmd21 --> not use cache because there's no child node
 / \ running cmd21, so we build a new image here
 / \
RUN cmd11 RUN cmd12
+-------+ +-------+
|image11| |image12|
+-------+ +-------+
```

大部分指令可以根据上述逻辑去寻找缓存，除了 ADD 和 COPY 。这两个指令会复制文件内容到镜像内，除了指令相同以外，Docker 还会检查每个文件内容校验(不包括最后修改时间和最后访问时间)，如果校验不一致，则不会使用缓存。

除了这两个命令，Docker 并不会去检查容器内的文件内容，比如 `RUN apt-get -y update`，每次执行时文件可能都不一样，但是 Docker 认为命令一致，会继续使用缓存。这样一来，以后构建时都不会再重新运行 `apt-get -y update`。

如果 Docker 没有找到当前指令的缓存，则会构建一个新的镜像，并且之后的所有指令都不会再去寻找缓存。

简单示例

接下来用一个简单的示例来感受一下 Dockerfile 是如何用来构建镜像启动容器。我们以定制 nginx 镜像为例，在一个空白目录中，建立一个文本文件，并命名为 Dockerfile：

```
1. mkdir mynginx
2. cd mynginx
3. vi  Dockerfile
```

构建一个 Dockerfile 文件内容为：

```
1. FROM nginx
2. RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
3. vi  Dockerfile
```

这个 Dockerfile 很简单，一共就两行涉及到了两条指令：FROM 和 RUN，FROM 表示获取指定基础镜像，RUN 执行命令，在执行的过程中重写了 nginx 的默认页面信息，将信息替换为：Hello, Docker!。

在 Dockerfile 文件所在目录执行：

```
1. docker build  -t nginx:v1  .
```

命令最后有一个. 表示当前目录

构建完成之后，使用 `docker images` 命令查看所有镜像，如果存在 REPOSITORY 为 nginx 和 TAG 是 v1 的信息，就表示构建成功。

```
1. docker images
2. REPOSITORY TAG IMAGE ID CREATED SIZE
3. nginx v1  8c92471de2cc  6  minutes ago  108.6  MB
```

接下来使用 docker run 命令来启动容器

```
1. docker run  --name docker_nginx_v1  -d  -p  80:80  nginx:v1
```

这条命令会用 nginx 镜像启动一个容器，命名为 `docker_nginx_v1`，并且映射了 80 端口，这样我们可以用浏览器去访问这个 nginx 服务器： `http://192.168.0.54/`，页面返回信息：

这样一个简单使用 Dockerfile 构建镜像，运行容器的示例就完成了！

修改容器内容

容器启动后，需要对容器内的文件进行进一步的完善，可以使用 `docker exec -it xx bash`命令再次进行修改，以上面的示例为基础，修改 nginx 启动页面内容：

```
1. root@3729b97e8226:/# echo '<h1>Hello, Docker neo!</h1>' > /usr/share/nginx/html/index.html
2. root@3729b97e8226:/# exit
3. exit
```

以交互式终端方式进入 docker*nginx*v1 容器，并执行了 bash 命令，也就是获得一个可操作的 Shell。然后，我们用 `<h1>Hello, Docker neo!</h1>`覆盖了 `/usr/share/nginx/html/index.html` 的内容。

再次刷新浏览器，会发现内容被改变。

修改了容器的文件，也就是改动了容器的存储层，可以通过 docker diff 命令看到具体的改动。

```
1. docker diff docker_nginx_v1
2. ...
```

这样 Dockerfile 使用方式就为大家介绍完了，下期为大家介绍 Dockerfile 命令的详细使用。