2021.3.10 基于 Dockerfile 构建镜像应用

一．镜像概述

Docker 镜像是 docker 容器技术的核心，也是应用打包构建的标准格式。一个完整的镜像可以支撑多个容器的运行，在 docker 的整个过程中，进入一个已经定型的容器之后，就可以在容器内操作，最常见的就是在容器内安装应用服务

如果想要把已经安装的服务进行迁移，就需要把环境以及部署的服务生成新的镜像

二．镜像构建方式

·基于已有的容器创建镜像
·基于本地模块创建镜像
·基于 dockerfile 创建镜像

三．镜像构建案例

1）基于已有的容器创建镜像

基于现有的镜像创建主要使用 docker commit 命令，即把一个容器里面运行的程序以及该程序的运行环境打包起来生成新的镜像

命令格式：docker commit[选项] 容器 ID/名称 仓库名称:[标签]

常用的选项：

-m ：说明信息

-a ：作者信息

-p ：生成过程中停止容器的运行

首先启动一个镜像，在容器里做相应的修改，然后将修改后的容器提交为新的镜像，需要记住该容器的 ID 号

2）

```
[root@server04 ~]# cat centos-7-x86_64.tar.gz | docker import - centos:7 #导入一个镜像
sha256:258156be32cdd678a27a048c670be46473b854f4a675462465b64984ee788279
[root@server04 ~]# docker run -it centos:7 /bin/bash #进入到容器内
[root@fdef3afc09e9 /]# touch centos.txt #准备测试文件
[root@server04 ~]# docker commit -m "zhangzhifeng" -a "zhang" fdef3afc09e9 centos:7 #开始构建镜像
[root@server04 ~]# docker images #查看镜像是否构建成功
REPOSITORY TAG IMAGE ID CREATED SIZE
centos 7 997f244943cd 33 seconds ago 589MB

[root@server04 ~]# docker run -it centos:7 /bin/bash #进入到容器内
[root@218a81c7c480 /]# ls #查看是否有你刚创建的文件
bin centos.txt etc home lib64 media opt root sbin sys usr
boot dev fastboot lib lost+found mnt proc run srv tmp var
```

基于本地模板创建镜像

通过导入操作系统模板文件可以生成镜像，模板可以从 OPENVZ 开源项目下载，

下载地址为：https://download.openvz.org/template/precreated

下面是使用 docker 导入命令将下载的 centos 模块压缩导入文本镜像的例子：

```
[root@server04~]#https://download.openvz.org/template/precreated/centos-7-x86_64.tar.gz
[root@server04 ~]# cat centos-7-x86_64.tar.gz  | docker import - centos:7  #导入 centos7 的镜像
```

3）基于 dockerfile 构建镜像

Docker 镜像构建

镜像并不是一个单一的文件，而是由多层堆叠构成。可以通过 docker history 命令查看镜像中各层的内容和大小，每层对应着 Dockerfile 构建时的一条指令。Docker 镜像默认存储在/var/lib/docker/<storage-driver>目录中。容器其实是在镜像的最上面加了一层读写层，在运行容器中做的任何文件改动，最终都会写到这个读写层。如果删除了容器，也就是删除了这个读写层，文件改动也就会丢失了。Docker 使用存储驱动管理镜像每层内容及可读写层的容器层。

Docker 镜像的特点

·dockerfile 中的每个指令都会创建一个新的镜像层：

·镜像层可以被缓存和复用

·当 dockerfile 中的指令被修改，复制的文件发生变化或者构建镜像时指定的变量值换了，那么对应的镜像层缓存也将会失效

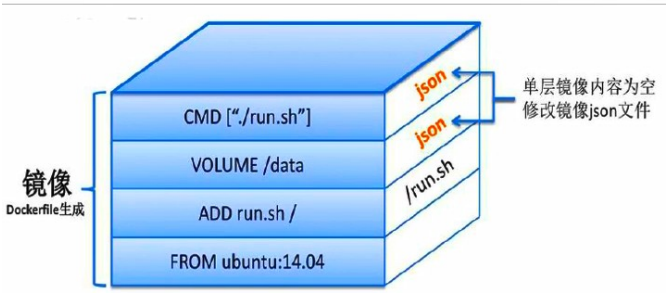
·某一层的镜像缓存失效，它之后的镜像层都会失效

·镜像层是不可变的，如果在某一层中添加一个文件，然后在某一层中删除它，则镜像中依然会包含该文件，只是这个文件中的容器不可见了

四．Dockerfile 介绍

Dockerfile 是一种可以被 docker 程序解释的脚本，dockerfile 由多条指令堆叠组成每条指令都对应 Linux 下面的一条命令。Docker 程序将这些 Dockerfile,指令翻译成真正的 Linux 命令。Dockerfile.有自己书写格式和支持的命令，Docker 程序解决这些命令间的依赖关系，类似于源码软件中的 Makefile。Docker 程序将读取 Dockerfile,并根据指令生成定制的镜像。

Dockerfile.这种显而易见的脚本更容易被使用者接受,它明确的表明镜像是怎么产生的。有了 Dockerfile,当有定制额外的需求时，只需在 Dackerfile.上添加或者修改指令，重新生成镜像即可。



Dockerfile 结构大致分为四个部分：基础镜像信息，维护者信息，镜像操作指令和容器。
Dockerfile 每行支持一条指令，每条指令可携带多个参数，支持使用以“#”号开头注释
dockerfile 的指令根据作用可以分为两种。“构建指令和设置指令”
·构建指令 image，其指定的操作不会在运行 image 的容器上执行
·设置指令用于设置 image 的属性，其指定的操作将运行在 image 的容器中执行
1）FROM 镜像：tag
构建指令：必须指定需要在 dockerfile 文件中其他指令的前面。后续的指令都依赖于该指令指定的 image。FROM 指令指定的基础可以是官方远程仓库中的，也可以位于本地仓库
2）MAINTAINER：作者信息
构建指令：用于将 image 的制作者相关的信息写入 image 中。当我们对该 image 执行 docker inspect 命令时，输出中有相应的字段记录该信息
3）RUN：命令
构建命令：RUN 可以运行任何被基础 image 支持的命令提交到新的镜像中，如基础 image 选择 ubuntu，那么软件管理部分只能使用 ubuntu 的命令
4）CMD[“要运行的程序” “,”参数 1 “,”参数 2“,”参数 3]
设置指令，用于 container（容器）启动时指定的操作。该操作可以是执行自定义的脚本，也可以是执行系统命令。该指令只能在文件中存在一次（最后一行），如果多个 CMD 命令，则只执行最后一条
5）ENTRYPOINT（设置 container 启动命令时执行的操作）
设置指令，指定容器启动时执行的命令，可以多次设置，但是只执行最后一条
6）USER 用户名/UID
设置指令，设置启动容器的用户，默认时 root 用户
7）EXPOSE 端口
设置指令，该指令会将容器中的端口映射成宿主机器中的某个端口。当你需要访问容器的时候，可以不使用容器的 IP 地址而是使用宿主机器的 IP 地址和映射后的端口。要完成整个操作需要两个步骤，首先在 Dockerfile.,使用 EXPOSE 设置需要映射的容器端口，然后在运行容器的时候指定-p 选项加上 EXPOSE 设置的端口，这样 EXPOSE 设置的端口号会被随机映射成宿主机器中的一个端口号。也可以指定需要映射到宿主机器的那个端口，这时要确保宿主机器上的端口号没有被使用。EXPOSE 指令可以一次设置多个端口号，相应的运行容器的时候，可以配套的多次使用-p 选项。
8）ENV 环境变量
构建指令：在 image 中设置一个环境变量
设置了后，后续的 RUN 命令都可以使用，container 启动后，可以通过 docker inspect 查看这个环境变量，也可以通过在 docker run --env key=value 时设置或修改环境变量。假如你安装了 JAVA 程序，需要设置 JAVA_HOME，那么可以在 Dockerfile. 中这样写 :ENV JAVA_HOM/usr/local/java

9）ADD 源文件 目标文件

构建指令：ADD 命令相对于 COPY 命令,可以解压缩文件并把它们添加到镜像中的功能，如果我们有一个压缩文件包，并且需要把这个压缩包中的文件添加到镜像中。需不需要先解开压缩包然后执行 COPY 命令呢?当然不需要!我们可以通过 ADD 命令一次搞定。

同时 ADD 还可以从 url 拷贝文件到镜像中，但官方不推荐这样使用，官方建议我们当需要从远程复制文件时，最好使用 curl 或 wget,命令来代替 ADD 命令。原因是，当使用 ADD 命令时，会创建更多的镜像层，当然镜像的 size 也会更大，所以 ADD 命令官方推荐只有在解压缩文件并把它们添加到镜像中时才需要。

10）COPY 源文件 目标文件

COPY 命令用于将于 Dackerfile 所在目录中的文件在镜像构建阶段从宿主机拷贝到镜像中，对于文件而言可以直接将文件复制到镜像中，对于目录而言，该命令只复制目录中的内容而不包含目录自身

11）VOLUME [“目录”]（指定数据卷的）

设置指令，使容器中的一个目录具有持久化存储数据的功能，该目录可以被容器本身使用，也可以共享给其他容器使用。我们知道容器使用的是 AUFS，这种文件系统不能持久化数据，当容器关闭后，所有的更改都会丢失。当容器中的应用有持久化数据的需求时可以在 Dockerfile 中使用该指令。

12）WORKDIR 目录

设置指令：相当于 cd 命令，为后续 RUN，CMD，ENTRYPOINT 指定工作目录

13）ONBUILD 命令

指定所生产的镜像作为一个基础镜像时所要运行的命令

14）HEALTHCHECK

健康检查

在编写 Dockerfile 时，有严格的格式需要遵循:第一行必须使用 FROM 指令指明所基于的镜像名称;之后使用 MAINTAINER 指令说明维护该镜像的用户信息;然后是镜像操作相关指令，如 RUN 指令。每运行一条指令，都会给基础镜像添加新的一层。最后使用 CMD 指令指定启动容器时要运行的命令操作。

Dockerfile 构建 SSHD 镜像

基于 dockerfile 制作镜像时首先建立工作目录，作为生成镜像的工作目录。然后分别创建并编写 dockerfile 文件，需要运行的脚本文件以及要复制到容器中的文件

实验思路：

1）关闭防火强：

```
[root@server04 ~]# systemctl  stop  firewalld
[root@server04 ~]# setenforce  0
[root@server04 ~]# iptables -F
```

2）关注 DNS 地址

```
[root@server04 ~]# cat /etc/resolv.conf
# Generated by NetworkManager
```

```
nameserver 223.5.5.5
nameserver 223.6.6.6
```

3）导入系统镜像

```
[root@server04 ~]# cat centos-7-x86_64.tar.gz | docker import - centos:7
```

4）建立工作目录

```
[root@server04 ~]# mkdir sshd  #sshd 镜像的工作目录
```

5）采用私钥建立连接，避免密码交互

```
[root@server04 sshd]# mv  /root/.ssh/id_rsa.pub  ./  #将私钥 cp 到当前目录下
```

6）编写 Dockerfile 文件

```
[root@server04 sshd]# vim Dockerfile   #注意 D 必须文大写
FROM centos:7         #基于的镜像

MAINTAINER Crushlinux <crushlinux@163.com>  ：#镜像作者信息

#镜像执行的命令
RUN yum -y install openssh-server net-tools openssh-devel lsof telnet  #安装依赖包
RUN sed -i 's/UsePAM yes/UsePAM no/g' /etc/ssh/sshd_config  #关闭 PAM 验证
RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key   #支持密钥对
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key   #支持密钥对
ADD id_rsa.pub /root/.ssh/authorized_keys         #将公钥放到指定的位置

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime  ：#定义时区（必须要有的固定格式）

EXPOSE 22   ：#开启 22 端口

CMD ["/usr/sbin/sshd" , "-D"]   ：#启动容器时执行指令
```

开始构建

```
[root@server04 sshd]# docker build -t sshd:new  .  #基于当前目录构建镜像
```

```
[root@server04 ~]# docker run  -d -p 2222:22 --name sshd-test --restart=always sshd:new     #将 2222 于宿主机的 22 端口做一个关联
```

测试连接

```
[root@server04 ~]# ssh  localhost -p 2222  #连接 2222 端口
```

Dockerfile 构建 httpd 镜像

建立工作目录

```
[root@server04 ~]#mkdir  httpd
[root@server04 ~]# cd  httpd
[root@server04 ~]#  vim  Dockerfile
[root@server04 sshd]# vim Dockerfile   #注意 D 必须文大写
FROM centos:7         #基于的镜像
```

```
MAINTAINER Crushlinux <crushlinux@163.com>  ：#镜像作者信息

RUN yum -y install httpd
RUN mkdir -p /var/www/html
RUN echo "crushlinux" > /var/www/html/index.html
RUN ls -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

EXPOSE 80
CMD ["httpd","-DFOREGROUND"]
```

构建镜像

```
[root@server04 ~]#  docker  build -t  httpd:new  .
[root@server04 ~]#  docker  images  httpd
```

测试容器：

```
[root@server04 ~]#  docker run  -d -p 8001:80 --name httpd-test --restart=always httpd:new    # #将 8001 于宿主机的 80 端口做一个关联
```