docker的dockerfile文件

原创 Mr大表哥 2017-03-18 17:38:47 评论(0) 243人阅读

博主QQ：819594300
博客地址：http://zpf666.blog.51cto.com/
有什么疑问的朋友可以联系博主，博主会帮你们解答，谢谢支持!

一、根据Dockerfile 构建出一个容器

Dockfile是一种被Docker程序解释的脚本，Dockerfile由一条一条的指令组成，每条指令对应Linux下面的一条命令。Docker程序将这些Dockerfile指令翻译真正的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Makefile。Docker程序将读取Dockerfile，根据指令生成定制的image。相比image这种黑盒子，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image是怎么产生的。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 由一行行命令语句组成，并且支持以 # 开头的注释行。

Dockerfile的指令是忽略大小写的，建议使用大写，每一行只支持一条指令，每条指令可以携带多个参数。

Dockerfile的指令根据作用可以分为两种，构建指令和设置指令。

构建指令用于构建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，其指定的操作将在运行image的容器中执行。

一般的，Dockerfile 分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令。

每运行一条 RUN 指令，镜像添加新的一层，并提交。

最后是 CMD 指令，来指定运行容器时的操作命令。

指令的一般格式为 INSTRUCTION arguments （指令 具体命令），指令包括 FROM 、 MAINTAINER 、 RUN 等。

1）FROM（指定基础image）

属于：构建指令。

作用：必须指定且需要在dockerfile的第一行指定。后续的指令都依赖于该指令指定的image，FROM指令指定的基础image可以是官方远程仓库

中的，也可以位于本地仓库。

两种格式：

FROM <image>

指定基础image为该image的最后修改的版本。

或者：

FROM <image>:<tag>

指定基础image为该image的一个tag版本。 （省略了tag就默认是最后的版本）

2）MAINTAINER（用来指定镜像创建者信息）

属于：构建指令。

作用：用于将image的制作者相关的信息写入到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。

格式：MAINTAINER <name>

3）RUN（安装软件用）

属于：构建指令

作用：RUN可以运行任何被基础image支持的命令。如基础image选择了centos7，那么软件管理部分只能使用centos7的命令。

两种格式：

RUN <command命令> (the commandis run in a shell - `/bin/sh -c`)

RUN ["executable可执行的","param参数1", "param2" ...] (exec form执行形式)

前者将在 shell 终端中运行命令，即 /bin/sh -c ；后者则使用 exec 执行。

指定使用其它终端可以通过第二种方式实现，例如

RUN ["/bin/bash", "-c", "echohello"]

每条 RUN 指令将在当前镜像基础上执行指定命令，并提交为新的镜像。当命令较长时可以使用 \ 来换行。

4）CMD（设置container启动时执行的操作）

属于：设置指令

作用：用于container启动时指定的操作。该操作可以是执行自定义脚本，也可以是执行系统命令。

三种格式：

CMD ["executable可执行的","param1参数","param2"] 使用 exec 执行，推荐方式；

CMD command命令 param参数1 param2 在 /bin/sh 中执行，提供给需要交互的应用；
当Dockerfile指定了ENTRYPOINT（进入点），那么使用下面的格式：
CMD ["param1参数","param2参数"] 提供给 ENTRYPOINT 的默认参数；
说明：ENTRYPOINT指定的是一个可执行的脚本或者程序的路径，该指定的脚本或者程序将会以param1和param2作为参数执行。所以如果CMD指令使用上面的形式，那么Dockerfile中必须要有配套的ENTRYPOINT。
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。

5）ENTRYPOINT（设置container启动时执行的操作）

属于：设置指令
作用：指定容器启动时执行的命令，可以多次设置，但是只有最后一个有效。
两种格式：
ENTRYPOINT ["executable可执行的", "param1参数","param2"]
ENTRYPOINT command命令 param参数1 param2 （shell中执行）。
与cmd的区别：配置容器启动后执行的命令，并且不可被 docker run 提供的参数覆盖。但是启动容器时候执行的是cmd命令，则可以被覆盖。
与cmd的相同点：每个 Dockerfile 中只能有一个 ENTRYPOINT ，当指定多个时，只有最后一个起效。
该指令的使用分为两种情况，一种是独自使用，另一种和CMD指令配合使用。
当独自使用时，如果你还使用了CMD命令且CMD是一个完整的可执行的命令，那么CMD指令和ENTRYPOINT会互相覆盖只有最后一个CMD或者ENTRYPOINT有效。
eg: CMD指令将不会被执行，只有ENTRYPOINT指令被执行
CMD echo “Hello, World!”
ENTRYPOINTls -l
另一种用法和CMD指令配合使用来指定ENTRYPOINT的默认参数，这时CMD指令不是一个完整的可执行命令，仅仅是参数部分；

ENTRYPOINT指令只能使用JSON方式指定执行命令，而不能指定参数。

```
eg:FROM doker.io/centos:centos7
CMD ["-l"]
ENTRYPOINT ["/usr/bin/ls"]
```

6）USER（设置container容器的用户，默认是root用户）

属于：构建指令

作用：指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户。

格式：USER daemon

例如：如： RUN groupadd -r postgres &&useradd -r -g postgres postgres

例如：指定memcached的运行用户

```
ENTRYPOINT ["memcached"]
USER daemon
```

或

```
ENTRYPOINT ["memcached", "-u","daemon"]
```

7）EXPOSE（指定容器需要映射到宿主机器的端口）

属于：设置指令

作用：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，可以不是用容器的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。

格式：EXPOSE <port>[<port>...]

例如：映射一个端口

```
EXPOSEport1
# 相应的运行容器使用的命令
dockerrun -p 宿主机端口:port1    image
```

例如： 映射多个端口
EXPOSE port1 port2 port3
相应的运行容器使用的命令
docker run -p 宿主机端口1:port1 -p 宿主机端口2:port2 -p 宿主机端口3:port3 image
需要指定需要映射到宿主机器上的某个端口
端口映射是docker比较重要的一个功能，原因在于我们每次运行容器的时候容器的IP地址不能指定而是在桥接网卡的地址范围内随机生成的。宿主机器的IP地址是固定的，我们可以将容器的端口的映射到宿主机器上的一个端口，免去每次访问容器中的某个服务时都要查看容器的IP的地址。
对于一个运行的容器，可以使用docker port加上容器中需要映射的端口和容器的ID来查看该端口号在宿主机器上的映射端口。

8）ENV（用于设置环境变量）

属于：构建指令
作用：指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。
格式：ENV <key><value>
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_HOME /path/to/java/dirent
再例如：
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC/usr/src/postgress
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH

9）ADD（从宿主机目录（src）复制文件到container的目录(dest)中去）

属于：构建指令
作用：所有拷贝到container中的文件权限和文件夹权限为0755，uid和gid为0；
如果是一个目录，那么会将该目录下的所有文件添加到container中，不包括目录；

如果文件是可识别的压缩格式，则docker会帮忙解压缩（注意压缩格式）；
如果<src>是文件且<dest>中不使用斜杠结束，则会将<dest>视为文件，<src>的内容会写入<dest>；
如果<src>是文件且<dest>中使用斜杠结束，则会<src>文件拷贝到<dest>目录下。
格式：ADD <src><dest>
该命令将复制指定的<src>到容器中的<dest>。
其中 <src> 可以是Dockerfile所在目录的一个相对路径；也可以是一个 URL；还可以是一个 tar 文件（自动解压为目录）
<dest> 是container中的绝对路径。

10）COPY

属于：设置指令
作用：复制本地主机的 <src>（为 Dockerfile 所在目录的相对路径）到容器中的 <dest>。
格式：COPY <src><dest>
复制本地主机的 <src>（为 Dockerfile 所在目录的相对路径）到容器中的 <dest>。

11）VOLUME（指定挂载点）

属于：设置指令
作用：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。
格式：VOLUME["<mountpoint>"]
例如：FROM base
VOLUME["/tmp/data"]
运行通过该Dockerfile生成image的容器，/tmp/data目录中的数据在容器关闭后，里面的数据还存在。例如另一个容器也有持久化数据的需求，且想使用上面容器共享的/tmp/data目录，那么可以运行下面的命令启动一个容器：
dockerrun -t -i -rm -volumes-from container1 image2 bash
container1为第一个容器的ID，image2为第二个容器运行image的名字。

12）WORKDIR（切换目录）

属于：设置指令

作用：可以多次切换(相当于cd命令)，对RUN,CMD,ENTRYPOINT生效。为后续的 RUN、CMD、ENTRYPOINT 指令配置工作目录。
格式：WORKDIR/path/to/workdir
例如： 在 /p1/p2 下执行 vim a.txt
WORKDIR/p1
WORKDIRp2
RUNvim a.txt
可以使用多个 WORKDIR 指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。
例如：WORKDIR /a
WORKDIRb
WORKDIRc
RUNpwd
则最终路径为 /a/b/c

13）ONBUILD（在子镜像中执行）

属于：构建指令
作用：ONBUILD 指定的命令在构建镜像时并不执行，而是在它的子镜像中执行。配置当所创建的镜像作为其它新创建镜像的基础镜像时，所执行的操作指令。
格式：ONBUILD <Dockerfile关键字>
ONBUILD[INSTRUCTION指令]
例如，Dockerfile 使用如下的内容创建了镜像 image-A 。
[...]
ONBUILDADD . /app/src
ONBUILDRUN /usr/local/bin/python-build --dir /app/src
[...]
如果基于 image-A 创建新的镜像时，新的Dockerfile中使用 FROM image-A 指定基础镜像时，会自动执行ONBUILD 指令内容。
等价于在后面添加了两条指令。
FROMimage-A
#Automaticallyrun the following
ADD. /app/src
RUN/usr/local/bin/python-build --dir /app/src
使用 ONBUILD 指令的镜像，推荐在标签中注明，例如 ruby:1.9-onbuild
编写完成 Dockerfile 之后，可以通过 docker build 命令来创建镜像。

基本的格式为 docker build [选项] 路径 ，该命令将读取指定路径下的 Dockerfile，并将该路径下所有内容发送给 Docker 服务端，由服务端来创建镜像。因此一般建议放置 Dockerfile 的目录为空目录。

要指定镜像的标签信息，可以通过 -t 选项，例如

```
$sudo docker build -t myrepo/myapp /tmp/test1/
```

docker应用案例：使用dockerfile创建sshd镜像模板

准备工作：

①配置宿主IP地址

```
TYPE=Ethernet
BOOTPROTO=static
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=eno16777736
UUID=9ec6e9f5-c342-426c-ae6c-91a159e5a36e
DEVICE=eno16777736
ONBOOT=yes
IPADDR=10.0.0.1
NETMASK=255.0.0.0
~
```

②重启network服务，并查看地址是否正确

```
[root@localhost ~]# systemctl restart network
[root@localhost ~]# ifconfig
eno16777736: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::20c:29ff:fee7:13d3  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:e7:13:d3  txqueuelen 1000  (Ethernet)
        RX packets 1  bytes 258 (258.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 4312 (4.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

③rpm包安装vsftpd

```
[root@localhost ~]# cd /run/media/root/CentOS\ 7\ x86_64/Packages/
[root@localhost Packages]# rpm -ivh vsftpd-3.0.2-10.el7.x86_64.rpm
警告：vsftpd-3.0.2-10.el7.x86_64.rpm: 头V3 RSA/SHA256 Signature, 密钥 ID f4a80eb5: NOKEY
准备中...                          ################################# [100%]
正在升级/安装...
   1:vsftpd-3.0.2-10.el7            ################################# [100%]
[root@localhost Packages]#
```


④防火墙开例外并重载防火墙，然后开启vsftpd服务，并开启路由转发功能

```
[root@localhost ~]# firewall-cmd --permanent --add-service=ftp
success
[root@localhost ~]# firewall-cmd --reload
success
[root@localhost ~]# systemctl start vsftpd
[root@localhost ~]# systemctl enable vsftpd
Created symlink from /etc/systemd/system/multi-user.target.wants/vsftpd.
ftpd.service.
[root@localhost ~]# vim /etc/sysctl.conf
```



```
# For more information,
net.ipv4.ip_forward=1
~
```



```
[root@localhost ~]# sysctl -p
net.ipv4.ip_forward = 1
[root@localhost ~]#
```

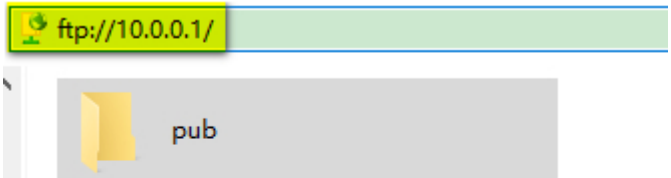

⑤挂载系统盘，挂载点是/var/ftp/pub，同时关闭selinux

```
[root@localhost ~]# mount /dev/sr0 /var/ftp/pub
mount: /dev/sr0 写保护，将以只读方式挂载
[root@localhost ~]#
```



```
[root@localhost ~]# setenforce 0
[root@localhost ~]#
```


⑥在一台客户机上测试ftp能否通



1）创建一个sshd_dockerfile工作目录

```
[root@localhost ~]# mkdir sshd_dockerfile
[root@localhost ~]# cd sshd_dockerfile/
[root@localhost sshd_dockerfile]# touch dockerfile run.sh
[root@localhost sshd_dockerfile]# ls
dockerfile  run.sh
[root@localhost sshd_dockerfile]#
```


编辑run.sh

```
[root@localhost sshd_dockerfile]# vim run.sh
```

```
#!/bin/bash
/usr/sbin/httpd -D DFOREGROUND
/usr/sbin/sshd -D
```

在主机上生成ssh秘钥对，并创建authorized_keys文件

```
[root@localhost ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
29:20:5f:63:dd:46:5b:21:94:fd:36:ab:ab:da:77:ab root@localhost.localdomain
The key's randomart image is:
+--[ RSA 2048]----+
|          .+oo.  |
|        . o.+.   |
|   . . + . +  .  |
|    o + . o   +  |
|     . . S   . o |
|      .      .   |
|             .   |
|        . o .    |
|       ..ooE+..  |
+-----------------+
[root@localhost ~]#
```

三个回车

```
[root@localhost ~]# cat ~/.ssh/id_rsa.pub > /root/sshd_dockerfile/authorized_keys
[root@localhost ~]#
```

2） 编写Dockerfile

```
[root@localhost ~]# vim /root/sshd_dockerfile/dockerfile
```

```
FROM docker.io/centos:latest
MAINTAINER from dabiaoge@example.com
RUN rm -f /etc/yum.repos.d/*
RUN echo "[centos7]" > /etc/yum.repos.d/yum.repo
RUN echo "name=centos7" >> /etc/yum.repos.d/yum.repo
RUN echo "baseurl=ftp://10.0.0.1/pub" >>/etc/yum.repos.d/yum.repo
RUN echo "enabled=1" >> /etc/yum.repos.d/yum.repo
RUN echo "gpgcheck=0" >> /etc/yum.repos.d/yum.repo
RUN yum install -q -y httpd openssh-server sudo
RUN useradd admin
RUN echo "admin:admin" | chpasswd
RUN echo "admin  ALL=(ALL)  ALL" >> /etc/sudoers
RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
RUN ssh-keygen -t ed25519 -f /etc/ssh/ssh_host_ed25519_key
RUN ssh-keygen -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key
RUN mkdir -p /var/run/sshd
RUN mkdir -p /home/admin/.ssh
RUN sed -ri 's/#ServerName www.example.com:80/ServerName www.benet.com/g' /etc/httpd/conf/httpd.conf
ADD authorized_keys /home/admin/.ssh/authorized_keys
ADD run.sh /run.sh
RUN chmod 775 /run.sh
EXPOSE 80 22
CMD ["/run.sh"]
```

以上选项的含义解释：

FROM centos:latest 选择一个已有的os镜像作为基础

MAINTAINER 镜像的作者

```
RUN yum install -y openssh-server sudo   安装openssh-server和sudo软件包
添加测试用户admin，密码admin，并且将此用户添加到sudoers里
RUN useradd admin
RUN echo "admin:admin" | chpasswd
RUN echo "admin   ALL=(ALL)       ALL" >>/etc/sudoers
下面这两句比较特殊，在centos6上必须要有，否则创建出来的容器sshd不能登录
RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
注意：centos7上必须要有，否则创建出来的容器sshd不能登录
RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
RUN ssh-keygen -t ed25519 -f/etc/ssh/ssh_host_ed25519_key
RUN ssh-keygen -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key
将公钥信息上传到远程连接用户的宿主目录的.ssh下
ADD authorized_keys /home/admin/.ssh/authorized_keys
启动sshd服务并且暴露22端口
RUN mkdir /var/run/sshd
EXPOSE 22
CMD ["/run.sh"]        也可以写成这种方式
CMD["/usr/sbin/sshd", "-D"]
```

3)使用docker build命令来创建镜像(在step13~16的时候会报错，但是可以忽略即可)

```
[root@localhost ~]# docker build -t "centos:benet" /root/sshd_dockerfile/
Successfully built df4a03193841
```

在最后一行看见这个说明成功了

4）执行docker images查看新生成的镜像

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              benet               df4a03193841        2 minutes ago       266.4 MB
docker.io/centos    latest              50dae1ee8677        7 months ago        196.7 MB
[root@localhost ~]#
```

5）使用刚才建好的镜像运行一个容器，将容器的端口22映射到主机的2222，80映射到主机的8080

```
[root@localhost ~]# docker run -dit --name=http_ssh -p 8080:80 -p 2222:22 centos:benet
c115c4c01c008a7b322c9e7e7234f64175941254785 48d642b7b93eccb0b163d
[root@localhost ~]#
```

6）在宿主主机打开一个终端，连接刚才新建的容器

```
[root@localhost ~]# docker exec -it http_ssh /bin/bash
[root@c115c4c01c00 /]# yum -y install net-tools
```

安装ifconfig软件


```
[root@localhost ~]# ssh admin@10.0.0.1 -p 2222
The authenticity of host '[10.0.0.1]:2222 ([10.0.0.1]:2222)' can't be established.
ECDSA key fingerprint is 66:14:18:8f:d6:b1:b4:f7:7f:5e:dd:32:8e:33:1c:8e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.0.0.1]:2222' (ECDSA) to the list of known hosts.
[admin@c115c4c01c00 ~]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:2  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 184  bytes 333229 (325.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 142  bytes 17470 (17.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[admin@c115c4c01c00 ~]$
```