轻量级虚拟化技术

5.1 教学目标

1. 能力目标

(1) 能够根据项目实际,运用 Docker 设计资源隔离运行环境解决方案,以支持开发复杂软件系统的解决方案。

(2) 能够根据项目需求,在容器和虚拟机技术之间进行准确评估和恰当选择,以解决复杂工程的关键问题。

(3) 能够基于工程实际,将 Docker、CephFS 和 Hadoop 进行有效集成,以解决复杂软件工程问题。

2. 素质目标

(1) 能够准确撰写 Docker 实现运行空间隔离的解决方案,并对方案分析和评估性价比。

(2) 能够翔实撰写 Docker 环境搭建的文档,对所遇问题和解决措施予以记录和分析。

5.2 Docker 容器实践基础

5.2.1 安装 Docker

Docker 是一个开源的应用容器引擎,基于 Go 语言并遵从 Apache 2.0 开源协议。Docker 可以让开发者打包应用以及依赖包到一个轻量级、可移植的容器中,然后发布到任何流行的 Linux 机器上,也可以实现虚拟化。容器是完全使用沙箱机制,相互之间不会有任何接口,更重要的是容器性能开销极低。

Docker 使用客户端/服务器(C/S)架构模式,使用远程 API 来管理和创建 Docker 容器。Docker 容器通过 Docker 镜像来创建。容器与镜像的关系类似于面向对象编程中的对象与类。Docker 的官方安装说明网址如下。

```
https://docs.docker.com/install/linux/docker-ee/centos/#install-from-the-repository
```

1. 检查系统内核

Docker 要求 CentOS 系统的内核版本高于 3.20，查看本页面的前提条件来验证 CentOS 版本是否支持 Docker。查看当前内核版本，执行如下的命令。

```
[root@master 189]#uname -r
```

命令执行结果显示如下。

```
3.20.0-693.el7.x86_64
```

2. 安装 Docker

Docker 软件包和依赖包包含在默认的 CentOS-Extras 软件源里，安装命令如下。

```
[root@master 189]#yum -y install docker
```

3. 查看 Docker 版本

启动 Docker 版本，执行如下的命令。

```
[root@master 189]#docker version
```

命令执行结果显示如下。

```
Client:
Version:         1.13.2
API version:     1.26
Package version:
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the
docker daemon running?
```

4. 启动 Docker

启动 Docker，执行如下的命令。

```
[root@master 189]#systemctl start docker.service
```

5. 验证 Docker 启动是否成功

验证启动是否成功，执行如下的命令。

```
[root@master 189]#  docker version
```

输出信息中如果有 Client 和 Server 两部分，表示 Docker 安装与启动均成功。

6. 加入开机启动

加入开机启动，执行如下的命令。

```
[root@master 189]#sudo systemctl enable docker
```

5.2.2 Docker 基本操作

1. Docker 镜像基本操作

本次实验包含 Docker 基本操作，仅提供简单 Docker 操作命令演示，更多操作请访问官方文档。

(1) 手动启动 Docker 服务，执行如下的命令。

```
#service docker start
```

(2) 查看本机所有的镜像，执行如下的命令。

```
#docker images
```

命令执行结果显示如下。

```
REPOSITORY    TAG         IMAGE ID    CREATED    SIZE
```

实验环境中没有镜像，所以只输出头信息。

(3) 查找镜像。

通过 docker search 命令查找镜像，如查找 alpine 镜像，执行如下的命令。

```
#docker search alpine
```

(4) 拉取镜像。

查找到需要的镜像以后，可以通过 docker pull 命令拉取指定的镜像。以拉取 alpine 镜像为例，执行如下的命令。

```
#docker pull alpine
```

如果不指定镜像标签，则默认拉取最新版本的镜像。查看已拉取的镜像，执行如下的命令。

```
#docker images
```

命令执行结果显示如下。

```
REPOSITORY          TAG         IMAGE ID        CREATED       SIZE
docker.io/alpine    latest      389fef711851    3 weeks ago   5.58 MB
```

(5) 构建镜像。

当搜索不到需要的镜像时，可以使用 docker build 命令构建镜像，以 alpine:latest 为基础镜像，添加自定义脚本，并在容器启动时执行脚本，输出“hello docker my New Build”。

首先查看 test.sh 文件，该脚本仅输出“hello docker”到终端，不执行其他操作。

```
#vi test.sh
```

文件打开后，仅有一个如下的输出操作。

```
echo "hello docker my New Build"
```

查看并编辑 Dockerfile 文件，执行如下的命令。

```
#vim Dockerfile
```

文件内容如下。

```
FROM alpine:latest
ADD ./test.sh /test.sh
RUN chmod +x /test.sh
CMD ["/bin/sh", "-c", "/test.sh"]
```

构建镜像，执行如下的命令。

```
#docker build -t hello-docker.
```

运行构建的容器，执行如下的命令。

```
#docker run --name hello-docker hello-docker
```

命令执行结果显示如下。

```
hello docker my New Build
```

可以看到容器正确执行了自定义脚本。制作 Docker 镜像有如下两种方式。

第 1 种：docker commit，保存容器(Container)的当前状态到镜像后，然后生成对应的镜像。

第 2 种：docker build，使用 Dockerfile 文件自动化制作镜像。

A. docker commit

启动一个实例，执行如下的命令。

```
[root@master 189]#docker images
[root@master 189]#docker run -it centos:latest /bin/bash
```

安装 Apache，执行如下的命令。

```
[root@DMaster]#yum -y install httpd
[root@DMaster]#exit
```

查看容器的状态，执行如下的命令。

```
[root@master docker]#docker ps -a
```

命令执行结果显示如下。

```
CONTAINER ID  IMAGE          COMMAND               CREATED       STATUS
72ad7c24ba38  centos:latest  "/bin/bash"           3 hours ago   Exited (0) 48 seconds ago
21eac7115704  hello-docker   "/bin/sh -c /test.sh" 4 hours ago   Exited (0) 4 hours ago
```

```
e4e63beea0e6  hello-world   "/hello"              5 hours ago   Exited (0) 5 hours ago
```

根据容器当前状态制作一个镜像，语法：docker commit ＜容器 ID＞［仓库］:［标签］，其具体示例语句如下。

```
[root@master docker]#docker commit 72ad7c24ba38 centos:httpd
```

命令执行结果显示如下。

```
sha256:6ec494cd22012b53698d3733b6bce4aa824d714f0f8384d29310d21c23dc7e15
```

```
[root@master docker]#docker images
```

命令执行结果显示如下。

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	httpd	6ec494cd2201	About a minute ago	209 MB
hello-docker	latest	864d2be3c4db	4 hours ago	5.58 MB
docker.io/alpine	latest	389fef711851	3 weeks ago	5.58 MB
docker.io/centos	latest	300e315adb2f	5 weeks ago	209 MB
docker.io/hello-world	latest	bf756fb1ae65	12 months ago	13.4 kB

启动新创建的镜像，查看是否存在 httpd 服务，执行如下的命令。

```
[root@master docker]#docker run -it centos:httpd /bin/bash
[root@master docker /]#rpm -qa httpd
```

命令执行结果显示如下。

```
httpd-2.4.6-89.el7.centos.1.x86_64
```

如果看到上面的输出信息，则说明 httpd 存在。

B. docker build

使用 docker build 创建镜像时，需要使用 Dockerfile 文件自动化制作镜像。Dockerfile 类似源码编译./configure 后产生的 Makefile。首先创建工作目录，制作 Dockerfile，执行如下的命令。

```
[root@master docker]#mkdir /docker-build
[root@master docker]#vim /docker-build/Dockerfile
```

文件打开后，编辑文件内容如下。

```
FROM centos:latest                                    #以哪个镜像为基础
MAINTAINER <youxi@163.com>                            #镜像创建者
RUN yum -y install httpd                              #运行安装 httpd 命令
ADD start.sh /usr/local/bin/start.sh
                                  #将本地文件复制到镜像中，权限为 755，uid 和 gid 为 0
ADD index.html /var/www/html/index.html
```

```
CMD /usr/local/bin/start.sh
                    #实例启动后执行的命令,在 strat.sh 里添加需要开机启动的服务或脚本
```

创建 start.sh 和 index.html,执行如下的命令。

```
[root@master docker]# echo "/usr/sbin/httpd -DFOREGROUND" > /docker-build/
start.sh
[root@master docker]# chmod +x /docker-build/start.sh
[root@master docker]# echo "docker image build test" > /docker-build/
index.html
```

删除已有的 centos:httpd,先查看其容器 ID,然后根据容器 ID 予以删除,执行如下的命令。

```
[root@master docker]# docker ps -a
```

命令执行结果显示如下。

```
CONTAINER ID  IMAGE          COMMAND                  CREATED          STATUS
660fb6e03ba3  centos:httpd   "/bin/bash"              21 minutes ago   Exited (127) 6 seconds ago
72ad7c24ba38  centos:latest  "/bin/bash"              4 hours ago      Exited (0) 35 minutes ago
21eac7115704  hello-docker   "/bin/sh -c /test.sh"    5 hours ago      Exited (0) 5 hours ago
e4e63beea0e6  hello-world    "/hello"                 5 hours ago      Exited (0) 5 hours ago
```

根据容器 ID 对容器予以先停止后删除,执行如下的命令。

```
[root@master docker]# docker stop 660fb6e03ba3
[root@master docker]# docker rm 660fb6e03ba3
[root@master docker]# docker ps -a
```

命令执行结果显示如下。

```
CONTAINER ID  IMAGE          COMMAND                  CREATED          STATUS
72ad7c24ba38  centos:latest  "/bin/bash"              4 hours ago      Exited (0) 35 minutes ago
21eac7115704  hello-docker   "/bin/sh -c /test.sh"    5 hours ago      Exited (0) 5 hours ago
e4e63beea0e6  hello-world    "/hello"                 5 hours ago      Exited (0) 5 hours ago
```

```
[root@master docker]# docker rmi centos:httpd
```

命令执行结果显示如下。

```
Untagged: centos:httpd
Deleted:
sha256:6ec494cd22012b53698d3733b6bce4aa824d714f0f8384d29310d21c23dc7e15
Deleted:
sha256:2c9b167f19cee62b426761a4e48d02ec6d1b8edd8107eeccb5617c4cd9f94868
```

```
[root@master docker]# docker images
```

命令执行结果显示如下。

```
REPOSITORY                TAG       IMAGE ID        CREATED          SIZE
hello-docker              latest    864d2be3c4db    5 hours ago      5.58 MB
docker.io/alpine          latest    389fef711851    3 weeks ago      5.58 MB
docker.io/centos          latest    300e315adb2f    5 weeks ago      209 MB
docker.io/hello-world     latest    bf756fb1ae65    12 months ago    13.4 kB
```

使用 build 创建新的镜像，语法：docker build -t [仓库名]:[标签] [Dockerfile 文件路径]，其示例语句如下。

```
[root@master docker]#docker build -t centos:httpd /docker-build/
[root@master docker]#docker images
```

命令执行结果显示如下。

```
REPOSITORY                TAG       IMAGE ID        CREATED          SIZE
centos                    httpd     5fde376f6cb5    13 minutes ago   250 MB
hello-docker              latest    864d2be3c4db    5 hours ago      5.58 MB
docker.io/alpine          latest    389fef711851    3 weeks ago      5.58 MB
docker.io/centos          latest    300e315adb2f    5 weeks ago      209 MB
docker.io/hello-world     latest    bf756fb1ae65    12 months ago    13.4 kB
```

(6) Docker 镜像发布有发布到本地和发布到网上两种方式，具体如下。

① 发布到本地。

语法：docker save -o [tar 包名] [仓库名]:[标签]，示例命令如下。

```
[root@master docker]# docker save -o docker.id-centos-httpd-image.tar
centos:httpd
[root@master docker]#ll -h
```

② 发布到网上。

一般先到 https://hub.docker.com/(DockerHub)上注册一个账号，并创建一个存储库，然后使用如下的登录命令。

```
[root@master docker]#docker login -u [用户名] -p [密码]
```

上传镜像。注意：上传前需要修改仓库名，否则上传有问题。

```
[root@master docker]#docker tag centos:httpd 738441242/centos
```

738441242 是笔者的 Docker 用户名，centos 是建立的存储库。

```
[root@master docker]#docker push 738441242/centos
```

发布成功后，登录 Dock Hub 用户，查看镜像仓库，如图 5-1 所示。

(7) 删除镜像，使用 docker rmi <ID> 命令予以删除。如果镜像已经被容器运行，则先停止并删除对应的容器，而后删除镜像。

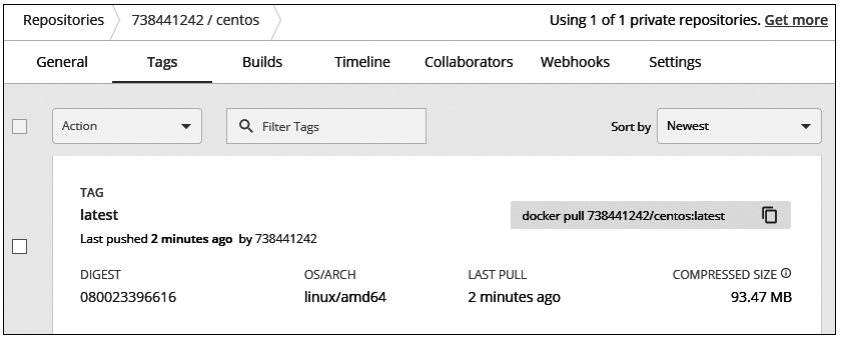

图 5-1　发布到远程 Dock Hub 用户下的仓库

2. Dockerfile 文件

(1) 什么是 Dockerfile。

Dockerfile 是组合镜像命令的文本文档,可以在命令行中调用任何命令。Docker 通过读取 Dockerfile 中的指令自动生成镜像。docker build 命令用于从 Dockerfile 构建镜像。可以在 docker build 命令中使用-f 标志指向文件系统中任何位置的 Dockerfile,如下所示。

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

(2) Dockerfile 的基本结构。

Dockerfile 一般分为基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令四部分,# 开头的内容为 Dockerfile 中的注释。

(3) Dockerfile 文件说明。

Docker 以从上到下的顺序运行 Dockerfile 的指令。为了指定基本镜像,第一条指令必须是 FROM。一个声明以 # 字符开头则被视为注释。可以在 Docker 文件中使用 RUN、CMD、FROM、EXPOSE、ENV 等指令。下面列出了一些常用的指令。

FROM: 指定基础镜像,必须为第一个命令,格式如下。

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

示例: FROM mysql: 5.6

tag 或 digest 是可选的,如果不使用这两个值时,会使用 latest 版本的基础镜像。

MAINTAINER: 维护者信息,格式如下。

```
MAINTAINER <name>
```

示例如下所示。

```
MAINTAINER Jasper Xu
```

```
MAINTAINER sorex@163.com
MAINTAINER Jasper Xu <sorex@163.com>
```

RUN：构建镜像时执行的命令，用于在镜像容器中执行命令，有以下两种命令执行方式。

shell 执行，格式：

```
RUN <command>
```

exec 执行，格式：

```
RUN ["executable", "param1", "param2"]
```

示例如下所示。

```
RUN ["executable", "param1", "param2"]
RUN apk update
RUN ["/etc/execfile", "arg1", "arg1"]
```

RUN 指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不想使用这些缓存镜像，可以在构建时指定--no-cache 参数，如：docker build --no-cache。

ADD：将本地文件添加到容器中，tar 类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似 wget，格式如下。

```
ADD <src>... <dest>
```

ADD ["<src>",... "<dest>"] 用于支持包含空格的路径。

示例如下所示。

```
ADD hom* /mydir/                 #添加所有以"hom"开头的文件
ADD hom?.txt /mydir/             #?替代一个单字符,例如: "home.txt"
ADD test relativeDir/            #添加 "test" 到 `WORKDIR`/relativeDir/
ADD test /absoluteDir/           #添加 "test" 到 /absoluteDir/
```

COPY：功能类似 ADD，但是不会自动解压文件，也不能访问网络资源。

CMD：构建容器后调用，也就是在容器启动时才进行调用，其格式如下。

```
CMD ["executable","param1","param2"] (执行可执行文件,优先)
CMD ["param1","param2"] (设置了 ENTRYPOINT,则直接调用 ENTRYPOINT 添加参数)
CMD command param1 param2 (执行 shell 内部命令)
```

示例如下所示。

```
CMD echo "This is a test." | wc -
CMD ["/usr/bin/wc","--help"]
```

CMD 不同于 RUN，CMD 用于指定在容器启动时所要执行的命令，而 RUN 用于指定镜像构建时所要执行的命令。

ENTRYPOINT：配置容器，使其可执行化。配合 CMD 可省去“application”，只使用参数，具体格式如下所示。

```
ENTRYPOINT ["executable", "param1", "param2"] (可执行文件, 优先)
ENTRYPOINT command param1 param2 (shell 内部命令)
```

示例如下所示。

```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD ["-c"]
```

ENTRYPOINT 与 CMD 非常类似，不同的是通过 docker run 执行的命令不会覆盖 ENTRYPOINT，而 docker run 命令中指定的任何参数，都会被当作参数再次传递给 ENTRYPOINT。Dockerfile 中只允许有一个 ENTRYPOINT 命令，多指定时会覆盖前面的设置，而只执行最后的 ENTRYPOINT 指令。

LABEL：用于为镜像添加元数据，其格式如下所示。

```
LABEL <key>=<value><key>=<value><key>=<value>...
```

示例如下所示。

```
LABEL version="1.0" description="这是一个 Web 服务器" by="IT 笔录"
```

使用 LABEL 指定元数据时，一条 LABEL 指定可以指定一或多条元数据，指定多条元数据时不同元数据之间通过空格分隔。推荐将所有的元数据通过一条 LABEL 指令指定，以免生成过多的中间镜像。

ENV：设置环境变量，其格式如下所示。

ENV ＜key＞ ＜value＞ ＃＜key＞之后的所有内容均会被视为其＜value＞的组成部分，因此，一次只能设置一个变量。

ENV ＜key＞＝＜value＞... ＃可以设置多个变量，每个变量为一个“＜key＞＝＜value＞”的键值对，如果＜key＞中包含空格，可以使用\来进行转义，也可以通过""来进行标示；另外，反斜线也可以用于续行。

示例如下所示。

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat=fluffy
```

EXPOSE：指定与外界交互的端口，格式：EXPOSE ＜port＞ [＜port＞...]，示例如下所示。

```
EXPOSE 80 443
EXPOSE 8080
EXPOSE 11211/tcp 11211/udp
```