Docker学习笔记

教程
视频

配置安装

本笔记安装在CentOS7.9版本下
CentOS7.9安装链接

如果按照上述教程走的话，会在配置docker yum源的地方报错

```
Could not retrieve mirrorlist http://mirrorlist.centos.org/?release=7&arch=x86_64&repo=os&infra=
```

首先检查网络连接是否正常，ping一下诸如 `www.baidu.com` 等域名
执行

```
vi /etc/sysconfig/network-scripts/ifcfg-ens33
```

将 `ONBOOT` 改为 `yes`

此时大概率还是不行，那么就备份一下当前yum源

```
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
```

下载新的CentOS-Base.repo到/etc/yum.repos.d/

```
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
```

下载完成以后，更新yum，下载过程中问你y/N，全部都是y

```
yum update
```

下载完成以后就可以正常配置docker了

```
sudo yum install -y yum-utils

sudo yum-config-manager \
--add-repo \
http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

下载最新版本docker

```
sudo yum install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-pl
```

到此，docker算是安装完毕了，使用以下命令开启docker，测试是否成功

```
sudo systemctl start docker

docker ps
```

配置开机自动启动docker

```
sudo systemctl enable --now docker
```

由于docker已经被墙，那么就要配置一些镜像节点加速，最好配置一些有效的源，不然会因为源不可用等问题出错

```
sudo tee /etc/docker/daemon.json <<-'EOF'
{
    "registry-mirrors": [
        "https://mirror.ccs.tencentyun.com",
        "https://docker.1panel.live/",
        "https://do.nark.eu.org",
        "https://dc.j8.work",
        "https://docker.m.daocloud.io",
        "https://dockerproxy.com",
        "https://docker.mirrors.ustc.edu.cn",
        "https://docker.nju.edu.cn"
    ]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

至此，安装过程完毕

命令

镜像操作

要求：启动一个nginx，将它的首页改为自己的页面，发布出去让所有人都能用
步骤：下载镜像->启动容器->修改页面->保存镜像->分享社区

视频中的

```
docker search nginx
```

命令因为国内所有镜像源都被干掉了，所以这一步不做也罢，直接执行下面命令

```
docker pull nginx
```

下载指定版本的镜像 `镜像名:版本` ，不带版本默认是最新latest

```
docker pull nginx:1.26.0
```

查看当前docker列表

```
docker images
```

删除指定版本的镜像

```
docker rmi (IMAGE ID)
```

容器操作

查看运行镜像常用参数命令

```
docker run --help
# -d: 后台启动，如果不加 终端必须一直开着不能关闭
# -p: 端口映射 外部端口:内部端口
# --name: 容器名

docker run -d --name 别名 nginx
```

如果是开启了Linux的防火墙，我们需要配置防火墙才能在外部访问到浏览器，否则只能在虚拟机内部访问
输入以下命令直接关闭防火墙(~~简单粗暴~~)

```
systemctl stop firewalld

# 开启防火墙命令
systemctl start firewalld
```

或者是手动配置防火墙

```
# 查看防火墙开放的服务和端口
firewall-cmd --list-all
# 此时的返回信息中一般会存在 services:dhcpv6-client ssh 说明没开放任何端口

# 开放http服务
firewall-cmd --add-service=http --permanent
# 开放8080端口
firewall-cmd --add-port=8080/tcp --permanent
# 开放80端口
firewall-cmd --add-port=80/tcp --permanent
# 重启防火墙
firewall-cmd --reload
```

常用的命令

```
# 查看运行中的容器
docker ps
# 查看所有容器
docker ps -a
# 运行一个新容器
docker run nginx
# 停止容器
docker stop 容器名或id
# 启动容器
docker start 容器名或id
# 重启容器
docker restart 容器名或id
# 查看容器资源占用情况
docker stats 容器名或id
# 查看容器日志
docker logs 容器名或id
# 删除指定容器
docker rm 容器名或id
# 强制删除指定容器
docker rm -f 容器名或id
# 一键删除所有运行的容器
docker rm -f $(docker ps -aq)
# 后台启动容器
docker run -d --name 容器别名 nginx
# 后台启动并暴露端口
docker run -d --name 容器别名 -p 80:80 nginx
# 进入容器内部
docker exec -it 容器别名 /bin/bash
```

docker容器内部也是一个虚拟、简易的Linux系统，其中只能进行简单的命令，容器内部的所有数据管理起来是非常困难的

分享操作

常用命令

```
# 提交容器变化打成一个新的镜像
docker commit -m "update index.html" mynginx mynginx:v1.0
# 保存镜像为指定文件
docker save -o mynginx.tar mynginx:v1.0
# 删除多个镜像
docker rmi 容器id1 容器id2 容器id3
# 加载镜像
docker load -i mynginx.tar
```

目前docker hub不支持个人镜像

存储

目录挂载

使用以下命令把docker内部文件夹挂载到外部文件夹

```
-v 外部文件夹路径:内部文件夹路径
```

```
-v /app/nghtml:/usr/share/nginx/html
```

这样就相当于内部的文件夹引用外部的文件夹，给内部文件夹增加了一个挂载目录。每次删除镜像时不必担心docker的数据丢失，因为在外部也保存了一份
如果进入到容器内部进行修改，需要使用追加(>>)双箭头
追加信息用双箭头，避免数据丢失用目录挂载

```
# 进入容器内部
docker exec -it 容器ID bash

cd /usr/share/nginx/html/

echo hello >> index.html

exit
```

卷映射

在目录挂载中，我们挂载的 `/usr/share/nginx/html` 并不是nginx的配置文件，也就是说在目录挂载中，我们初始的 `/app/nghtml` 默认是空的，但这并不影响nginx运行，但如果我们想要修改的

是 /etc/nginx/nginx.conf 文件的话，就不能使用目录挂载，因为在外部我们没有该配置文件的内容，如果使用目录挂载就会导致nginx启动不起来，因此我们要使用卷映射

卷映射格式

```
-v 外部卷名:内部文件夹路径
```

这样我们就可以把容器内部的文件夹相当于拷贝到了我们外部定义的卷中，在docker中定义的卷通常都放在 /var/lib/docker/volumes/卷名 中，在外部的卷中改文件内容，会相应的映射到容器内部的文件中

```
# 查看docker当中有哪些卷
docker volume ls

# 在docker中创建一个卷
docker volume create 卷名

# 查看卷的详情
docker volume inspect 卷名
```

删除运行中的容器，并不会影响外部挂载的目录或者卷的内容，因此以前容器的所有内容都能够被外部保留下来，不会数据丢失

网络

docker为每个容器分配一个唯一ip，使用 容器ip+容器端口 可以互相访问
如果在本机ip内的两个应用想相互通信，一种最直接能想到的办法就是app1通过自己内部端口所映射的外部端口，向本机ip地址下的app2的外部端口发送消息，app2的外部端口接到消息传递给其映射的内部端口，这样app2便可以和app1进行通信。但这样就有点小题大做，相当于在公司里我想和坐在我旁边的同事聊天，那么我必须先出公司大门，再进入公司大门找到同事再进行聊天。
每一个应用在docker启动的时候，都会加入一个docker0的默认网络，所有在docker启动的应用都可以通过docker0的ip地址(172.17.0.1/16)来进行通信
使用以下命令即可查看应用的细节

```
docker container insepect 容器名
```

在应用内部可以直接使用细节当中查到的IP地址进行通信，它们在内部都是通过docker0来进行数据转发
在容器内部使用以下命令来访问其他容器内容

```
curl http://其他容器内部ip地址:内部端口
```

ip地址可能会由于各种原因进行改变，但此时可以通过域名来访问，但docker0默认不支持主机域名，那么就可以创建自定义网络，容器的名字就是稳定的域名
查看docker网络能做什么用以下命令

```
docker network --help
```

在启动docker镜像的时候使用以下命令加入自定义网络

```
--network 自定义的网络名
```

在加入自定义网络以后，可以直接通过容器名来访问它，而不需要输入ip地址

```
curl http://容器名:容器内部开放端口
```

Redis主从集群同步
主机进行写，从机进行读，这样可以减弱对主机的访问压力
Bitnami-Redis文档
主机配置

```
-e REDIS_REPLICATION_MODE=master
-e REDIS_PASSWORD=123456
```

从机配置

```
-e REDIS_REPLICATION_MODE=slave
-e REDIS_MASTER_HOST=redis01
-e REDIS_MASTER_PORT_NUMBER=6379
-e REDIS_MASTER_PASSWORD=123456
-e REDIS_PASSWORD=123456
```

实践命令

启动容器的时候外部与内部端口的映射
目录或卷是否需要挂载或映射
是否需要配置环境变量，参考docker hub官方文档

数据库不建议放到容器内运行，因为读写数据多了一层docker的io性能消耗

Docker Compose

在使用docker的时候，有时每次启动需要配置很多命令，我们可以直接新建一个 `compose.yaml` 文件，在文件中写上每次启动所需要的命令和配置，这样就可以简化启动过程
compose-file文档

`compose.yaml` 文件配置

```
name: myblog
services:
  mysql:
    container_name: mysql
    image: mysql:8.0
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=123456
      - MYSQL_DATABASE=wordpress
    volumes:
      - mysql-data:/var/lib/mysql
      - /app/myconf:/etc/mysql/conf.d
    restart: always
    networks:
      - blog

  wordpress:
    image: wordpress
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_USER: root
      WORDPRESS_DB_PASSWORD: 123456
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress:/var/www/html
    restart: always
    networks:
      - blog
    depends_on:
      - mysql

volumes:
  mysql-data:
  wordpress:

networks:
  blog:
```

启动时直接输入以下命令

```
docker compose -f compose.yaml up -d
# docker compose -f 文件名 up -d
```

如果yaml文件被修改，docker compose只会动修改了的地方，其他地方不会动

docker compose下线命令，下线命令不会移除卷，也就是数据不会丢失

```
docker compose -f compose.yaml down

# 查看下线说明
docker compose down --help

# 下线时顺便删除所有相应数据卷
docker compose -f compose.yaml dowm --rmi all -v
```

Dockerfile

dockerfile说明文档

dockerfile就是将容器进行打包，得到一个jar包，以便于可以直接将jar包复制到某个机器中可以直接运行相同容器
常用命令

常见指令	作用
FROM	指定镜像基础环境
RUN	运行自定义命令
CMD	容器启动命令或参数
LABEL	自定义标签
EXPOSE	指定暴露端口
ENV	环境变量
ADD	添加文件到镜像
COPY	复制文件到镜像
ENTRYPOINT	容器固定启动命令

常见指令	作用
VOLUME	数据卷
USER	指定用户和用户组
WORKDIR	指定默认工作目录
ARG	指定构建参数

启动jar包需要使用java环境

`Dockerfile` 配置

```
FROM openjdk:17

LABEL author=leifengyang

COPY app.jar /app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app.jar"]
```

启动 `app.jar` 命令

```
# docker build -f 构建镜像的文件名 -t 指定镜像的名称 .指的是构建的整个上下文目录就是当前目录 相当于.
docker build -f Dockerfile -t myjavaapp:v1.0 .
```