1. linux 内核版本3.10以上才很好的支持docker ，出现很少的bug

uname -r

1. 卸载旧版本

yum remove docker docker-common docker-selinux docker-engine

yum remove docker-ce

卸载后删除已经下载好的镜像

rm -rf /var/lib/docker

1. 在线安装最新版本(安装docker-ce 社区版本)
   1. 安装依赖包

yum install -y yum-utils device-mapper-persistent-data lvm2

#安装前可查看

device-mapper-persistent-data和lvm2是否已经安装rpm -qa|grep device-mapper-persistent-datarpm -qa|grep lvm2

* 1. 设置 yum 源

yum-config-manager--add-repo https://download.docker.com/linux/centos/docker-ce.rep

* 1. 安装 最新版 docker-ce -y

yum install docker-ce -y 注意：docker-ce 默认安装最新版本

#安装指定版本docker-ce可使用以下命令查看

yum list docker-ce.x86\_64 --showduplicates | sort -r

# 安装完成之后可以使用命令查看

docker version

1. Docker的安装镜像的默认地址是国外的 ，在下载镜像时速度比较慢，因此可以替 换 国 内的镜像网站 ： <https://mirrors.tuna.tsinghua.edu.cn/docker-ce/>

Docker镜像下载配置地址 ： /etc/yum.respos.d/ddocker-ce.repo

1. 配置连接镜像服务器 镜像加速

各种应用容器镜像默认是用官网的地址下载的也很慢 ， 可以使用阿里云的地址

1. .注册登录开通阿里云[容器镜像服务](https://cr.console.aliyun.com/cn-hangzhou/repositories" \t "_blank)
2. .查看控制台，招到镜像加速器并复制自己的加速器地址
3. 找到/etc/docker目录下的daemon.json文件，没有则直接vi daemon.json
4. 填写 加速器地址

#填写自己的加速器地址

{

"registry-mirrors": ["https://zfzbet67.mirror.aliyuncs.com"]

}

1. 重新刷新设置

systemctl daemon-reload

1. 重启docker
2. systemctl restart docker
3. 设置 开机启动

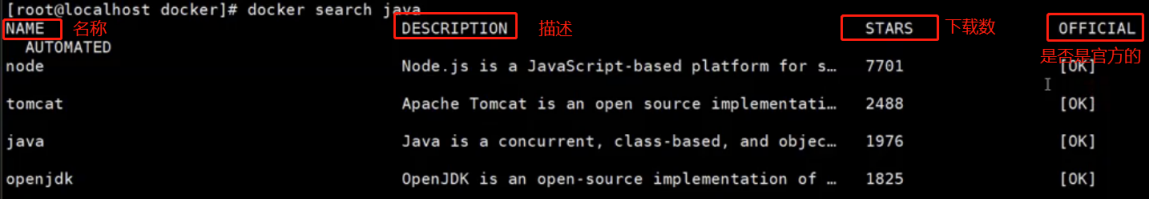
systemctl enable docker

1. Docker 的常用操作 (输入docker 可查看 docker命令的用法)
   1. 查看docker 服务整体运行情况

docker info

* 1. 查找镜像(在线搜索 相关的镜像)

docker search 关键词



* 1. 下载镜像

docker pull 镜像名:TAG

# Tag表示版本，有些镜像的版本显示latest，为最新版本

* 1. 查看镜像列表

docker images ls

# 查看本地所有镜像

* 1. 查看镜像详细信息系

docker image inspect 镜像名:TAG

* 1. 删除镜像

docker rmi -f 镜像ID或者镜像名:TAG

# 删除指定本地镜像# -f 表示强制删除

* 1. 获取元信息

docker inspect 镜像ID或者镜像名:TAG

# 获取镜像的元信息，详细信息

1. 容器相关的操作命令
   1. 查看容器

docker ps -a -q

# docker ps查看正在运行的容器

# -a 查看所有容器（运行中、未运行）# -q 只查看容器的ID

* 1. 运行容器

docker run --name 容器名 -i -t -p 主机端口:容器端口 -d -v 主机目录:容器目录:ro 镜像ID或镜像名:TAG

# --name 指定容器名，可自定义，不指定自动命名

# -i 以交互模式运行容器

# -t 分配一个伪终端，即输入命令行的窗口，通常-it组合来使用

# -p 指定映射端口，将主机端口映射到容器内的端口 80:8080

# -d 后台运行容器 类似 nohup

# -v 指定挂载主机目录到容器目录，默认为rw读写模式，ro表示只读

注意： 在每次运行容器时 最好起一个名字，以便操作方便 。如果不取docker会默认取一个名字

* 1. 停止容器

docker stop 容器ID或容器名

* 1. 删除容器

docker rm -f 容器ID或容器名

# -f 表示强制删除

* 1. 查看日志(docker 会将所有容器的日志 做统一管理)

docker logs 容器ID或容器名

* 1. 进入正在运行容器

docker exec -it 容器ID或者容器名 /bin/bash

# 进入正在运行的容器并且开启交互模式终端

# /bin/bash是固有写法，作用是因为docker后台必须运行一个进程，使用shell与当前容器进行交互

# 也可以用docker exec在运行中的容器执行命令

* 1. 拷贝文件 scp

docker cp 主机文件路径 容器ID或容器名:容器路径 #主机中文件拷贝到容器中

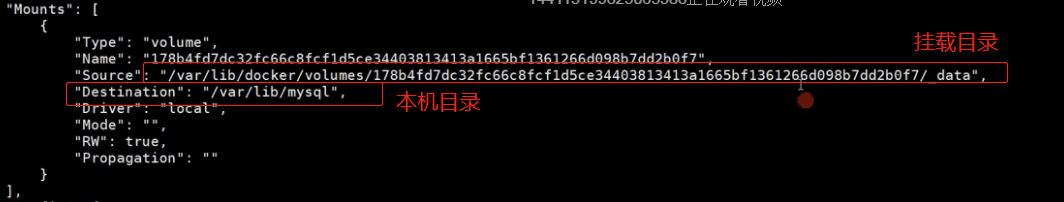
docker cp 容器ID或容器名:容器路径 主机文件路径 #容器中文件拷贝到主机中

1. 挂载

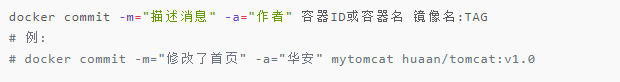
运行容器后 ，有时候需要修改这个容器的配置文件 ，而进到容器内不好操作 一般都 会把文件挂载到外面

1. 启动容器时 通过Docker run -v 参数挂载
2. 通过镜像自己的挂载设置

查看镜像元信息 docker inspect ，然后设置在Mounts



1. 制作镜像
   1. 方式1:基于已经做好的基础镜像 ，将其启动然后更改配置文件实现自己想要的功能 ， 然后 再把这个容器提交 制作成新的镜像
      * 1. 更改配置
        2. 提交新的镜像



* + - 1. 运行 新的镜像



* 1. 方式2： docker file

1. 什么是docker file ：构建 镜像的通用源码指令
   1. Docker file 格式 : 带#号的注释部分、不带#号的命令部分
2. 命令格式 大写
3. 第一行必须是 FROM 开头 ，后面跟**基础镜像，** 表示对底层的镜像

如 ： FROM java:8表示当前制作的容器运行环境依赖的基础镜像，也是最 终生成的那个镜像。这个镜像已经配置好了环境变量 ，后面增加 jar包 ， 后面的所有操作都是放到这个镜像里 如： 添加jar 包 ，配置端口等

1. 加入自己的镜像 如 制作springBoot镜像

ADD 宿主机文件路径 镜像中存放目录

如 ： ADD xxx.jar xxx.jar (可以add 多个文件)

**为什么这里没有使用路径 ？**

在docker build 时会指定 . 表示将当前目录指定为工作目录

也就是在 docker bulid . 时 ==> ADD /xxx/xxx/xxx.jar xxx.jar

1. 暴露镜像容器端口 (镜像的网络是封闭的需要暴露端口，不然无法访问)

如：EXPOSE 80 还可以指定端口传输协议 默认TCP

1. 制作容器启动成功后 ， 紧接着自动运行的代码

ENTRYPOINT [“jar ,”-jar”,”xxx.jar””]

1. 使用 docker bulid 命名构建docker file

构建时 docker bulid

注意事项

A、默认在当前目录下 ，找Dockerfile 文件 ，如果制作的dockerfile 不 是这么命名的 ，必须指定 -f 参数 告诉docker build命令

* 1. 如何制作

1. 创建一个新的 目录 ,制作docker file时，会将当前目录下的所有问价扫描一遍
2. 写docerfile文件
3. 必须FROM开头

**13、Docker build构建镜像**

* 1. 如果写的dockerfile文件的命名不是 dockerfile 那么在构建时需要指定全路径
  2. docker build -t 指定镜像名：tag .

. 表示表示当前dockerfile文件所在的目录

docker build -t huaan/mypro:v1 .

# -f指定Dockerfile文件的路径

# -t指定镜像名字和TAG

# .指当前目录，这里实际上需要一个上下文路径

**14、docker run运行镜像**

docker run --name pro -p 80:80 -d 镜像名:TAG

如果镜像中的时间不对，也可以设置时间



**15、dockerfile 常用指令**

* 1. FORM (from)
     + 1. 起始命令 ，表示制作dockerfile依赖的基础镜像且只能指定一个
       2. 语法 ：

FROM <image>

FROM <image>:<tag>

FROM <image>@<digest>

* 1. MAINTAINER (maintainer) 将要过期被label替代
     + 1. 表示制作dockerfile的作者是谁
       2. 语法：

MAINTAINER <name>

* 1. LABEL (label)
     + 1. 给镜像指定元数据，key value键值对
       2. 语法：

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

如指定作者 ： LACEL name=myd123 [mail=863836234@qq.com](mailto:mail=863836234@qq.com)

* 1. COPY (copy)
     + 1. 用于从宿主机复制文件到镜像中，如果是压缩包不解压
       2. 语法：

COPY <src>...<dest>

COPY ["<src>",..."<dest>"]

# <src>：要复制的源文件或者目录 必须是dockerfile的上下文目录，可以使用通配符

# <dest>：目标路径，即正在创建的image的文件系统路径；建议<dest>使用绝对路径，否则COPY指令则以WORKDIR为其起始路径

* 1. ADD (add)
     + 1. 添加文件到制作的镜像中 同copy功能一样，但是如果是压缩包并解压。并且add 支持url ，指定文件url后，可以自己下载文件完成添加。如果url文件是压缩不会解压
       2. 语法

ADD <src>...<dest>

ADD ["<src>",..."<dest>"]

* 1. WORKDIR (workdir)
     + 1. 用于在镜像中创建一个工作目录 ，即镜像运行后 当前目录是在哪个目录下
       2. 语法：

WORKDIR <dirpath>

* 1. VOLUME (volume)
     + 1. 用于挂载目录
       2. 语法

VOLUME <mountpoint>

VOLUME ["<mountpoint>"]

不能指定宿主机当中的目录，宿主机挂载的目录是自动生成的

* 1. EXPOSE (expose)
     + 1. 暴露端口 这个端口必须和tomca端口一致
       2. 语法

EXPOSE <port>[/<protocol>] [<port>[/<protocol>]...]

<protocol>用于指定传输层协议，可以是TCP或者UDP，默认是TCP协议

EXPOSE可以一次性指定多个端口，例如：EXPOSE 80/tcp 80/udp

* 1. ENV (env) 环境变量的意思
     + 1. 定义环境变量，并且用于被其他命令 ENV、ADD、COPY等调用（此处的环境变量就是 定义全局的变量 如shell脚本中的A=1。 且下面的命令可引用这个变量 ENV A ，类似 shell中的 ${A} 或者定义JAVA\_HOME 环境变量）
       2. 语法

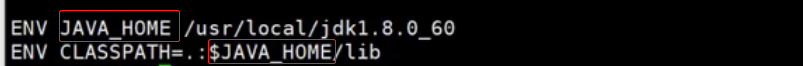
ENV <key> <value>

ENV <key>=<value>...

第一种格式: <key>之后的所有内容都会被视为<value>的组成部分，所以一次只能设置一个变量

第二种格式: 可以一次设置多个变量，如果<value>当中有空格可以使用\进行转义或者对 <value>加引号进行标识；另外\也可以用来续行

如定义jdk环境变量：



* 1. ARG (arg)
     + 1. 指定参数 ，在duild镜像时可以实现动态传参 类似shell 中的 A=$1
       2. 语法 ：

ARG <name>[=<default value>]

指定一个变量，可以在docker build创建镜像的时候，使用--build-arg <varname>=<value>来指定参数

* 1. RUN (run)
     + 1. 用于指定在运行docker build命令中 ，运行run指定的命令
       2. 语法

RUN <command>

RUN ["<executable>","<param1>","<param2>"]

第一种格式 ：一般是一个shell命令，以/bin/sh -c来运行它

第二种格式：是一个JSON格式的数组，当中<executable>是要运行的命令，后面是传 递给命令的选项或者参数；但是这种格式不会用/bin/sh -c来发起，所以 常见的shell操作像变量替换和通配符替换不会进行；如果你运行的命令 依赖shell特性，可以替换成类型以下的格式

RUN ["/bin/bash","-c","<executable>","<param1>"]

使用场景 ： 容器启动后给容器装一些环境、工具 yum install xxx等

* 1. CMD (cmd)
     + 1. 容器启动后，第一条紧接着运行的命令 ,如果设置多条会被覆盖那么最后一条会生效
       2. 语法

CMD <command>

CMD ["<executable>","<param1>","<param2>"]

CMD ["<param1>","<param2>"]

* 1. ENTRYPOINT (entrypoint)
     + 1. 类似 cmd指令，用于指定运行镜像后第一条运行的程序 ,如果entrypoint有多个也只最后一条生效 。 如果在dockerfile中cmd 和 entrypoint都指定了 ，那么谁在最后谁起作用

ENTRYPOINT<command>

ENTRYPOINT["<executable>","<param1>","<param2>"]

* 1. ONBUILD (onbuild)
     + 1. 用于定义一个触发器
       2. ONBUILD <instruction>

**16、run 和 cmd的区别**

(1)RUN指令运行于镜像文件构建过程中，CMD则运行于基于Dockerfile构建出的新镜像文件 启动为一个容器的时候

(2)CMD指令的主要目的在于给启动的容器指定默认要运行的程序，且在运行结束后，容器也 将终止；不过，CMD命令可以被docker run的命令行选项给覆盖

(3)Dockerfile中可以存在多个CMD指令，但是只有最后一个会生效

**17、cmd 和 entrypoint的区别**

entrypoint和CMD不同的是ENTRYPOINT启动的程序不会被docker run命令指定的参数所 覆盖，而且，这些命令行参数会被当做参数传递给ENTRYPOINT指定的程序(但是，docker run 命令的--entrypoint参数可以覆盖ENTRYPOINT)

docker run命令传入的参数会覆盖CMD指令的内容并且附加到ENTRYPOINT命令最后作为 其参数使用

同样，Dockerfile中可以存在多个ENTRYPOINT指令，但是只有最后一个会生效

Dockerfile中如果既有CMD又有ENTRYPOINT，并且CMD是一个完整可执行命令，那么谁 在最后谁生效

**Docker网络**

<https://www.yuque.com/books/share/9f4576fb-9aa9-4965-abf3-b3a36433faa6/eqaeh0>

18、Docker可以设置外部网络访问docker容器

19、Docker容器有3中网络连接模式 bridge(默认)、host、none、containr

containr 是共享容器莫斯

**Docker compose**

20、Docker compose

在多个环境需要相互配合时使用，compose可以定义运行多个容器应用，只需要配置 yml配置文件，然后启动一条命令将就可以将多个容器启动

21、compose 是python写的需要安装python依赖