容器技术

docker与Kubernetes

Docker 是一个基于Go语言开源的应用容器引擎。与笨重的虚拟化主机不同，它使用进程虚拟化的方式来使程序不受系统环境影响的情况下独立运行。

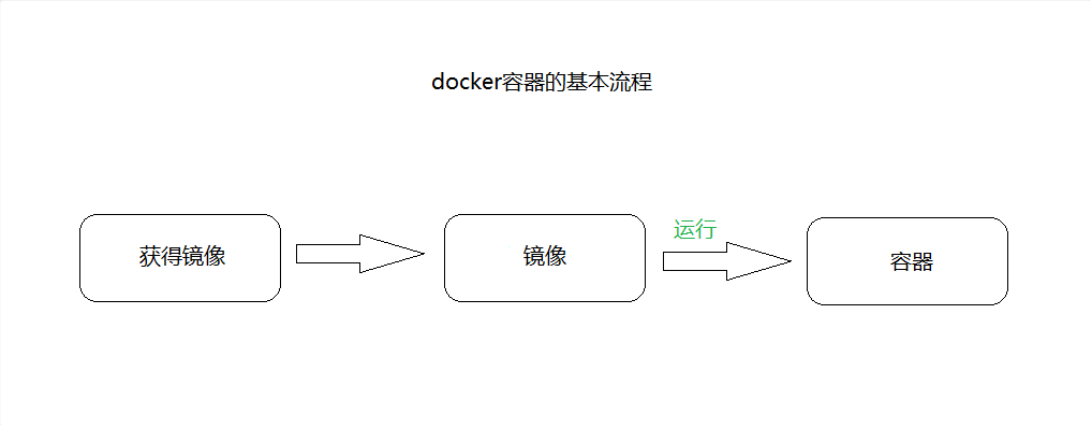
docker常用网址：

- 官网地址：https://www.docker.com
- Github源码：https://github.com/docker/docker-ce
- Docker Hub：https://hub.docker.com

docker的使用

使用docker之前，需要在你的主机上安装docker客户端，安装方式在文章尾部，docker安装之后就可以使用自己的虚拟进程了。

docker是进程虚拟化，即我们需要一个程序来变成进程，这个程序被称为镜像，镜像在启动之后被称作容器。docker命令需要使用管理员权限运行。



docker基本命令：

```Bash
# 查看所有docker命令
sudo docker
```

镜像获取

我们可以通过下载已有镜像或者自己构建镜像2种方式获得docker镜像。

镜像仓库

docker客户端默认使用镜像仓库服务docker hub，它同时有多个镜像仓库，其中包含官方与第三方的仓库，每个仓库有若干镜像。

上传镜像到远程

dockerHub 提供远程镜像仓库，我们可以自由下载镜像，如果我们需要将自己的镜像上传到远程仓库，那么我们需要先使用 Docker ID 登录 Docker Hub，然后为待推送的镜像打上合适的标签。

```Bash
# 登录dockerhub
sudo docker login
# 推送镜像，默认为docker.io/仓库名:latest
sudo docker image push
# 但你的账户没有权限访问上面的仓库，只能尝试推送到二级命名空间（Namespace）之下
# 使用你的用户名为镜像打标签
sudo docker image tag web:latest 用户名/仓库名:latest
```

没有为 Registry 和 Tag 指定值的时候，Docker 会默认 Registry=docker.io、Tag=latest。Docker 并没有给 Repository 提供默认值，而是从被推送镜像中的 REPOSITORY 属性值获取。

通过镜像仓库下载镜像

```Bash
# 搜索镜像
docker search 镜像

# 从仓库拉取镜像
sudo docker pull 镜像
```

linux 中会拉取 ubuntu:latest 镜像；windows 中会拉取 microsoft/powershell:nanoserver 镜像。

手动构建镜像

```Bash
# 新建一个镜像（按Dockerfile内容顺序）
sudo docker image build [参数] 镜像
```

构建参数：

- -t：为镜像打标签
- -f：指定 Dockerfile 的路径和名称
- --nocache=true：强制忽略缓存机制
- --squash：合并镜像镜像层

可以通过多个镜像来构建一个新镜像。

镜像管理

下面几种方式可以指定一个镜像：

```Bash
1.镜像名称                          # ubuntu
2.镜像ID                           # 02674b9cb179
3.镜像名称:版本标签                  # ubuntu:latest
4.dockerHub账户名/镜像名称:版本标签    # microsoft/ubuntu:latest
```

常用操作：

```bash
# 查看镜像列表
sudo docker image ls


# 启动镜像，latest为镜像标签，表示最新版本（无标签镜像为虚悬镜像）
sudo docker container run 启动参数 镜像 /bin/bash［运行的程序］
# 启动参数：
# [-i]：交互模式启动
# [-t]：启动后进入终端
# [-d]：后台启动
# [-v 主机目录:容器目录]：挂载主机的目录为容器的目录
# [-p 主机端口:容器端口]：映射主机和容器端口，可以[-p 主机端口:容器端口/udp]来指定udp协议
# [--name 名称]：为容器命名
# [--restart=策略]：no容器退出时不重启、on-failure容器非正常退出时重启、
# on-failure:3容器非正常退出时重启最多3次、always容器退出时重启、
# unless-stopped容器退出时重启，不考虑在Docker守护进程启动时就已经停止了的容器。
######## 启动实例：运行时启动容器内的shell脚本
sudo docker container run -itd --name wenmc -p 8080:9999 ubuntu /bin/bash /home/test.sh


# 删除镜像
sudo docker image rm 镜像
```

其他操作：

```bash
# 删除所有虚悬镜像
sudo docker image prune

# 查看镜像构建历史
sudo docker history

# 查看镜像分层
sudo docker image inspect 镜像
```

容器管理

下面几种方式可以指定一个容器：

```bash
1.容器ID
2.容器名称
```

容器基本信息：

container id	image	command	created	status	ports	names
容器ID	镜像名称	已运行的命令	创建时间	状态	端口信息|连接类型	名称

容器状态有：created(已创建)、restarting(重启中)、running(运行中)、removing(迁移中)、paused(暂停)、exited(停止)、dead(死亡)

常用操作：

```bash
# 查看所有容器
sudo docker ps -a

# 查看运行中的容器列表
sudo docker ps
sudo docker container ls

# 进入容器
sudo docker attach 容器           # 退出后容器会停止
sudo docker exec -it 容器 bash    # 退出后容器会继续运行

# 停止容器
sudo docker stop
sudo docker stop 容器

# 查看容器端口
sudo docker port 容器 端口
```

```
18
19  # 上传文件到容器
20  sudo docker cp 本地文件路径 ID全称:容器路径
21
```

其他操作：

```Bash
1   # 显示容器的配置细节和运行时信息
2   sudo docker container inspect 容器
3
4   # 查看容器日志
5   sudo docker logs 容器
6
7   # 启动停止的容器
8   sudo docker start 容器
9
10  # 重启容器
11  sudo docker restart 容器
12
13  # 删除容器
14  sudo docker rm -f 容器
15
16  # 清除全部未运行的容器
17  sudo docker container prune
18
19  # 退出容器
20  exit 或 CTRL+D
```

容器转镜像

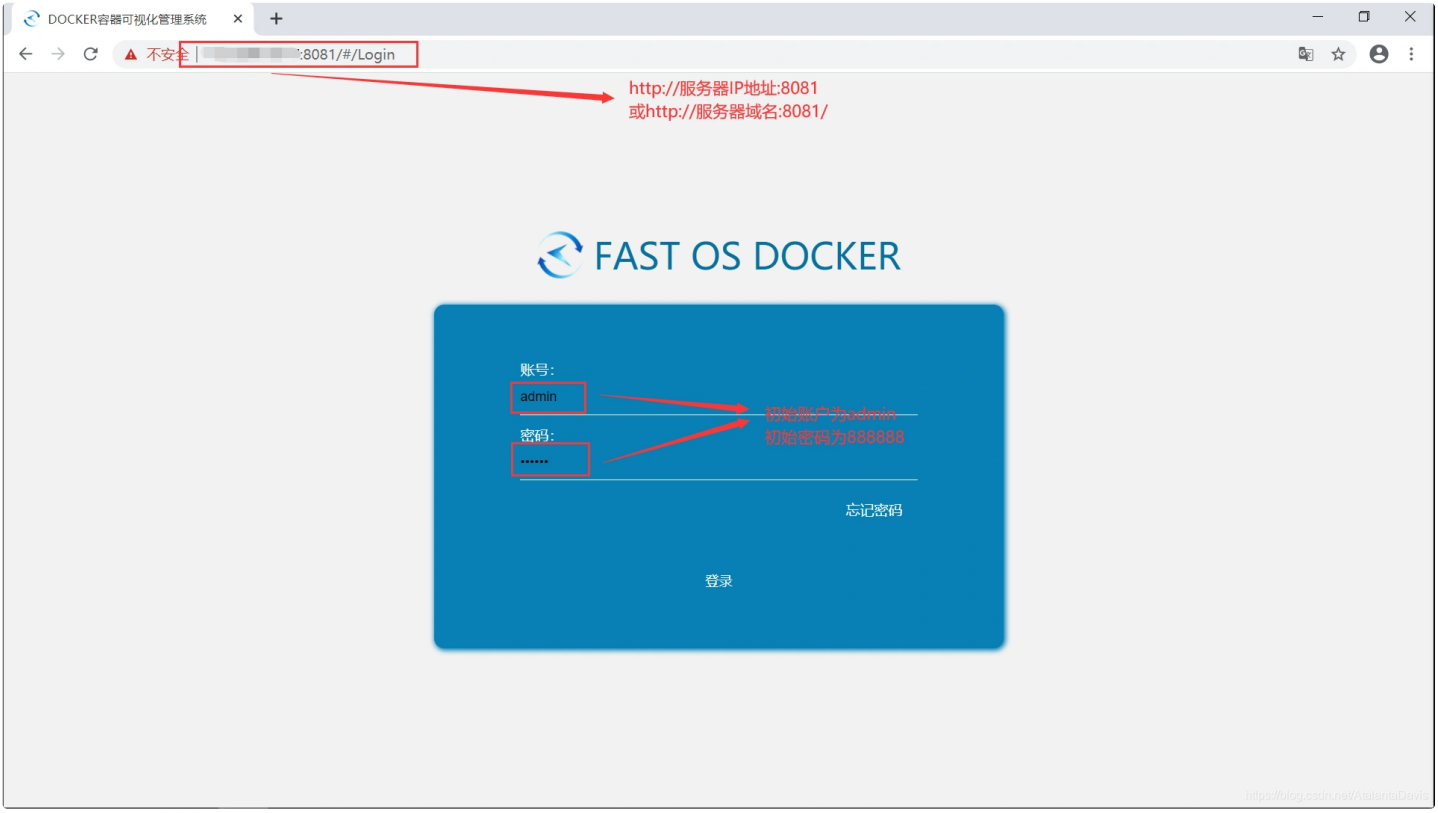
```Bash
1   # 容器转快照
2   sudo docker export -o 快照名.tar 容器
3
4   # 快照转镜像
5   sudo cat 快照名.tar | docker import - 镜像
6
7   # 远程快照转镜像
8   sudo docker import 快照URL example/imagerepo
9
10  # 查看容器信息
11  docker ps -a --no-trunc
```

docker 面板

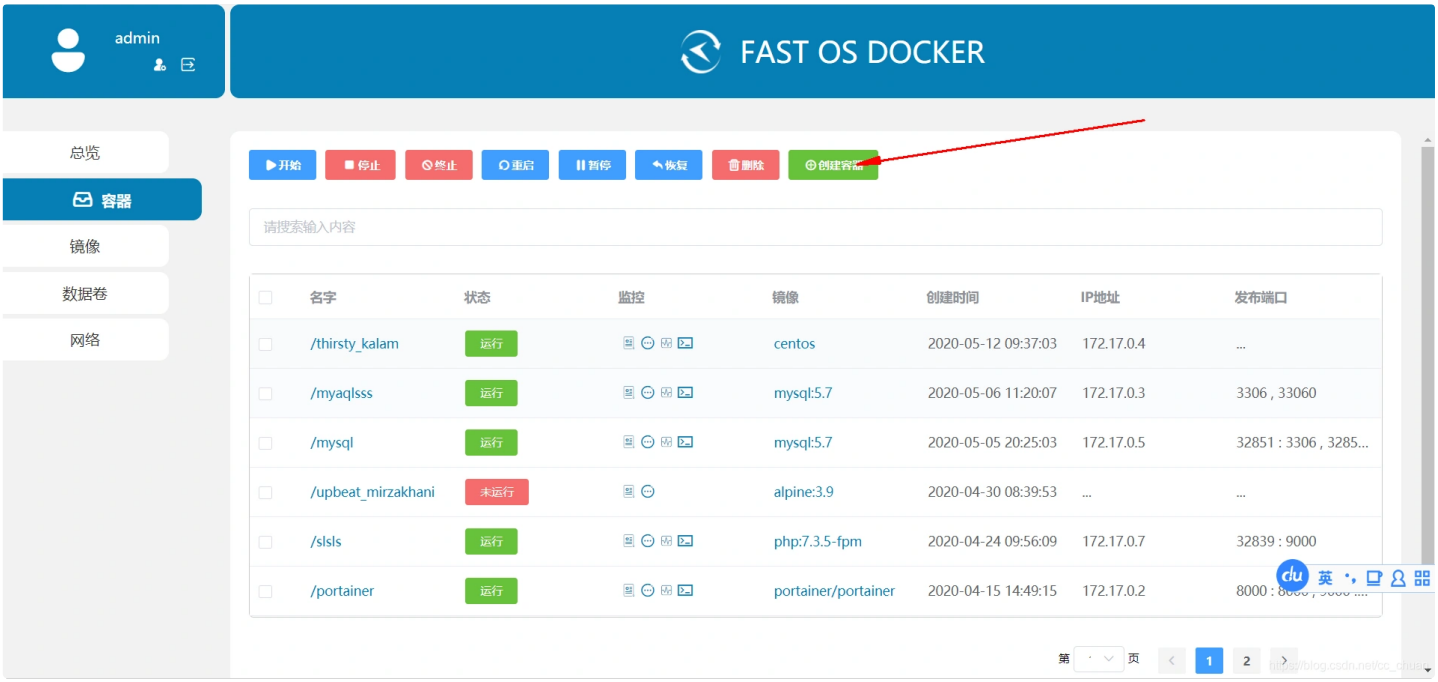
面板就是一个可视化管理界面，可以省去繁琐的指令操作，Linux 上有个比较好的面板 `fast os docker`

```Bash
sudo docker run --restart always --name fast -p 8081:8081 -d -v /var/run/docker.sock:/var/run/docker.sock wangbinxingkong/fast
```

电脑访问：http://服务器IP地址或域名:8081/



可以获取镜像、运行容器等操作。



Dockerfile 介绍

Dockerfile 是一个用来构建镜像的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

缓存命中

Dokckerfile 从上往下执行，第一次较慢，第二次使用缓存内容较快，如果某一条无法从缓存中找到对应资源，后续指令将不再使用缓存命中。

指令集

dockerfile 是由众多指令组成的，指令字符不区分大小写。使用指令既可以向镜像中增添新的文件或者程序来构建新建镜像层，还能告诉 docker 如何完成构建及运行应用程序，此时只会增加镜像的元数据。

镜像应尽量使用更小的体积，run 指令会新增一个镜像层从而导致体积变大，因此用 && 连接多个命令与反斜杠\换行来将多个命令包含在一个 RUN 指令中比较合适。

指令	用途	其它
#	注释行	
FROM	指定要构建的镜像的基础镜像	新增镜像层
RUN	在镜像中执行命令	新增镜像层
COPY	将文件复制到镜像中	新增镜像层
ADD	COPY 的扩展，支持 URL	
EXPOSE	指定应用使用的网络端口	新增元数据
WORKDIR	指定工作目录（或者称为当前目录）目录不存在时建立	新增元数据
USER	指定当前用户	
ENV	定义镜像的环境变量	新增元数据
ARG	定义镜像的环境变量（容器运行时无法使用）	
ENTERPOINT	指定镜像以容器方式启动后默认运行的程序	新增元数据
CMD	指定镜像以容器方式启动后默认运行的程序	
LABEL	设置一个 key=value 信息	
ONBUILD	为镜像添加触发器。其参数是任意一个 Dockerfile 指令	
HEALTHCHECK	健康检查	
VOLUME	定义匿名卷	

Dockerfile 主要包括两个用途：

- 对当前应用的描述。
- 指导 Docker 完成应用的容器化（创建一个包含当前应用的镜像）。

Docker

```
#第一行、添加当前镜像的基础镜像层
FROM alpine
#添加标签，即key=value
LABEL myemail="2476545423@qq.com"
#使用 alpine 的 apk 包管理器将 nodejs 和 nodejs-npm 安装到当前镜像之中
RUN apk add --update nodejs nodejs-npm
#指令将应用相关文件从构建上下文复制到了当前镜像中，并且新建一个镜像层来存储
COPY . /src
#为 Dockerfile 中尚未执行的指令设置工作目录
WORKDIR /src
#RUN npm install 指令会根据 package.json 中的配置信息，使用 npm 来安装当前应用的相关依赖包
#npm 命令会在前文设置的工作目录中执行，并且在镜像中新建镜像层来保存相应的依赖文件。
RUN npm install
#EXPOSE 8080 指令来完成相应端口的设置
EXPOSE 8080
#指定当前镜像的入口程序。
ENTRYPOINT ["node", "./app.js"]
```

多阶段构建：减少镜像中的冗杂。

run 命令太多导致镜像层多体积大、而且会拉取很多构建工具，导致镜像比较辣。而建造者模式又难。

多阶段构建方式使用一个 Dockerfile，其中包含多个 FROM 指令。每一个 FROM 指令都是一个新的构建阶段（Build Stage），并且可以方便地复制之前阶段的构件。

示例：

```PowerShell
#阶段一
FROM node:latest AS 阶段0的名字
WORKDIR /usr/src/atsea/app/react-app
COPY react-app .
RUN npm install
RUN npm run build
#阶段二
FROM maven:latest AS 阶段1的名字
WORKDIR /usr/src/atsea
COPY pom.xml .
RUN mvn -B -f pom.xml -s /usr/share/maven/ref/settings-docker.xml dependency
\:resolve
COPY . .
RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package -DskipTests
#阶段三，设置为容器启动时的主程序
FROM java:8-jdk-alpine AS 阶段2的名字
RUN adduser -Dh /home/gordon gordon
WORKDIR /static
COPY --from=storefront /usr/src/atsea/app/react-app/build/ .
WORKDIR /app
COPY --from=appserver /usr/src/atsea/target/AtSea-0.0.1-SNAPSHOT.jar .
ENTRYPOINT ["java", "-jar", "/app/AtSea-0.0.1-SNAPSHOT.jar"]
CMD ["--spring.profiles.active=postgres"]
```

COPY --from 指令，它从之前的阶段构建的镜像中仅复制生产环境相关的应用代码，而不会复制生产环境不需要的构件。

docker 安装

使用 docker 的前提条件是在主机上安装 docker 以支持。

Ubuntu 上安装

方法一：使用官方安装脚本自动安装

```Bash
#原镜像
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
#国内镜像
curl -sSL https://get.daocloud.io/docker | sh
```

方法二：手动安装

1.卸载旧版：Docker 的旧版本被称为 docker、docker.io 或 docker-engine

```Bash
sudo apt-get remove docker docker-engine docker.io containerd runc
```

2.设置 Docker 仓库（非必须）：

在新主机上首次安装 Docker Engine-Community 之前，需要设置 Docker 仓库。之后，您可以从仓库安装和更新 Docker 。

```Bash
#设置apt包索引
$ sudo apt-get update
#安装 apt 依赖包，用于通过HTTPS来获取仓库
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
#添加 Docker 的官方 GPG 密钥
$ curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
#搜索指纹的后8个字符，验证您现在是否拥有带有指纹的密钥
$ sudo apt-key fingerprint 0EBFCD88
#设置稳定版仓库
$ sudo add-apt-repository "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/ $(lsb_release -cs)  stable"
```

3.安装 Docker Engine-Community

```Bash
#更新 apt 包索引
$ sudo apt-get update
```

```
#安装最新版本的 Docker Engine-Community 和 containerd ，或者转到下一步安装特定版本
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
#列出仓库版本列表（安装特定版本使用）
$ apt-cache madison docker-ce
#使用版本字符串（列表第2列完整内容）安装特定版本
$ sudo apt-get install docker-ce=<版本字符串> docker-ce-cli=<版本字符串> containerd.io
#检查Docker是否安装成功，打印出“hello world”等字样
$ sudo docker run hello-world
```

如果不打印hello world，则执行如下命令：

```
systemctl daemon-reload
sudo service docker restart
#按q退出
sudo service docker status
```

3.Shell 脚本安装：

在 Linux 上安装最新版本的Docker Engine-Community。要安装最新的测试版本，请改用 test.docker.com。在下面的每个命令，取代每次出现 get 用 test。

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

使用 Docker 作为非 root 用户，则应考虑使用类似以下方式将用户添加到 docker 组

```
$ sudo usermod -aG docker your-user
```

Ubuntu 18.04 安装

1.更换国内软件源，推荐中国科技大学的源，稳定速度快（可选）

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
sudo sed -i 's/archive.ubuntu.com/mirrors.ustc.edu.cn/g' /etc/apt/sources.list
sudo apt update
```

2.安装需要的包

```
sudo apt install apt-transport-https ca-certificates software-properties-common curl
```

3.添加 GPG 密钥，并添加 Docker-ce 软件源，这里还是以中国科技大学的 Docker-ce 源为例

```
curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
```

4.添加成功后更新软件包缓存

```
sudo apt update
```

5.安装 Docker-ce

```
sudo apt install docker-ce
```

6.设置开机自启动并启动 Docker-ce（安装成功后默认已设置并启动，可忽略）

```
sudo systemctl enable docker
sudo systemctl start docker
```

7.测试运行

```
```

```
sudo docker run hello-world
```

8.添加当前用户到 docker 用户组，可以不用 sudo 运行 docker（可选）

```Bash
sudo groupadd docker
sudo usermod -aG docker $USER
```

9.测试添加用户组（可选）

```Bash
docker run hello-world
```

Centos 7 安装

1.初始准备：安装yum

```Bash
#1.通过 uname -r 命令查看你当前的内核版本
$ uname -r
#2.使用 `root` 权限登录 Centos。确保 yum 包更新到最新
$ sudo yum update
#3.卸载旧版本(如果安装过旧版本的话)
$ sudo yum remove docker  docker-common docker-selinux    docker-engine
```

2.安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

```Bash
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

3.设置yum源

```Bash
$ sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

4.可以查看所有仓库中所有docker版本，并选择特定版本安装

```Bash
$ yum list docker-ce --showduplicates | sort -r
```

5.安装docker

```Bash
#安装最新版
$ sudo yum install docker-ce
#安装指定版
$ sudo yum install <FQPN>
# 例如：sudo yum install docker-ce-17.12.0.ce
```

如果步骤5报错，是container-selinux问题，需要安装epel源再执行步骤5 。

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

6.启动并加入开机启动

```Bash
#启动
$ sudo systemctl start docker
#开机启动
$ sudo systemctl enable docker
```

7.验证安装是否成功(有client和service)、service启动后显示

```Bash
$ docker version
```

如果安装了旧版

```PowerShell
```

```
#卸载旧版（版本为报错尾部）
$ sudo yum erase 版本名
#安装最新版
$ sudo yum install docker-ce
#修改docker.service文件
vim /usr/lib/systemd/system/docker.service
#添加内容：ExecStart=/usr/bin/dockerd --graph /home/docker
```

使用案例

创建一个ubuntu容器

```bash
# 必须使用管理员权限sudo
sudo docker pull ubuntu

# 运行容器，将容器19132端口映射到本地41132端口
sudo docker run -itd --name md-doker -p 41132:19132 ubuntu

# 查看容器列表
sudo docker container ls

# 进入容器
sudo docker exec -it 42fc24a33814 bash
```

创建一个Springboot镜像

1.创建Dockerfile文件

```
FROM java:8
MAINTAINER bingo
ADD demo-0.0.1-SNAPSHOT.jar demo.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","demo.jar"]

# from java:8   拉取一个jdk为1.8的docker image

# maintainer  作者是bingo

# demo-0.0.1-SNAPSHOT.jar 就是你上传的jar包，替换为jar包的名称

# demo.jar  是你将该jar包重新命名为什么名称，在容器中运行

# expose  该容器暴露的端口是多少，就是jar在容器中以多少端口运行

# entrypoint 容器启动之后执行的命令，java -jar demo.jar  即启动jar
```

2.构建镜像

```bash
# 在dockerfile文件所在目录下执行命令
docker build -t my/demo .
```

3.运行容器

```bash
docker run -d --name demo -p 8080:8080 my/demo
```

Springboot的pom修改

```xml
<properties>
    ...
    <docker.image.prefix>包名</docker.image.prefix>
</properties>

<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.4.13</version>
    <configuration>
          <!--镜像-->
```

```
            <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
            <!-- 指定Dockerfile所在的路径 -->
            <dockerDirectory>src/main/docker</dockerDirectory>
            <resources>
                <resource>
                    <targetPath>/</targetPath>
                    <directory>${project.build.directory}</directory>
                    <include>${project.build.finalName}.jar</include>
                </resource>
            </resources>
        </configuration>
    </plugin>
```

项目文件夹下使用maven命令构建：

PowerShell

```
mvn clean package docker:build
```

极速使用

XML

```
 <build>
        <plugins>
            <!-- docker的maven插件，官网：https://github.com/spotify/docker-maven-plugin -->
            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>docker-maven-plugin</artifactId>
                <version>0.4.12</version>
                <configuration>
                    <!-- 注意imageName一定要是符合正则[a-z0-9-_.]的，否则构建不会成功 -->
                    <!-- 详见：https://github.com/spotify/docker-maven-plugin    Invalid repository name ... only [a-z0-9-_.] are allowed-->
                    <imageName>microservice-discovery-eureka</imageName>
                    <baseImage>java</baseImage>
                    <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>
                    <resources>
                        <resource>
                            <targetPath>/</targetPath>
                            <directory>${project.build.directory}</directory>
                            <include>${project.build.finalName}.jar</include>
                        </resource>
                    </resources>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

执行命令：

PowerShell

```
mvn clean package docker:build
```