课程学习目标

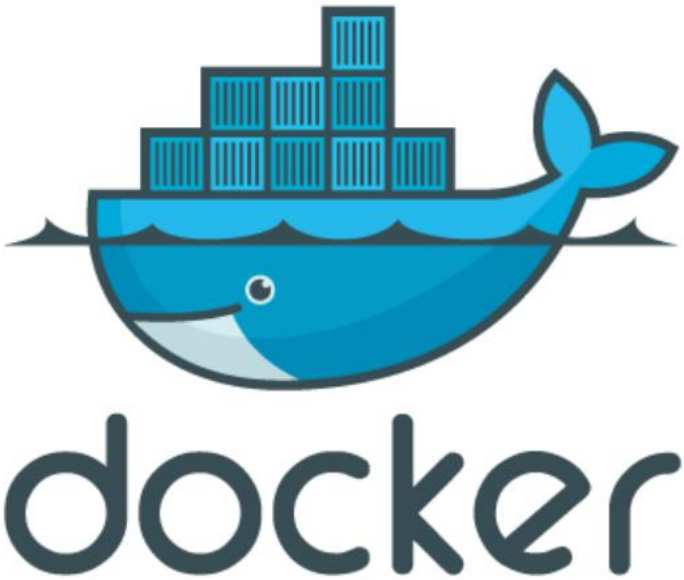
- Docker 简介
- Docker 容器 vs 虚拟化技术
- Docker 安装
- Docker 启动与停止
- Docker 镜像命令
- Docker 容器命令
- Docker 安装 MySQL，Redis
- 搭建 Eureka 微服务，Zuul 网关微服务，文章微服务
- Dockerfile 编写
- Dockerfile 部署 Eureka，Zuul，文章微服务
- IDEA 使用 DockerMaven 插件微服务部署
- Rancher 管理部署微服务
- Docker 私服-registry

1. Docker 简介

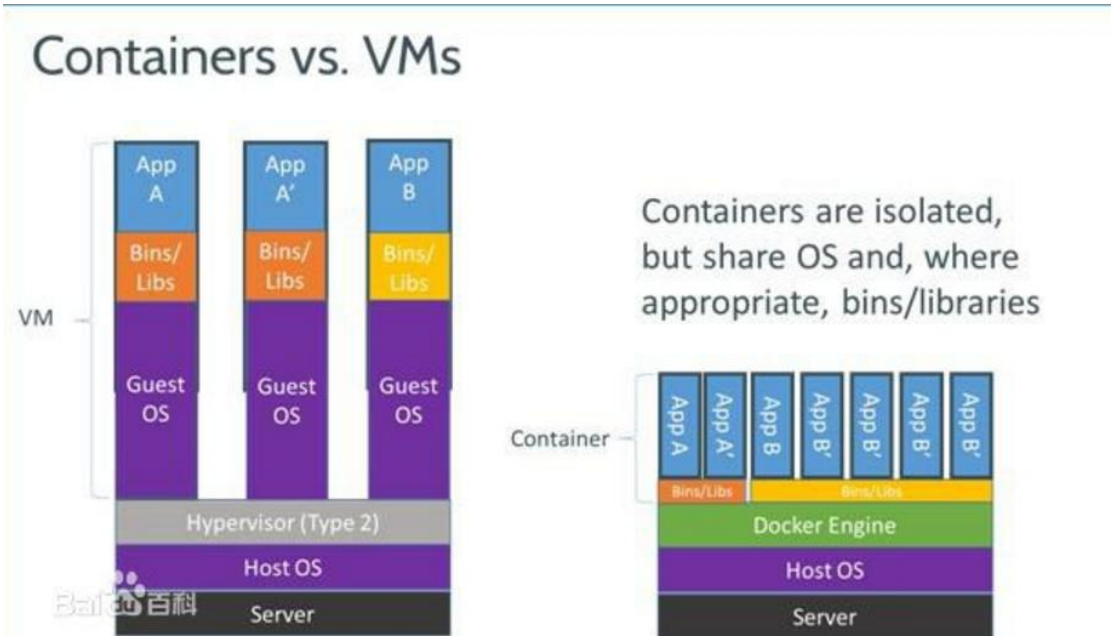
Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

Docker 是 PaaS 提供商 dotCloud 开源的一个基于 LXC 的高级容器引擎，源代码托管在 Github 上，基于 go 语言并遵从 Apache2.0 协议开源。

Docker 自 2013 年以来非常火热，无论是从 github 上的代码活跃度，还是 Redhat 在 RHEL6.5 中集成对 Docker 的支持，就连 Google 的 Compute Engine 也支持 docker 在其之上运行。



2. Docker 容器 vs 虚拟化技术



3. Docker 安装

1）安装 Ubuntu 操作系统

本课程使用 Ubuntu16.045

2）安装 Docker 的 AUFS 存储驱动程序

```
$ sudo apt-get install \
    linux-image-extra-$(uname -r) \
    linux-image-extra-virtual
```

3）安装 docker 包

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

4）添加 Docker 的官方 GPG 密钥

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

5）设置 stable 稳定的仓库(stable 稳定版每季度发布一次，Edge 版每月一次)

```
$ sudo add-apt-repository \ "deb [arch=amd64] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) \ stable"
```

6）更新 apt 包

```
$ sudo apt-get update
```

7）安装 Docker CE

```
$ apt-get install docker-ce
```

8）测试 Docker 是否安装成功

```
$ docker -v
```

4. Docker 启动与停止

4.1. 启动

```
$ sudo service docker start
```

4.2. 停止

```
$ sudo service docker stop
```

4.3. 重启

```
$ sudo service docker restart
```

5. Docker 镜像命令

5.1. 设置 ustc 加速源

```
{
    "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
}
```

5.2. 查看所有镜像

```
$ docker images
```

5.3. 搜索镜像

```
$ docker search redis
```

5.4. 下载镜像

```
$ docker pull redis
```

5.5. 删除镜像

```
$ docker rmi redis
```

6. Docker 容器命令

6.1. 查询运行的容器

```
$ docker ps
```

6.2. 查询所有容器

```
$ docker ps -a
```

6.3. 创建容器

交互式容器：

```
docker run -it --name=容器名称  镜像名称  /bin/bash
```

守护式容器：

```
docker run -di --name=容器名称  镜像名称
```

进入守护式容器：

```
docker exec -it  容器名称  /bin/bash
```

6.4. 删除容器

```
docker rm  容器名称
```

6.5. 启动容器

```
docker start  容器名称
```

6.6. 停止容器

```
docker stop  容器名称
```

6.7. 重启容器

```
docker restart  容器名称
```

6.8. 拷贝文件

从宿主机拷贝到容器内部

```
docker cp  宿主机目录  容器名称:容器目录
```

从容器内部拷贝到宿主机

```
docker cp  容器名称:容器目录  宿主机目录
```

6.9. 目录挂载

```
docker run -di --name=容器名称  -v  宿主机目录:容器目录  镜像命令
```

7. Docker 安装 MySQL， Redis

7.1. 安装 MySQL

7.1.1. 下载 MySQL 镜像

```
docker pull centos/mysql-57-centos7
```

7.1.2. 运行 MySQL 容器

```
docker  run  -di  --name=mysql  -p  3306:3306  -e  MYSQL_ROOT_PASSWORD=123456
centos/mysql-57-centos7
```

7.2. 安装 Redis

7.2.1. 下载 redis 镜像

```
docker pull redis
```

7.2.2. 运行 redis 容器

```
docker run -di --name=redis -p 6379:6379 redis
```

7.3. 安装 Tomcat

7.3.1. 下载 tomcat 镜像

```
docker pull    tomcat:7-jre7
```

7.3.2. 运行 tomcat 容器

```
docker run -di --name=tomcat7 -p 8080:8080 tomcat:7-jre7
```

目录挂载方法运行：

```
docker run -di --name=tomcat7 -p 8080:8080 -v /home/eric/webapps:/usr/local/tomcat/webapps tomcat:7-jre7
```

8. 搭建文章微服务

1）建立模块

2）导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <parent>
        <artifactId>sm1234_parent</artifactId>
        <groupId>cn.sm1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>


    <artifactId>sm1234_article</artifactId>


    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
    </dependencies>
</project>
```

3）编写 application.yml

```
server:
  port: 9001
spring:
  application:
    name: sm1234-article
  datasource:
    url: jdbc:mysql://192.168.66.138:3306/docker?characterEncoding=UTF8
```

```
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: 123456
  jpa:
    database: mysql
    show-sql: true
    generate-ddl: true
```

4）编写启动类

```
package cn.sm1234.aricle;


import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;


/**
 * 文章微服务
 */
@SpringBootApplication
public class ArticleApplication {


    public static void main(String[] args) {
        SpringApplication.run(ArticleApplication.class,args);
    }
}
```

5）编写 pojo 实体类

```
package cn.sm1234.aricle.pojo;
```

```
import javax.persistence.*;

import java.io.Serializable;

import java.util.Date;


/**

 *文章实体

 */

@Entity

@Table(name = "tb_article")

public class Article implements Serializable{


    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Integer id;

    private String title;

    private String content;

    private String author;

    private Date addtime;


    public Integer getId() {

        return id;

    }


    public void setId(Integer id) {

        this.id = id;

    }


    public String getTitle() {

        return title;
```

```
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public Date getAddtime() {
        return addtime;
    }

    public void setAddtime(Date addtime) {
        this.addtime = addtime;
    }
```

```
}
```

6）编写 Dao

```
package cn.sm1234.aricle.dao;


import cn.sm1234.aricle.pojo.Article;

import org.springframework.data.jpa.repository.JpaRepository;


/**

 * 文章 dao

 */

public interface ArticleDao extends JpaRepository<Article,Integer>{


}
```

7）编写 service

```
package cn.sm1234.aricle.service;


import cn.sm1234.aricle.pojo.Article;


import java.util.List;


/**

 * 文章 service 接口

 */

public interface ArticleService {


   public List<Article> findAll();
```

```
    public Article findById(Integer id);

    public void add(Article article);

    public void update(Article article);

    public void deleteById(Integer id);
}
```

```
package cn.sm1234.aricle.service;

import cn.sm1234.aricle.dao.ArticleDao;
import cn.sm1234.aricle.pojo.Article;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * 文章 service 实现
 */
@Service
public class ArticleServiceImpl implements ArticleService {

    @Autowired
    private ArticleDao articleDao;
```

```
    @Override
    public List<Article> findAll() {
        return articleDao.findAll();
    }

    @Override
    public Article findById(Integer id) {
        return articleDao.findById(id).get();
    }

    @Override
    public void add(Article article) {
        articleDao.save(article);
    }

    @Override
    public void update(Article article) {
        articleDao.save(article);
    }

    @Override
    public void deleteById(Integer id) {
        articleDao.deleteById(id);
    }
}
```

8）编写 Controller

```
package cn.sm1234.aricle.controller;
```

```
import cn.sm1234.aricle.pojo.Article;
import cn.sm1234.aricle.pojo.Result;
import cn.sm1234.aricle.service.ArticleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;


/**
 * 文章 Controller
 */
@RestController
@RequestMapping("/article")
public class ArticleController {


    @Autowired
    private ArticleService articleService;


    /**
     * 查询所有
     */
    @RequestMapping(method = RequestMethod.GET)
    public Result findAll(){
        return new Result(true,"查询成功",articleService.findAll());
    }


    /**
     * 查询一个
     */
    @RequestMapping(value = "/{id}",method = RequestMethod.GET)
```

```
    public Result findById(@PathVariable Integer id){
        return new Result(true,"查询成功",articleService.findById(id));
    }

    /**
     * 添加
     */
    @RequestMapping(method = RequestMethod.POST)
    public Result add(@RequestBody Article article){
        articleService.add(article);
        return new Result(true,"添加成功");
    }

    /**
     * 修改
     */
    @RequestMapping(value = "/{id}",method = RequestMethod.PUT)
    public Result update(@RequestBody Article article,@PathVariable Integer id){
        article.setId(id);
        articleService.update(article);
        return new Result(true,"修改成功");
    }

    /**
     * 删除
     */
    @RequestMapping(value = "/{id}",method = RequestMethod.DELETE)
    public Result deleteById(@PathVariable Integer id){
        articleService.deleteById(id);
```

```
        return new Result(true,"删除成功");
    }
}
```

运行启动类，使用 postman 进行 CRUD 测试

9. 搭建 Eureka 微服务

1）创建模块

2）导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>sm1234_parent</artifactId>
        <groupId>cn.sm1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>sm1234_eureka</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
    </dependencies>
</project>
```

3）编写 application.yml

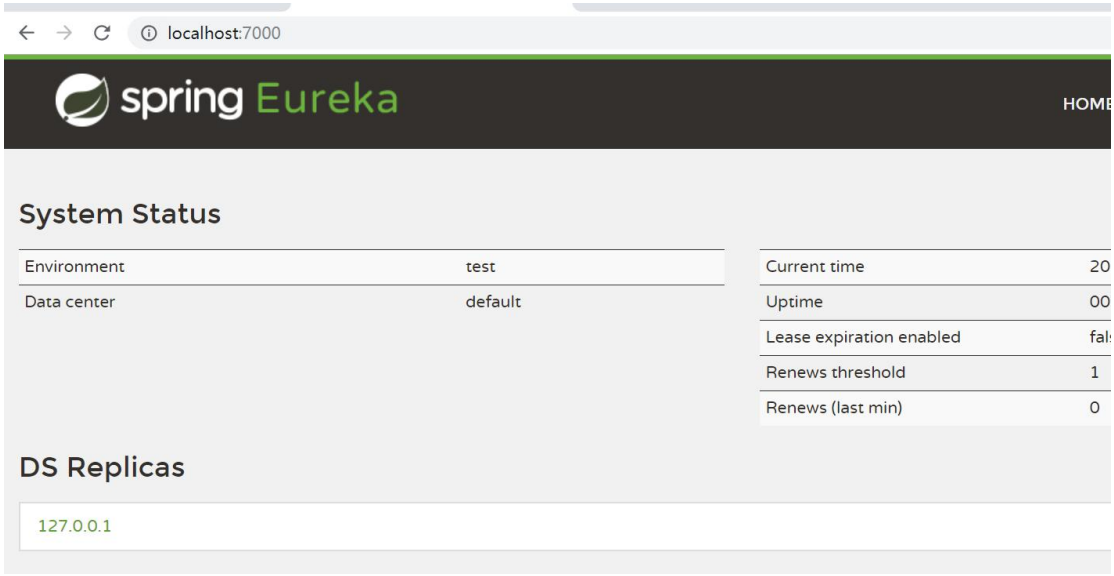
```
server:
  port: 7000
spring:
  application:
    name: sm1234-eureka
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://127.0.0.1:${server.port}/eureka
```

4）编写启动类

```
package cn.sm1234.eureka;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/**
 * 注册中心微服务
```

```
 */
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {


    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class,args);
    }


}
```

6）启动，就看到以下界面



10. 搭建 Zuul 网关微服务

1）搭建模块

2）导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>sm1234_parent</artifactId>
        <groupId>cn.sm1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>sm1234_zuul</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>
    </dependencies>
</project>
```

3）编写 application.yml

```
server:
  port: 8888
spring:
  application:
    name: sm1234-zuul
zuul:
  routes:
    app:
```

```
    path: /app/*

    serviceId: sm1234-article
```

4）编写启动类

```
package cn.sm1234.zuul;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;


/**

 * 微服务网关

 */

@SpringBootApplication

@EnableZuulProxy

public class ZuulApplication {


    public static void main(String[] args) {

        SpringApplication.run(ZuulApplication.class,args);

    }


}
```

11. 文章微服务和 Zuul 网关注册到 Eureka

同时在文章微服务和 Zuul 网关微服务加入以下：

1）导入依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2）在 application.yml 加入

```
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://127.0.0.1:7000/eureka
  instance:
    prefer-ip-address: true
    instance-id: zuul.com
```

3）在启动类加入注解

```
@EnableEurekaClient
```

12. 学习 Dockerfile 编写

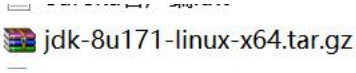
命令	作用
FROM image_name:tag	定义了使用哪个基础镜像启动构建流程
MAINTAINER user_name	声明镜像的创建者

ENV key value	设置环境变量 (可以写多条)

RUN command	是 Dockerfile 的核心部分(可以写多条)
ADD source_dir/file dest_dir/file	将宿主机的文件复制到容器内， 如果是一个压缩文件， 将会在复制后自动解压
COPY source_dir/file dest_dir/file	和 ADD 相似， 但是如果有压缩文件并不能解压
WORKDIR path_dir	设置工作目录
EXPOSE port1 prot2	用来指定端口， 使容器内的应用可以通过端口和外界交 互
CMD argument	在构建容器时使用， 会被 docker run 后的 argument 覆 盖
ENTRYPOINT argument	和 CMD 相似， 但是并不会被 docker run 指定的参数覆盖
VOLUME	将本地文件夹或者其他容器的文件挂载到容器中

需求：使用 Dockerfile 制作一个 jdk1.8 镜像

1）上传 jdk 安装文件到 Ubuntu



2）在 jdk 安装文件同目录下创建 Dockerfile 文件

```
FROM ubuntu

MAINTAINER eric

RUN mkdir /usr/local/jdk
```

```
WORKDIR /usr/local/jdk

ADD jdk-8u171-linux-x64.tar.gz /usr/local/jdk



ENV JAVA_HOME /usr/local/jdk/jdk1.8.0_171

ENV JRE_HOME /usr/local/jdk/jdk1.8.0_171/jre

ENV PATH $JAVA_HOME/bin:$PATH
```

3）执行以下命令构建镜像

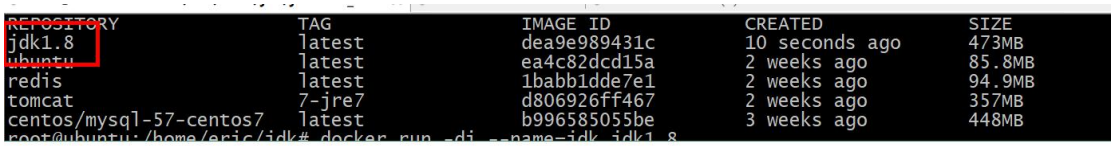
```
docker build -t 镜像名称 .
```

注意：空格和点不能少！

```
Sending build context to Docker daemon  190.9MB
Step 1/8 : FROM ubuntu
 ---> ea4c82dcd15a
Step 2/8 : MAINTAINER eric
 ---> Using cache
 ---> cef8fe3ca46d
Step 3/8 : RUN mkdir /usr/local/jdk
 ---> Using cache
 ---> 07f689c2c977
Step 4/8 : WORKDIR /usr/local/jdk
 ---> Using cache
 ---> 820dbec7a4f8
Step 5/8 : ADD jdk-8u171-linux-x64.tar.gz /usr/local/jdk
 ---> Using cache
 ---> 82cfdfdee15b
Step 6/8 : ENV JAVA_HOME /usr/local/jdk/jdk1.8.0_171
 ---> Using cache
 ---> 24e1f296f387
Step 7/8 : ENV JRE_HOME /usr/local/jdk/jdk1.8.0_171/jre
 ---> Using cache
 ---> 1c4c0dbc839d
Step 8/8 : ENV PATH $JAVA_HOME/bin:$PATH
 ---> Using cache
 ---> dea9e989431c
Successfully built dea9e989431c
```

4）查看是否构建成功

```
docker images
```



13. Dockerfile 部署 Eureka，Zuul，文章服务

1）在微服务项目 pom.xml 加入 springboot 打包插件

```
<build>
    <finalName>app</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

2）在 cmd 打包项目

```
mvn clean package
```

3）app.jar 上传到 linux

4）编写 Dockerfile

```
FROM jdk1.8
ADD app.jar /app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

5）构建镜像

```
docker build -t 镜像名称 .
```

6）运行容器

7）访问微服务

14. 使用 DockerMaven 插件微服务部署

1）修改 docker 的配置，开放远程部署端口

```
vi /lib/systemd/system/docker.service
```

其中 ExecStart=后添加配置

-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock

2）在每个微服务加入 dockermaven 插件

```
<build>
    <finalName>app</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <!-- 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>0.4.13</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>article</imageName>
```

```
            <baseImage>jdk1.8</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>
            <dockerHost>http://192.168.66.138:2375</dockerHost>
        </configuration>
    </plugin>
  </plugins>
</build>
```

注意：每个微服务的<imageName>是不一样的！

3）在 cmd 运行

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

4）查看 docker 看镜像是否构建成功

15. Rancher 管理部署微服务

15.1. 什么是 Rancher

Rancher 是一个开源的企业级容器管理平台。通过 Rancher，企业再也不必自己

使用一系列的开源软件去从头搭建容器服务平台。Rancher 提供了在生产环境中使用的管理 Docker 和 Kubernetes 的全栈化容器部署与管理平台。

官网：https://rancher.com/

16. Docker 私服-registry

16.1. 下载 registry

```
docker pull registry
```

16.2. 运行 registry

```
docker run -di --name=registry -p 5000:5000 registry
```

访问 registry:

http://192.168.66.138:5000/v2/_catalog

16.3. 配置 registry

修改 vi /etc/docker/daemon.json

添加：

```
"insecure-registries":["192.168.66.138:5000"]
```

添加之后重启 docker，这时 docker 就信任 registry 地址了。

16.4. 上传镜像到 registry

1）手动上传

```
docker tag eureka 192.168.66.138:5000/eureka


docker push 192.168.66.138:5000/eureka
```

2）DockerMaven 上传

```
<build>
    <finalName>app</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <!-- 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>0.4.13</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>192.168.66.138:5000/zuul</imageName>
                <baseImage>jdk1.8</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>
                <dockerHost>http://192.168.66.138:2375</dockerHost>
```

```
            </configuration>
        </plugin>
    </plugins>
</build>
```

16.5. 从 registry 下载镜像

```
docker pull 192.168.66.138:5000/eureka
```