**Docker**

优就业.JAVA教研室

# 课程目标

目标1：了解Docker简介、版本、架构

目标2：掌握Docker的安装、启动、停止方法

目标3：掌握Docker镜像加速器配置

目标4：掌握Docker镜像操作

目标5：掌握Docker容器操作

目标6：掌握jdk、mysql tomcat nginx 等容器的部署方法

目标7：掌握 DockerFile使用方法

目标8：掌握 Nexus搭建Docker私有镜像仓库

目标9：掌握 Maven Docker插件使用方法

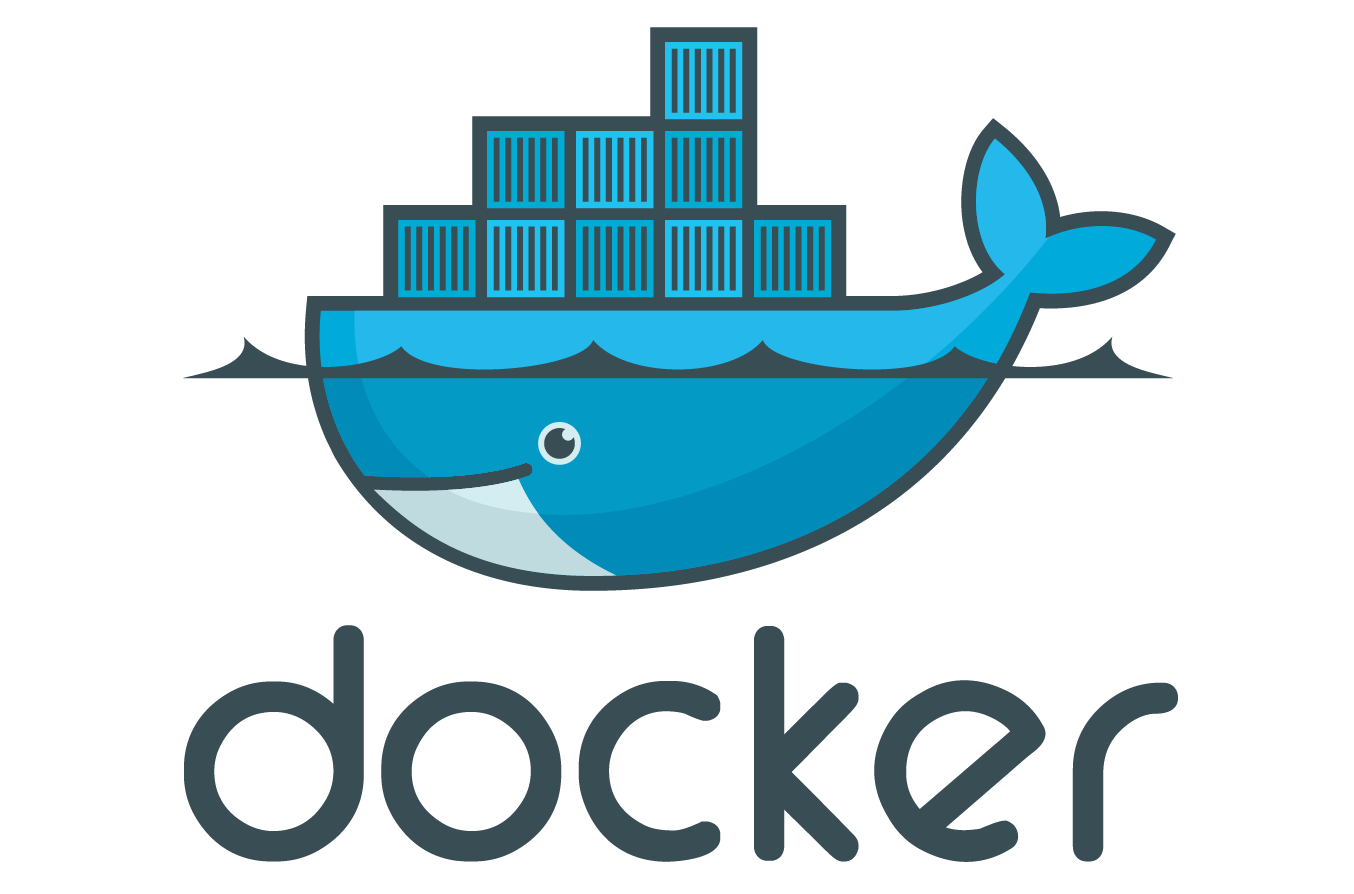
目标10：掌握 手动Docker发布微服务方法

目标11：掌握 Docker Compose使用方法

目标12：掌握 Jekins自动化集成工具使用方法

# 1.Docker简介

Docker是一个开源的容器引擎，它可以帮助我们更快地交付应用。Docker可将应用程序和基础设施层隔离，并且能将基础设施当作程序一样进行管理。使用Docker，可更快地打包、测试以及部署应用程序，并可减少从编写到部署运行代码的周期。



Docker官方网站：<https://www.docker.com/>

## 1.1、Docker版本

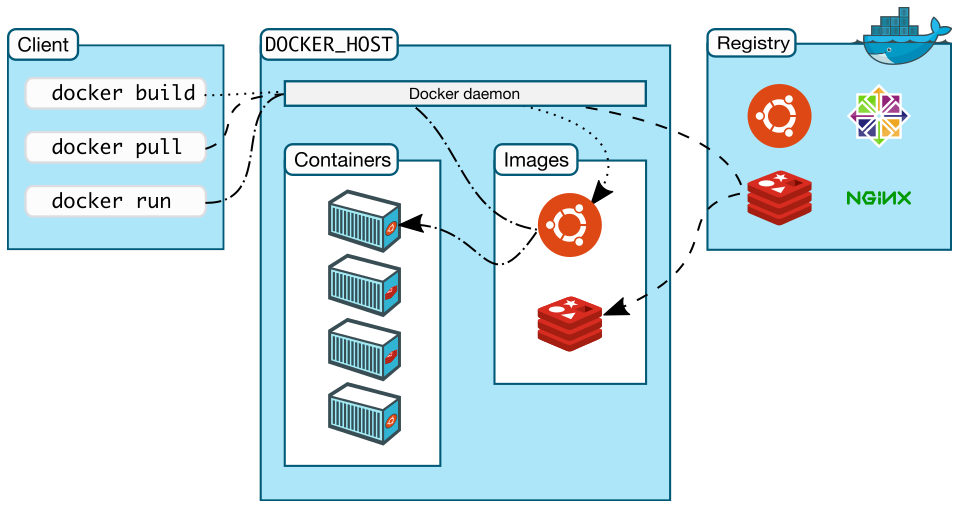
1、Docker EE（企业版）

Docker EE由公司支持，可在经过认证的操作系统和云提供商中使用，并可运行来自Docker Store的经过认证的容器和插件。

2、Docker CE（免费版）

Docker CE是免费的Docker产品的新名称，Docker CE包含了完整的Docker平台，非常适合开发人员和运维团队构建容器APP

## 1.2、 Docker架构



### 1.2.1、Docker daemon（Docker守护进程）

Docker daemon是一个运行在宿主机（DOCKER\_HOST）的后台进程。我们可通过Docker客户端与之通信。

### 1.2.2、Client（Docker客户端）

Docker客户端是Docker的用户界面，它可以接受用户命令和配置标识，并与Docker daemon通信。图中，docker build等都是Docker的相关命令。

### 1.2.3、 Images（Docker镜像）

Docker镜像是一个只读模板，它包含创建Docker容器的说明。它和系统安装光盘有点像——我们使用系统安装光盘安装系统，同理，我们使用Docker镜像运行Docker镜像中的程序。

### 1.2.4、 Container（容器）

容器是镜像的可运行实例。镜像和容器的关系有点类似于面向对象中，类和对象的关系。我们可通过Docker API或者CLI命令来启停、移动、删除容器。

### 1.2.5、 Registry（仓库）

Docker Registry是一个集中存储与分发镜像的服务。我们构建完Docker镜像后，就可在当前宿主机上运行。但如果想要在其他机器上运行这个镜像，我们就需要手动拷贝。此时，我们可借助Docker Registry来避免镜像的手动拷贝。

一个Docker Registry可包含多个Docker仓库；每个仓库可包含多个镜像标签；每个标签对应一个Docker镜像。这跟Maven的仓库有点类似，如果把Docker Registry比作Maven仓库的话，那么Docker仓库就可理解为某jar包的路径，而镜像标签则可理解为jar包的版本号。

# Docker安装启动

## 2.1、运行安装环境

采用Linux操作系统，版本为：Centos7.0 64位

Docker采用CE版：18.06.0.ce-3

## 2.2、 Docker安装过程

### 2.2.1、 下载Docker-CE rpm安装包

https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/

注意下载版本要和操作系统版本对应。

docker-ce-18.06.2.ce-3.el7.x86\_64.rpm

### 2.2.2、 安装Docker

执行命令:yum install -y docker-ce-18.06.2.ce-3.el7.x86\_64.rpm

注意：确保linux服务器处于联网状态。

### 2.2.3、 启动与停止Docker

启动docker：systemctl start docker

停止docker：systemctl stop docker

重启docker：systemctl restart docker

查看docker状态：systemctl status docker

### 2.2.3、 验证Docker是否安装成功

执行命令：docker info

查看返回信息，能否查看到Docker相关信息

到此，Docker安装完毕！

# Docker配置镜像加速器

国内访问Docker Hub的速度很不稳定，有时甚至出现连接不上的情况。我们来为Docker配置镜像加速器，从而解决这个问题。目前国内很多云服务商都提供了镜像加速的服务。

常用的镜像加速器有：阿里云加速器、DaoCloud加速器等。各厂商镜像加速器的使用方式大致类似，笔者以阿里云加速器为例进行讲解。

注册阿里云账号后，即可在阿里云控制台（https://cr.console.aliyun.com/#/accelerator）看到类似如下的页面。



### 3.1、编辑：/etc/docker/daemon.json

{

"registry-mirrors": ["https://w5z91a3d.mirror.aliyuncs.com"]

}

### 3.2、修改保存后输入如下命令重启Docker守护进程和服务

systemctl daemon-reload

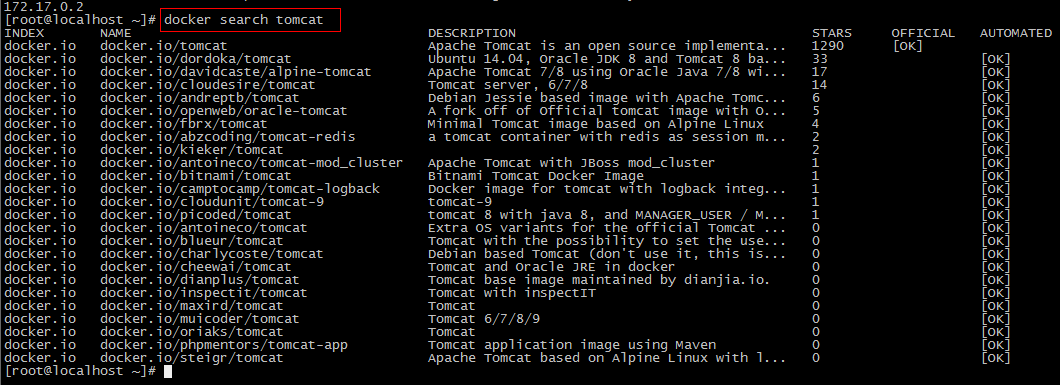
systemctl restart docker

# Docker镜像操作常用指令

## 4.1、搜索镜像

可使用docker search 命令搜索存放在Docker Hub中的镜像

|  |
| --- |
| 执行命令：docker search tomcat |



列说明：

NAME：仓库名称

DESCRIPTION：镜像描述

STARS：用户评价，反应一个镜像的受欢迎程度

OFFICIAL：是否官方

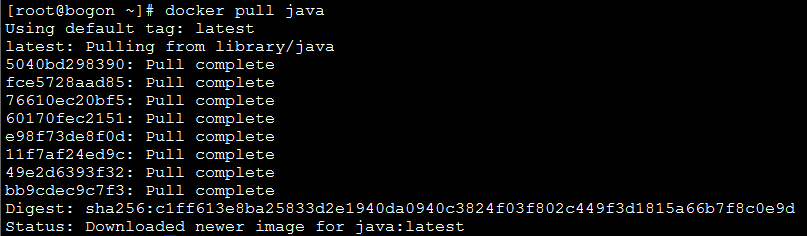
AUTOMATED：自动构建，表示该镜像由Docker Hub自动构建流程创建的

## 4.2、下载镜像

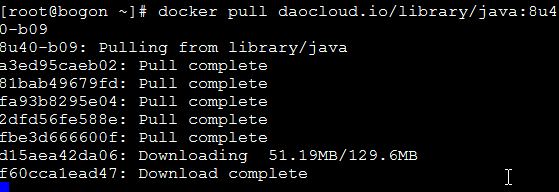
使用命令docker pull 命令即可从Docker Registry上下载镜像。

|  |
| --- |
| 执行命令1：docker pull java |

执行该命令后，Docker会从Docker Hub中的java仓库下载最新版本的Java镜像



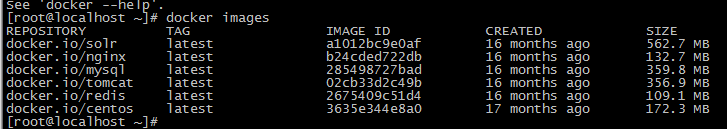
|  |
| --- |
| 执行命令2：docker pull daocloud.io/library/java:8u40-b09 |

执行该命令后，Docker会从指定的Docker Registry中下载标签为8的Java镜像。

## 4.3、列出本地镜像

使用docker images 命令即可列出已下载的镜像。

|  |
| --- |
| 执行命令：docker images |

执行该命令后，将会看到类似于如下的表格：

REPOSITORY：镜像所在的仓库名称

TAG：镜像标签

IMAGE ID：镜像ID

CREATED：镜像的创建日期（不是获取该镜像的日期）

SIZE：镜像大小

这些镜像都是存储在Docker宿主机的/var/lib/docker目录下

## 4.4、删除本地镜像

使用docker rmi 命令即可删除指定镜像。

|  |
| --- |
| 执行命令1：docker rmi hello-world |

执行以上指令，就会删除hello-world这个镜像

注意：已经使用镜像创建过容器的不能删除

|  |
| --- |
| 执行命令2：docker rmi -f $(docker images) |

执行以上指令，就会删除全部镜像

## 4.5、保存镜像

使用docker save 即可保存镜像。

|  |
| --- |
| 执行命令：docker save -o centos7.tar centos:7 |

-o 参数后面跟要把镜像保存的文件路径和名称

## 4.6、加载镜像

使用docker load 即可加载镜像

|  |
| --- |
| 执行命令：docker load -i centos7.tar |

-i参数后面跟要加载的文件路径和名称

# Docker容器操作常用指令

## 5.1、新建并启动容器

使用以下docker run 命令即可新建并启动一个容器。该命令是我们最常用的命令了，它有很多选项，下面列举一些常用的选项。

① -i选项：表示运行容器

② -d选项：表示后台运行

③ -t：表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。

④ -P选项：随机端口映射

⑤-p选项：表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个－p做多个端口映射

⑥--name ：指定容器的名称

⑦ –network选项：指定网络模式，该选项有以下可选参数：

–network=bridge： 默认选项，表示连接到默认的网桥。

–network=host：容器使用宿主机的网络。

–network=container:NAME\_or\_ID：告诉Docker让新建的容器使用已有容器的网络配置。

–network=none：不配置该容器的网络，用户可自定义网络配置。

|  |
| --- |
| 执行命令1：docker run java /bin/echo 'Hello World' |

终端会打印Hello World的字样，跟在本地直接执行/bin/echo 'Hello World' 一样

|  |
| --- |
| 执行命令2：docker run -it --name=mycentos centos:7 /bin/bash |

就能启动一个Centos容器，并进入到容器的命令行终端。在本例2个参数，含义如下：

-t #容器启动后会进入其命令行

-i #运行容器

--name #指定容器的名字

|  |
| --- |
| 执行命令3：docker run -d -p 91:80 nginx |

就能启动一个Nginx容器。在本例中，我们为docker run添加了2个参数，含义如下：

-d # 后台运行

-p 宿主机端口:容器端口 # 开放容器端口到宿主机端口

## 5.2、列出容器

使用docker ps 命令即可列出运行中的容器。执行该命令后，可看到类似于如下的表格。

|  |
| --- |
| 执行命令：docker ps |

3

如需列出所有容器（包括已停止的容器），可使用-a参数；只查看已经退出的容器使用-q参数

该表格包含了七列，含义如下：

① CONTAINER\_ID：表示容器ID。② IMAGE：表示镜像名称。③ COMMAND：表示启动容器时运行的命令。

④ CREATED：表示容器的创建时间。

⑤ STATUS：表示容器运行的状态。Up表示运行中，Exited表示已停止。

⑥ PORTS：表示容器对外的端口号。

⑦ NAMES：表示容器名称。该名称默认由Docker自动生成，也可使用docker run命令的–name选项自行指定。

## 5.3、停止容器

使用docker stop 命令，即可停止容器

|  |
| --- |
| 执行命令：docker stop 784fd3b294d7 |

其中784fd3b294d7 是容器ID，当然也可使用docker stop 容器名称 来停止指定容器。

## 5.4、强制停止容器

可使用docker kill 命令停止一个或更多运行着的容器。

|  |
| --- |
| 执行命令：docker kill 784fd3b294d7 |

## 5.5、启动已经停止容器

使用docker start 命令，即可启动已经停止的容器。

|  |
| --- |
| 执行命令：docker start 784fd3b294d7 |

其中784fd3b294d7 是容器ID，当然也可使用docker stop 容器名称 来启动指定容器。

## 5.6、重启容器

可使用docker restart 命令重启一个容器。该命令实际上是先执行了docker stop 命令，然后执行了docker start 命令。

|  |
| --- |
| 执行命令：docker restart 784fd3b294d7 |

## 5.7、进入容器

一些时候，我们可能需要进入运行中的容器。

1. 使用docker exec 命令进入容器

|  |
| --- |
| 执行命令：docker exec -it 784fd3b294d7 /bin/bash |

## 5.8、拷贝文件到容器

如果我们需要将文件拷贝到容器内可以使用docker cp命令

|  |
| --- |
| 执行命令1：docker cp /usr/local/1.txt c1:/usr/local |

参数说明：需要拷贝的文件或目录 容器名称:容器目录

执行以上命令即可把本地文件拷贝到容器指定目录。

|  |
| --- |
| 执行命令2：docker cp c1:/usr/local/1.txt /usr/local |

参数说明：容器名称:容器目录 需要拷贝的文件或目录

执行以上命令即可把容器中的文件拷贝到宿主机指定目录。

## 5.9、目录挂载

我们可以在创建容器的时候，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器。

|  |
| --- |
| docker run -di -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos2 centos:7 |

参数说明：创建容器 添加-v参数 后边为 宿主机目录:容器目录

如果你共享的是多级的目录，可能会出现权限不足的提示。



这是因为CentOS7中的安全模块selinux把权限禁掉了，我们需要添加参数 --privileged=true 来解决挂载的目录没有权限的问题

|  |
| --- |
| docker run -di -v /usr/local/myhtml:/usr/local/myhtml --privileged=true --name=mycentos2 centos:7 |

## 5.10、删除容器

使用docker rm 命令即可删除指定容器。

|  |
| --- |
| 执行命令1：docker rm 784fd3b294d7 |

删除指定容器，该命令只能删除已停止的容器，如需删除正在运行的容器，可使用-f参数。

|  |
| --- |
| 执行命令2：docker rm -f $(docker ps -a -q) |

删除所有的容器。

## 5.11、导入、导出容器

将容器导出成一个压缩包文件。

|  |
| --- |
| 执行命令：docker export --output="latest.tar" centos1 |

使用docker import 命令即可从归档文件导入内容并创建镜像。

|  |
| --- |
| 执行命令：docker import nginx2.tar nginx |

## 5.12、查看容器ip

我们可以通过以下命令查看容器运行的各种数据

|  |
| --- |
| 执行命令1：docker inspect mycentos2 |

也可以直接执行下面的命令直接输出IP地址

|  |
| --- |
| 执行命令2：docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2 |

最后跟容器的名称。

## 5.13、查看容器日志

docker logs [options] 容器获取容器的日志

docker logs --tail="10" mytest

//--tail="10" 查看最后10行

## 5.14、退出容器

退出容器，只需要在容器内输入命令exit

# java开发者Docker常用镜像容器创建

## 6.1、jdk部署

### 6.1.1、 拉取jdk镜像

|  |
| --- |
| 执行命令：docker pull java:8 |

### 6.1.2、 创建jdk容器

|  |
| --- |
| 执行命令： docker run -di --name jdk1 java:8 |

说明：jdk安装的是openjdk，具体安装路径是：/usr/lib/jvm/java-1.8.0-openjdk-amd64

## 6.2、mysql部署

### 6.2.1、 拉取mysql镜像

|  |
| --- |
| 执行命令：docker pull mysql:5.7 |

### 6.2.2、 创建mysql容器

|  |
| --- |
| 执行命令：docker run -di --name=offcn\_mysql -p 33306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -e MYSQL\_DATABASE=springcloud01 mysql |

-p 代表端口映射，格式为 宿主机映射端口:容器运行端口

-e 代表添加环境变量 MYSQL\_ROOT\_PASSWORD是root用户的登陆密码

-e MYSQL\_DATABASE=springcloud01 指定跟随容器一起创建数据库springcloud01

### 6.2.3、 连接mysql容器数据库

远程连接mysql数据库只需要使用宿主机ip和端口33306就可以

### 6.2.4、 解决mysql数据库乱码问题

从mysql容器拷贝mysql配置文件

docker cp db1:/etc/mysql/mysql.conf.d/mysqld.cnf .

在宿主机修改配置文件mysqld.cnf

|  |
| --- |
| [mysqld]  character-set-server=utf8  collation-server=utf8\_general\_ci |

拷贝配置文件到容器

docker cp mysqld.cnf db1:/etc/mysql/mysql.conf.d/mysqld.cnf

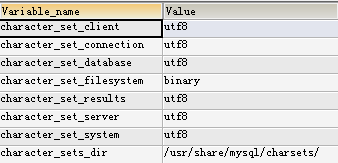
重启mysql

systemctl restart mysqld

登录mysql：mysql -uroot -p123

查看mysql编码状态

mysql> show variables like 'character\_set\_%';



## 6.3、tomcat部署

### 6.3.1、 拉取tomcat镜像

|  |
| --- |
| 执行命令：docker pull tomcat:8.5-jre8 |

### 6.3.2、 创建tomcat容器

|  |
| --- |
| 执行命令：docker run -di --name= tomcat1 -p 9000:8080 tomcat:8.5-jre8 |

## 6.4、nginx部署

### 6.4.1、 拉取nginx镜像

|  |
| --- |
| 执行命令：docker pull nginx |

### 6.4.2、 创建nginx容器

|  |
| --- |
| 执行命令：docker run -di --name=n1 -p 89:80 nginx |

# Dockerfile入门

Dockerfile是由一系列命令和参数构成的脚本，这些命令应用于基础镜像（centos或者ubuntu，即操作系统级别的镜像）并最终创建一个新的镜像。

1、对于开发人员：可以为开发团队提供一个完全一致的开发环境；

2、对于测试人员：可以直接拿开发时所构建的镜像或者通过Dockerfile文件构建一个新的镜像开始工作了；

3、对于运维人员：在部署时，可以实现应用的无缝移植。

## 7.1、Dockerfile常用命令

|  |  |
| --- | --- |
| **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)，可以在创建容器的时候传递参数给容器 |
| RUN command | 是Dockerfile的核心部分，执行系统指令(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 |
| CMD | 指定容器启动时运行的命令 |

## 7.2、Dockerfile搭建jdk环境

### 第一步:在宿主机创建jdk存储目录

mkdir /usr/local/dockerjdk

### 第二步：下载jdk-8u131-linux-x64.tar.gz并上传到服务器（宿主机）中的/usr/local/dockerjdk目录

### 第三步：在/usr/local/dockerjdk目录下创建文件Dockerfile

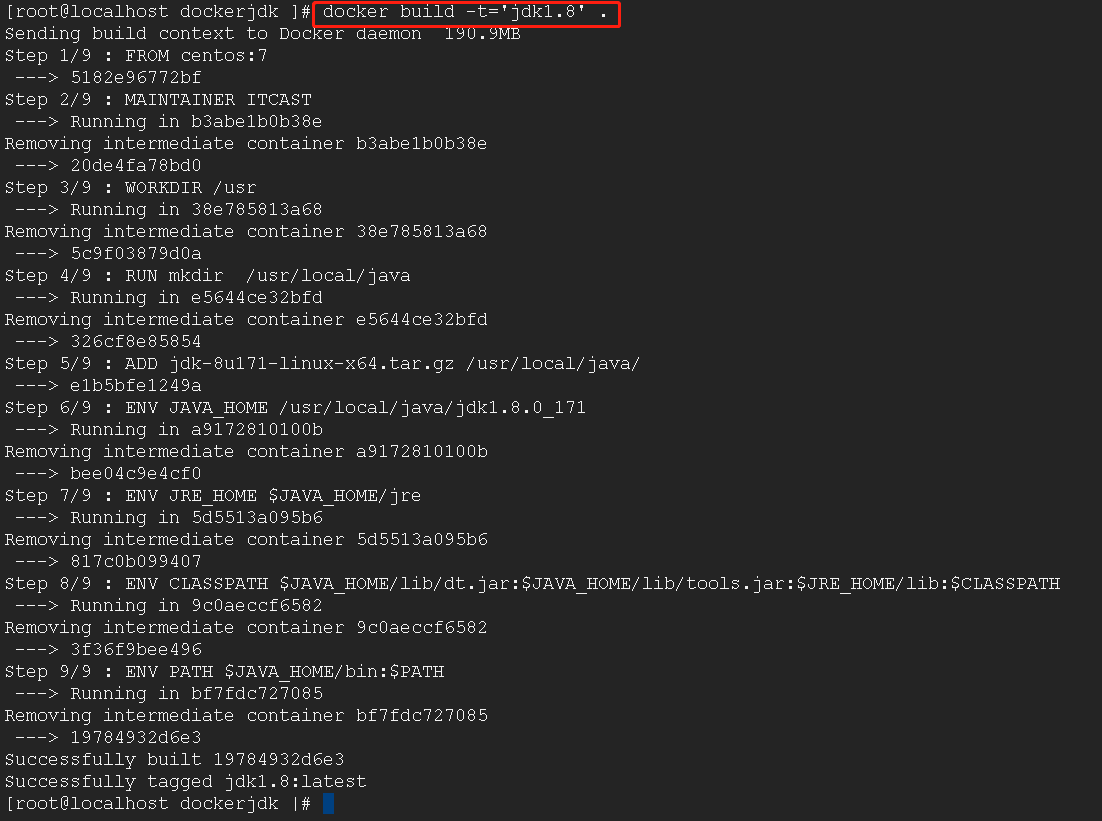
vi Dockerfile

|  |
| --- |
| FROM centos:7 #设置基础镜像名称  MAINTAINER ujiuye #设置镜像作者  WORKDIR /usr #设置工作目录  RUN mkdir /usr/local/java #在镜像创建目录  ADD jdk-8u131-linux-x64.tar.gz /usr/local/java/ #复制宿主机jdk安装包到镜像（自动解压）  ENV JAVA\_HOME /usr/local/java/jdk1.8.0\_131 #设置环境变量JAVA\_HOME  ENV CLASSPATH $JAVA\_HOME/lib #设置环境变量 CLASSPATH  ENV PATH $JAVA\_HOME/bin:$PATH #设置环境变量 PATH |

### 第四步：执行命令构建镜像

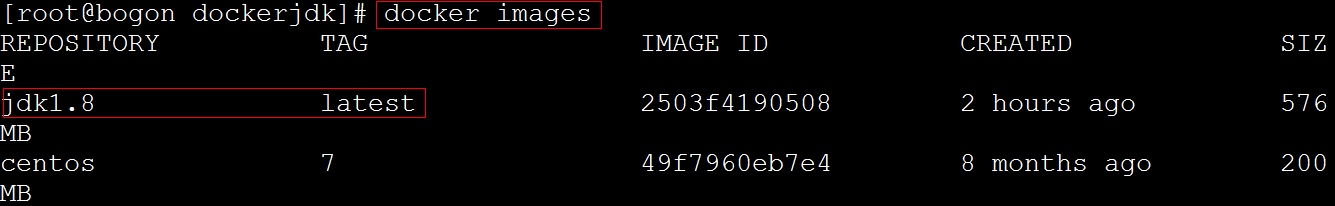
docker build -t='jdk1.8' .

注意后边的空格和点（.表示当前目录，也就是Dockerfile所在目录），不要省略



### 第五步：查看镜像是否建立完成

docker images



## 7.3、Dockerfile搭建tomcat环境镜像

### 第一步:在宿主机创建tomcat存储目录

mkdir /usr/local/dockertomcat

### 第二步：上传安装包到服务器

下载jdk-8u131-linux-x64.tar.gz、apache-tomcat-8.5.15.tar.gz并上传到服务器（宿主机）中的/usr/local/dockertomcat目录

### 第三步：在/usr/local/dockertomcat目录下创建文件Dockerfile

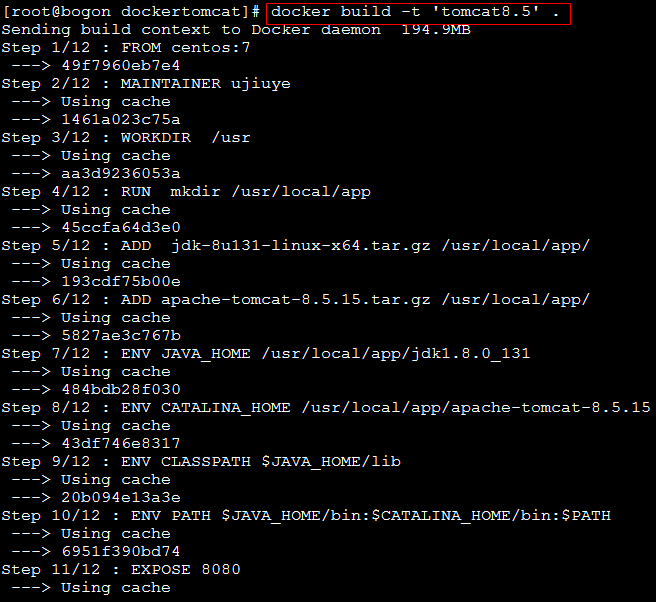
vi Dockerfile

|  |
| --- |
| FROM centos:7 #设置基础镜像名称  MAINTAINER ujiuye #设置镜像作者  WORKDIR /usr #设置工作目录  RUN mkdir /usr/local/app #在镜像创建目录  ADD jdk-8u131-linux-x64.tar.gz /usr/local/app/ #复制宿主机jdk安装包到镜像（自动解压）  ADD apache-tomcat-8.5.15.tar.gz /usr/local/app/  ENV JAVA\_HOME /usr/local/app/jdk1.8.0\_131 #设置环境变量JAVA\_HOME  ENV CATALINA\_HOME /usr/local/app/apache-tomcat-8.5.15  ENV CLASSPATH $JAVA\_HOME/lib #设置环境变量 CLASSPATH  ENV PATH $JAVA\_HOME/bin:$CATALINA\_HOME/bin:$PATH #设置环境变量 PATH  EXPOSE 8080 #暴露8080端口  CMD ["/usr/local/app/apache-tomcat-8.5.15/bin/catalina.sh","run"] #容器启动时运行tomcat |

### 第四步：执行命令构建镜像

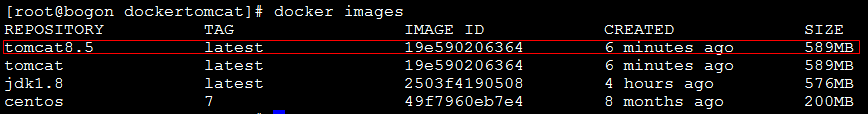
docker build -t='tomcat8.5' .

注意后边的空格和点（.表示当前目录），不要省略



### 第五步：查看镜像是否建立完成

docker images

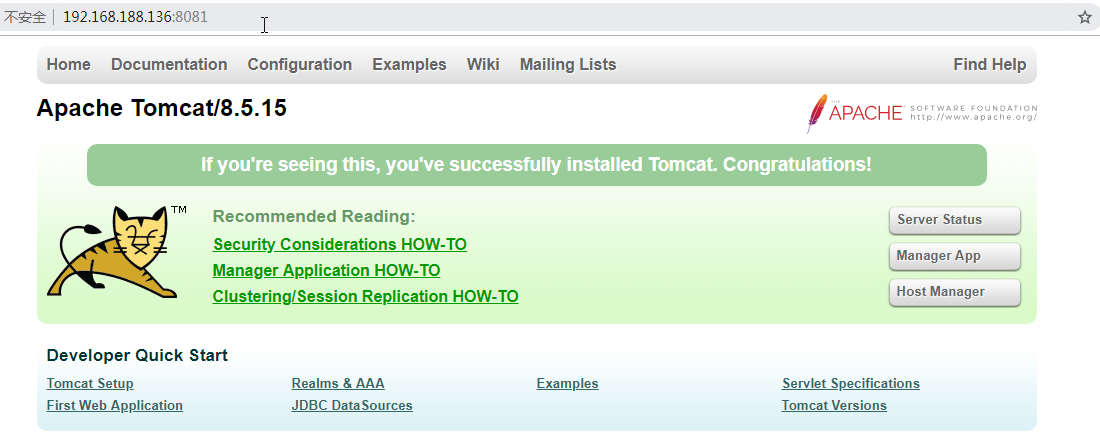


### 第六步：使用镜像创建运行容器

docker run -di --name=t1 -p 8081:8080 tomcat8.5

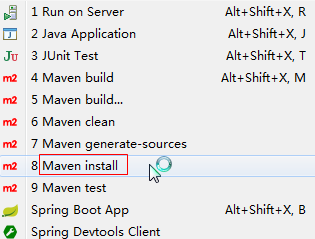
### 第七步：访问测试tomcat

<http://ip:8081>



## 7.5、Dockerfile 创建微服务java镜像

### 第一步、编译工程EurekaServer01执行构建指令



注意：EurekaServer01服务器单独部署，暂不使用集群，需要修改属性配置文件

|  |
| --- |
| #服务器端口号  server.port=8888  #应用名称  spring.application.name=EurekaServer01  #是否将自己注册到Eureka服务中，本身就是所有无需注册  eureka.client.register-with-eureka=false  #是否从Eureka中获取注册信息  eureka.client.fetch-registry=false  #EurekaServer注册中心地址  eureka.client.service-url.defaultZone=http://localhost:${server.port}/eureka |

得到构建好的jar包：EurekaServer01-0.0.1-SNAPSHOT.jar

### 第二步:在宿主机创建jar存储目录

mkdir /usr/local/dockerspringboot

上传编译构建好的jar包 EurekaServer01-0.0.1-SNAPSHOT.jar到该目录

### 第三步：在/usr/local/dockerspringboot目录下创建文件Dockerfile

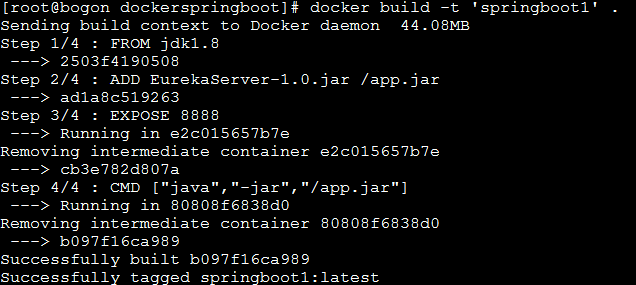
vi Dockerfile

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  ADD EurekaServer01-0.0.1-SNAPSHOT.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  # 配置容器启动后执行的命令  CMD ["java","-jar","/app.jar"] |

### 第四步：执行命令构建镜像

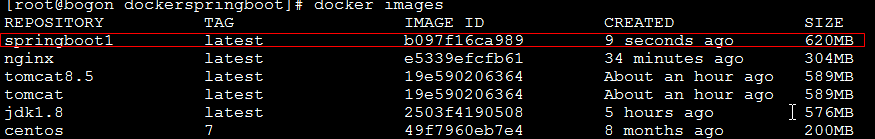
docker build -t='springboot1' .

注意后边的空格和点（.表示当前目录），不要省略



### 第五步：查看镜像是否建立完成

docker images

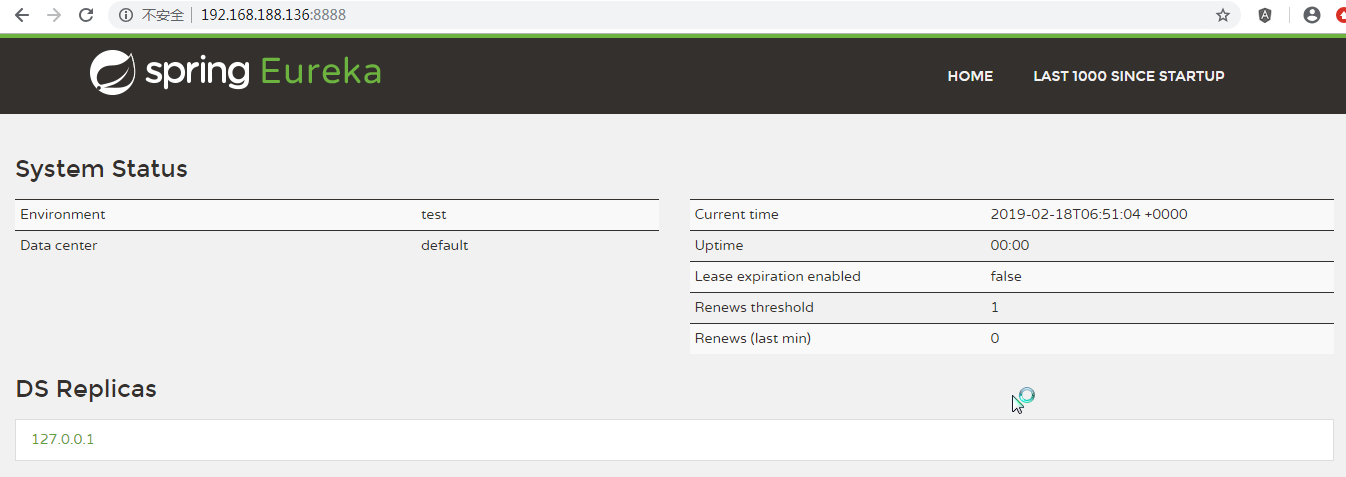


### 第六步：使用镜像创建运行容器

docker run -di --name=s1 -p 8888:8888 springboot1

### 第七步：访问测试

<http://ip:8888>



# 使用Nexus搭建Docker镜像私有仓库

Docker镜像创建好了，怎么样实现在不同的用户和服务器之间共享镜像，Docker官方提供了DockerHub可以允许用户把自己创作的镜像上传到服务器，但是对于包含企业私有应用的镜像不允许暴露给公共用户，这个时候就需要企业搭建自己的私有镜像存储仓库。

nexus是一款非常优秀的私有仓库搭建软件，可以很方便的搭建私有镜像仓库。

下载地址：<https://www.sonatype.com/download-oss-sonatype>



## 8.1、安装Nexus

### 1、安装nexus需要首先安装jdk环境

java -version

### 2、上传nexus安装包到服务器

nexus-3.15.2-01-unix.tar.gz

### 3、创建目录、解压缩nexus

mkdir /usr/local/nexus

tar xvf nexus-3.15.2-01-unix.tar.gz -C /usr/local/nexus

### 4、启动nexus

cd /usr/local/nexus/nexus-3.15.2-01/bin

./nexus start

### 5、启动后稍等大于1分钟，查看端口监听

netstat -antp|grep 8081

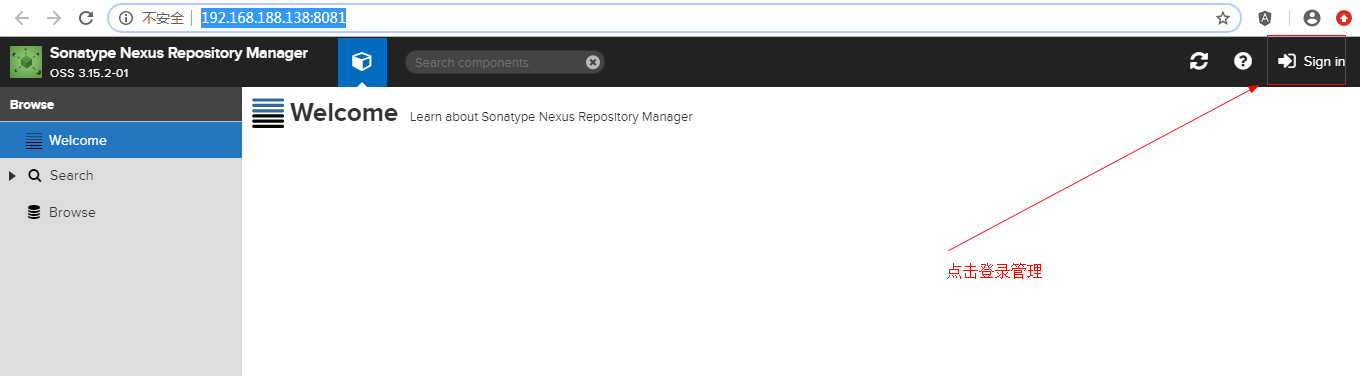
### 6、开放防火墙端口

firewall-cmd --add-port=8081/tcp --permanent

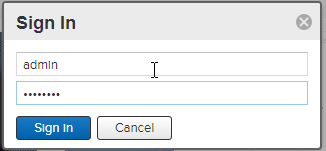
firewall-cmd --reload

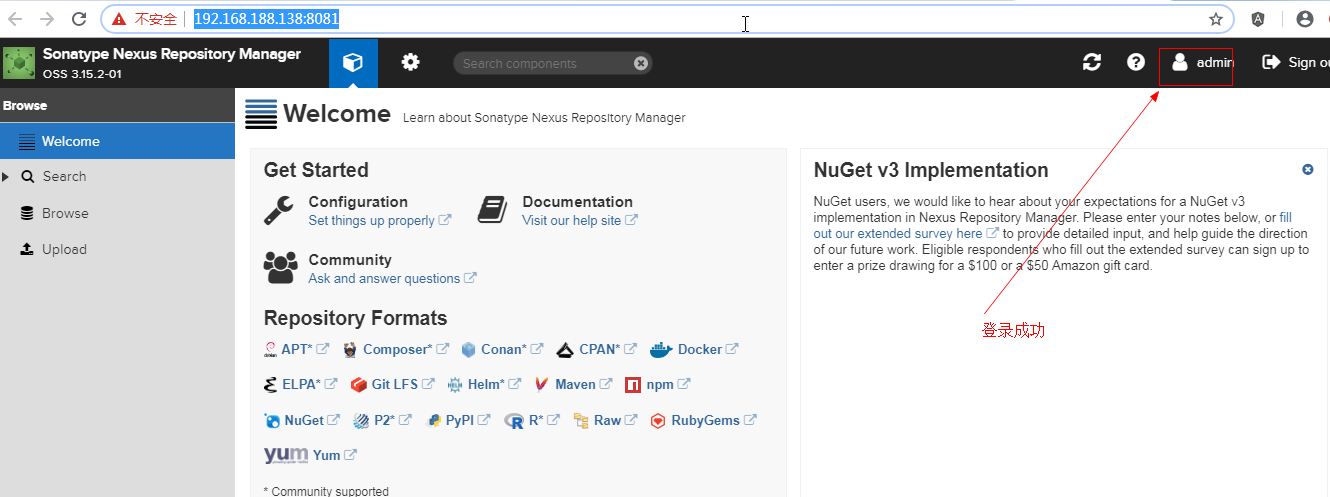
## 8.2、访问配置Nexus

### 1、访问地址http://ip:8081



出现登陆框，输入账号 admin 密码 admin123





登录成功

### 配置nexus创建Docker私有仓库

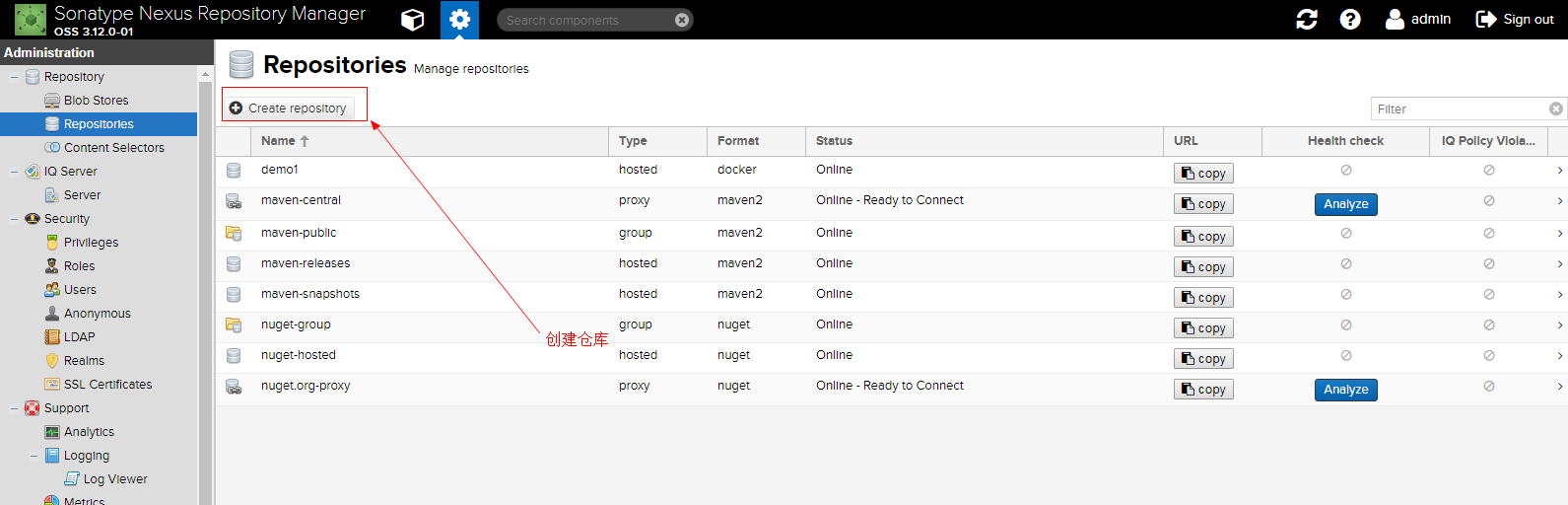
进入系统设置界面



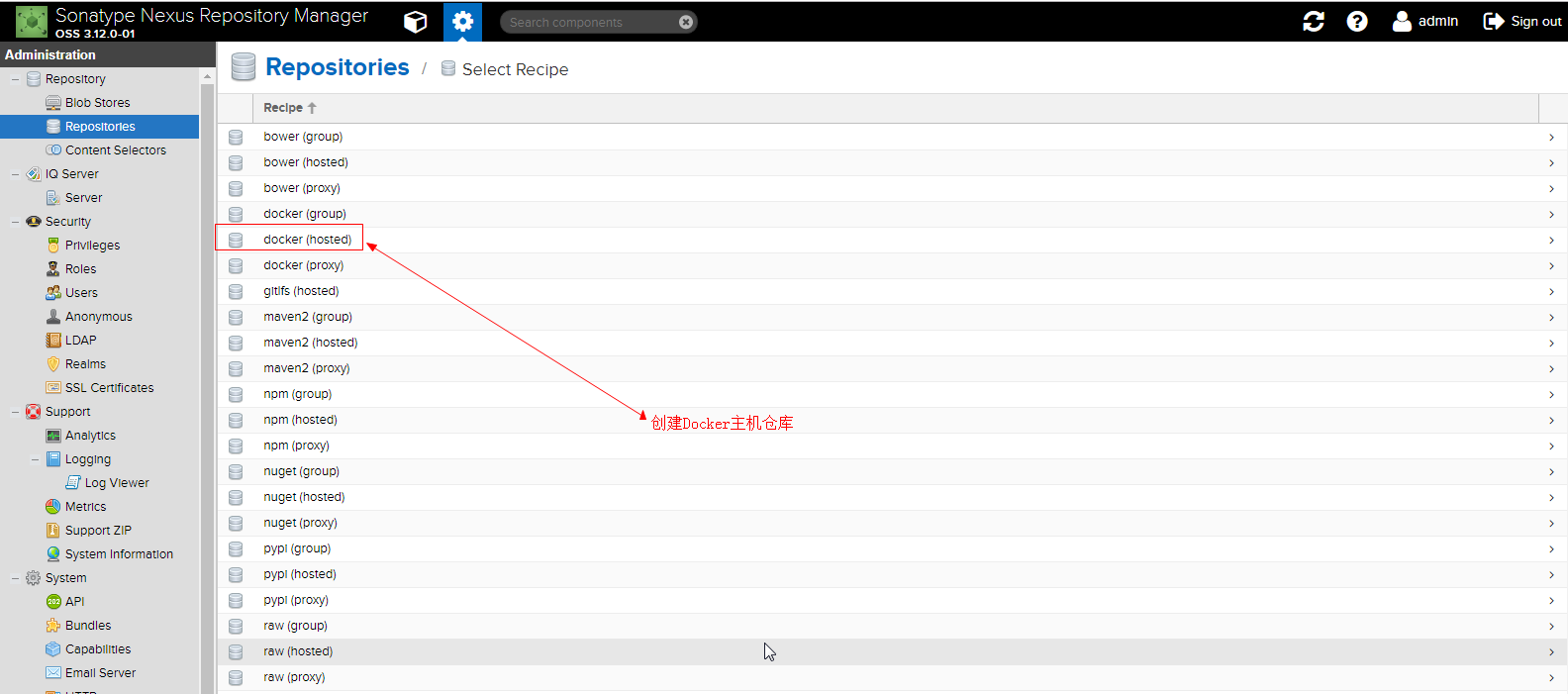
点击Repositories



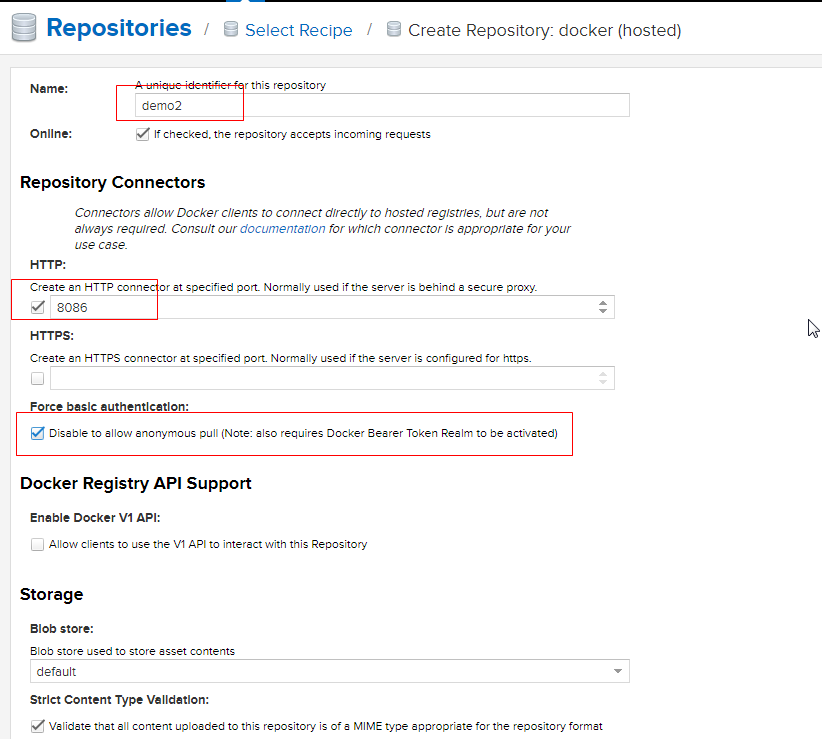
点击创建仓库（Create Repository）



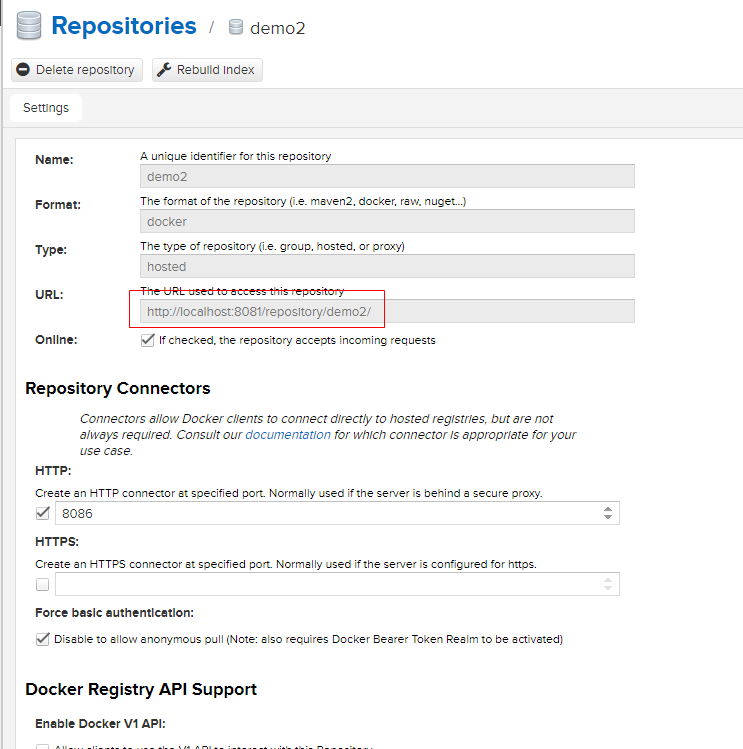
选择要创建的仓库类型（docker(hosted)）



配置docker仓库信息



查看仓库详情



**仓库地址：192.168.0.109:8086**

**仓库账号: admin**

**密码: admin123**

## 8.3、配置Docker服务器连接到Nexus私有仓库

### 1、修改 /etc/docker/daemon.json 配置docker私有仓库

编辑：/etc/docker/daemon.json

|  |
| --- |
| {  "registry-mirrors": ["https://w5z91a3d.mirror.aliyuncs.com"],  "insecure-registries":["192.168.0.109:8086"]  } |

### 2、刷新守护进程、重启dcker

systemctl daemon-reload

systemctl restart docker

## 8.4、连接docker私服镜像仓库及使用

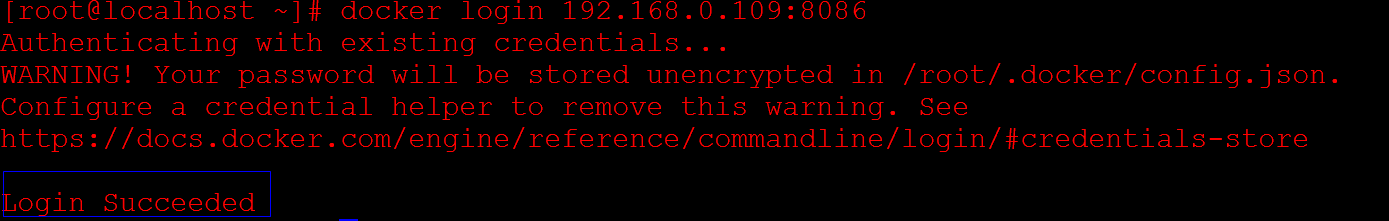
### 1、登录私服镜像仓库

docker login 192.168.0.109:8086

输入账号:admin

密码:admin123

提示，Login Succeeded 表示登录成功



### 2、为镜像打一个tag标签

在上传镜像之前需要先打一个tag，用于版本标记。

格式是这样的：

docker tag <imageId or imageName> <nexus-hostname>:<repository-port>/<image>:<tag>

例如：

docker tag tomcat8.5 192.168.0.109:8086/tomcat:8.5

### 3、上传镜像

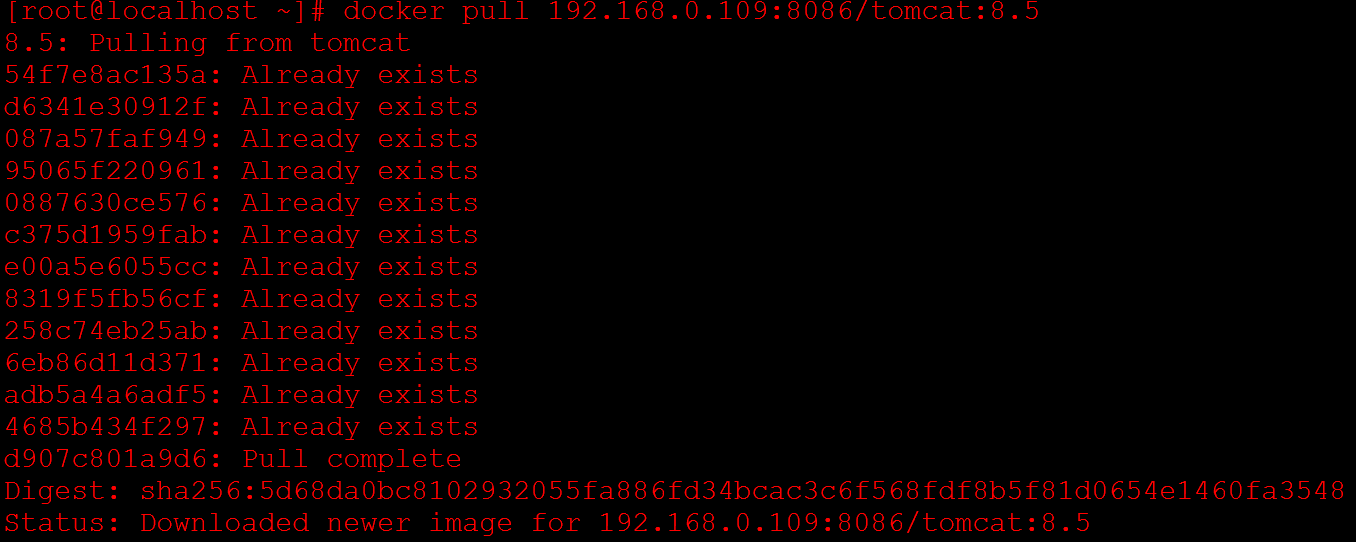
docker push 192.168.0.109:8086/tomcat:8.5

### 4、下载镜像

从私服中下载镜像也很简单，执行以下命令即可

docker pull 192.168.0.109:8086/tomcat:8.5

注意：确保docke服务器和私服在同一网段、或者能够进行通信。

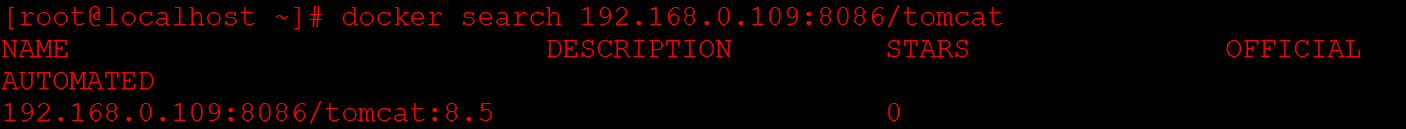


### 5、搜索镜像

搜索镜像也与之前的类似，如下：

docker search 192.168.0.109:8086/tomcat

就可以搜索带有tomcat的镜像了：



# Maven Docker插件构建Docker镜像

Maven是一个强大的项目管理与构建工具。可以使用Maven构建Docker镜像，那么我们的工作能得到进一步的简化。

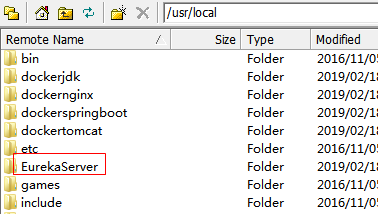
插件地址：<https://github.com/spotify/docker-maven-plugin>

## 9.1、maven Docker插件构建Docker镜像入门

### 第一步：修改项目EurekaServer的pom.xml

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>offcn/microservice-discovery-eureka:0.0.1</imageName>  <baseImage>java: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>  </configuration>  </plugin>  </plugins>  </build> |

### 第二步：把项目文件上传到linux宿主机目录/usr/local



### 第三步：下载并上传jdk、maven的安装包到linux服务器

jdk-8u131-linux-x64.tar.gz

apache-maven-3.6.0-bin.tar.gz

### 第四步：安装jdk

（1）、tar xvf jdk-8u131-linux-x64.tar.gz -C /usr/local/

（2）、cd /usr/local

（3）、mv jdk1.8.0\_131 jdk

（4）、vi /etc/profile

#在最后添加

export JAVA\_HOME=/usr/local/jdk

export CLASSPATH=$JAVA\_HOME/lib:.

export PATH=$JAVA\_HOME/bin:$PATH

#重新加载配置文件

1. 、source /etc/profile
2. 测试jdk是否安装成

java -version

### 第五步：安装maven

（1）、tar xvf apache-maven-3.6.0-bin.tar.gz -C /usr/local/

（2）、cd /usr/local

（3）、mv apache-maven-3.6.0 maven

（4）、vi /etc/profile

#在最后添加

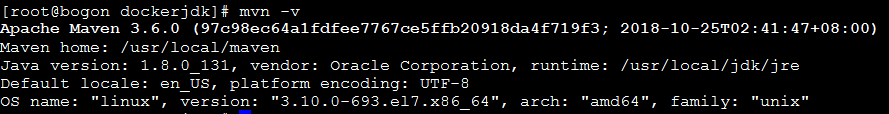
export MAVEN\_HOME=/usr/local/maven

export PATH=$MAVEN\_HOME/bin:$PATH

#重新加载配置文件

1. 、source /etc/profile
2. 测试maven安装

执行命令: mvn -v



### 第六步：编译构建项目

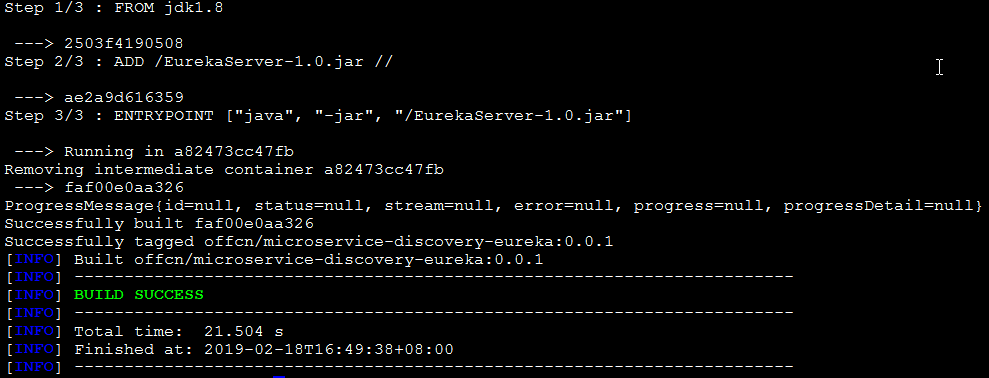
切换到测试JAVA项目目录

cd /usr/local/EurekaServer

执行编译指令：

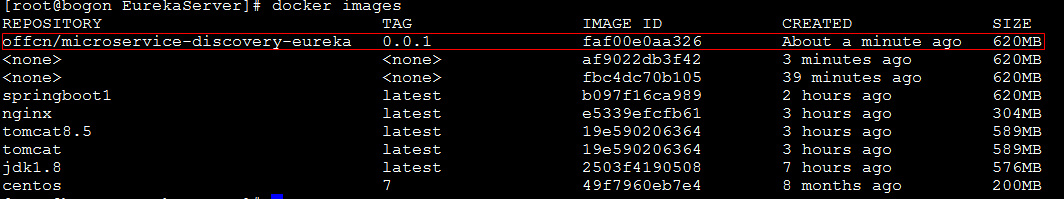
mvn clean -Dmaven.test.skip=true package docker:build

我们会发现终端输出类似于如下的内容：



### 第七步：查看镜像

执行命令: docker images

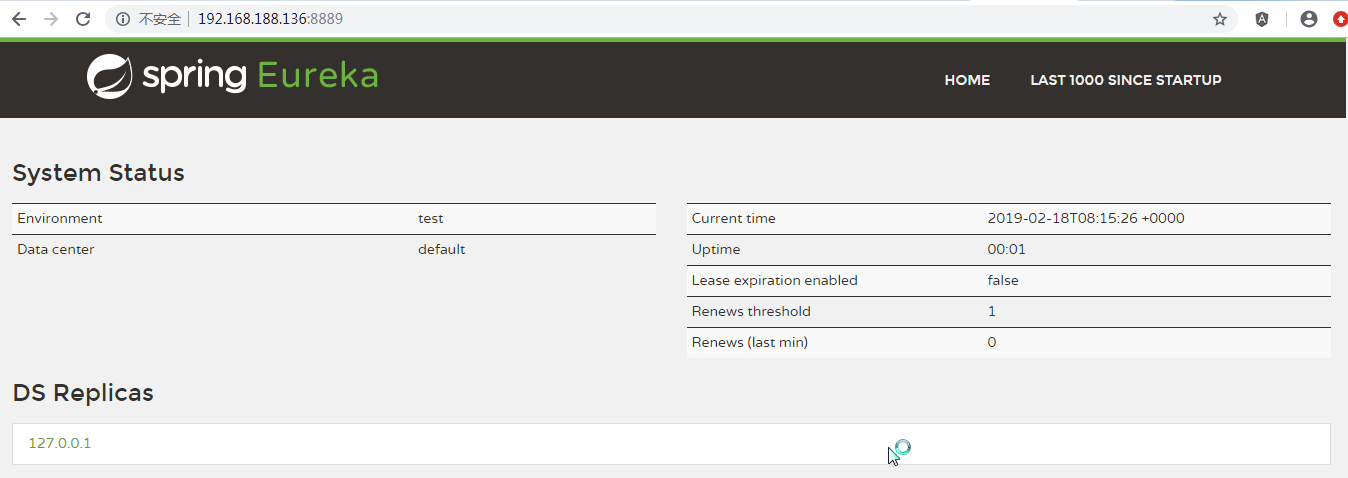


### 第八步：使用镜像创建运行容器

docker run -di --name=e2 -p 8889:8888 offcn/microservice-discovery-eureka:0.0.1

### 第九步：访问测试

<http://ip:8889>



## 9.2、maven Docker插件构建Docker镜像，自定义DockerFile

之前的示例，我们直接在pom.xml中设置了一些构建的参数。很多场景下，我们希望使用Dockerfile更精确、有可读性地构建镜像。

下面我们带领大家看一下如何使用Dockerfile基于Maven插件构建镜像。

### 第一步：首先我们在项目的src/main/docker 目录下，新建一个Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  ADD EurekaServer01-0.0.1-SNAPSHOT.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  # 配置容器启动后执行的命令  CMD ["java","-jar","/app.jar"] |

### 第二步：修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>offcn/microservice-discovery-eureka:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  </configuration>  </plugin>  </plugins>  </build> |

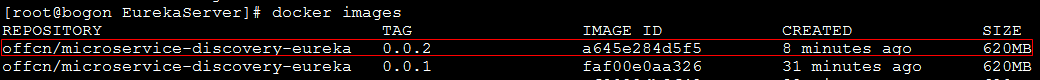
### 第三步：上传到linux服务器重新执行编译构建，即可根据Dockerfile生成镜像

切换到测试JAVA项目目录

cd /usr/local/EurekaServer

执行编译构建指令：

mvn clean -Dmaven.test.skip=true package docker:build



## 9.3、maven Docker插件构建并推送镜像到Docker私有仓库

前文我们使用docker push 命令实现了镜像的推送，我们也可使用Maven插件推送镜像。我们不妨使用Maven插件推送一个Docker镜像到Docker 私有仓库。

### 第一步：修改Maven的全局配置文件setttings.xml

在其中添加以下内容，配置Docker私有仓库的用户信息。

|  |
| --- |
| <server>  <id>docker-hub</id>  <username>admin</username>  <password>admin123</password>  <configuration>  <email>hk109@139.com</email>  </configuration>  </server> |

### 第二步：修改项目EurekaServer01的pom.xml

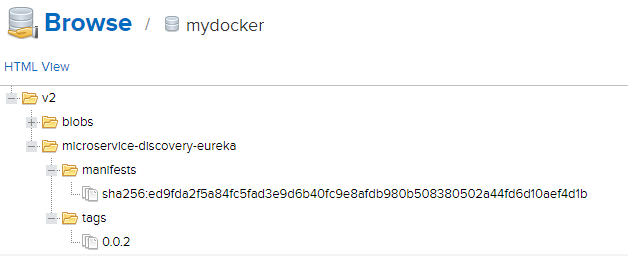
|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.1:8086/microservice-discovery-eureka:0.0.2</imageName>  <dockerDirectory>${project.basedir}/src/main/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

serverId:必须和maven配置文件设置的server 节点的id一致

### 第三步：执行以下命令，添加pushImage的标识，表示推送镜像

mvn clean -Dmaven.test.skip=true -DpushImage package docker:build

执行完毕，查看Docker私服已经上传成功镜像



# 手动发布部署微服务项目到Docker

## 10.1、项目工程发布镜像规划

按照项目EurekaServer01的配置方式把相关项目都发布成镜像

|  |  |  |
| --- | --- | --- |
| 序号 | 项目名称 | 镜像名称 |
| 1 | EurekaServer01 | 192.168.188.138:8086/microservice-discovery-eureka:0.0.1 |
| 2 | ConfigServer001 | 192.168.188.138:8086/microservice-config-server:0.0.1 |
| 3 | ZuulGateWay | 192.168.188.138:8086/microservice-zuul-gateway:0.0.1 |
| 4 | UserProvdier01 | 192.168.188.138:8086/microservice-provdier-service:0.0.1 |
| 5 | UserWeb06 | 192.168.188.138:8086/microservice-user-web:0.0.1 |

注意因为EurekaServer01和EurekaServer02只是端口和配置不同，没必要单独创建2个独立镜像，仅仅需要在Docker镜像创建容器的时候传递相关参数即可。

同理ConfigServer001、ConfigServer002也公用一个镜像。

UserProvdier01、UserProvdier02也公用一个镜像。

## 10.2、项目工程发布镜像配置

### 10.2.1、项目EurekaServer01

一、在项目根目录创建docker目录，创建Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  COPY EurekaServer01-\*.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  #配置环境变量，接受外部传递参数  #启动端口  ENV PORT=""  #注册中心地址  ENV EUREKA=""  # 配置容器启动后执行的命令  ENTRYPOINT ["sh","-c","java $PORT $EUREKA -jar /app.jar"] |

二、修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.138:8086/microservice-discovery-eureka:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

### 10.2.2、项目ConfigServer001

一、在项目根目录创建docker目录，创建Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  COPY ConfigServer001-\*.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  #配置环境变量，接受外部传递参数  #启动端口  ENV PORT=""  #注册中心地址  ENV EUREKA=""  # 配置容器启动后执行的命令  ENTRYPOINT ["sh","-c","java $PORT $EUREKA -jar /app.jar"] |

二、修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.138:8086/microservice-config-server:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

### 10.2.3、项目ZuulGateWay

一、在项目根目录创建docker目录，创建Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  COPY ZuulGateWay-\*.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  #配置环境变量，接受外部传递参数  #启动端口  ENV PORT=""  #注册中心地址  ENV EUREKA=""  # 配置容器启动后执行的命令  ENTRYPOINT ["sh","-c","java $PORT $EUREKA -jar /app.jar"] |

二、修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.138:8086/microservice-zuul-gateway:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

### 10.2.4、项目UserProvdier01

一、在项目根目录创建docker目录，创建Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  COPY UserProvdier01-\*.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  #配置环境变量，接受外部传递参数  #监听端口  ENV PORT=""  #应用名称，用来前端显示用，不是serviceId  ENV APPLICATION=""  #注册中心地址  ENV EUREKA=""  # 配置容器启动后执行的命令  ENTRYPOINT ["sh","-c","java $PORT $APPLICATION $EUREKA -jar /app.jar"] |

二、修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.138:8086/microservice-provdier-service:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

1. bootstrap.yml配置文件内容

|  |
| --- |
| **spring**:  **cloud**:  **config**:  **discovery**:  **enabled**: **true  service-id**: CONFIG-SERVER  **profile**: dev  **label**: master |

1. 项目在git存储的配置文件

|  |
| --- |
| ProviderVersion: UserProvider0.0.1  spring:  datasource:  url: jdbc:mysql://db1:3306/springboot-005?serverTimezone=GMT%2B8  username: root  password: 123  driver-class-name: com.mysql.cj.jdbc.Driver  jpa:  hibernate:  ddl-auto: update  show-sql: true  application:  name: UserProvider  management:  endpoints:  web:  exposure:  include: refresh,health,info |

注意：配置文件级别 本地配置文件---》-D参数注入配置文件----》远程配置中心配置文件，级别越来越高。

### 10.2.5、项目UserWeb06

一、在项目根目录创建docker目录，创建Dockerfile文件

|  |
| --- |
| # 基于哪个镜像  FROM java:8  # 拷贝文件到容器，也可以直接写成ADD xxxxx.jar /app.jar  COPY UserWeb06-\*.jar /app.jar  # 声明需要暴露的端口  EXPOSE 8888  #配置环境变量，接受外部传递参数  #监听端口  ENV PORT=""  #应用名称  ENV APPLICATION=""  #注册中心服务器地址  ENV EUREKA=""  # 配置容器启动后执行的命令  ENTRYPOINT ["sh","-c","java $PORT $APPLICATION $EUREKA -jar /app.jar"] |

二、修改pom.xml,修改maven Docker插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>0.4.13</version>  <configuration>  <imageName>192.168.188.138:8086/microservice-user-web:0.0.1</imageName>  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <resources>  <resource>  <targetPath>/</targetPath>  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  </resources>  <serverId>docker-hub</serverId>  </configuration>  </plugin>  </plugins>  </build> |

三、bootstrap.yml配置文件内容

|  |
| --- |
| **spring**:  **application**:  **name**: UserWeb06  **cloud**:  **config**:  **discovery**:  **enabled**: **true  service-id**: CONFIG-SERVER  **profile**: dev  **label**: master |

## 10.3、编译工程创建镜像

一、上传项目EurekaServer01到Linux服务器目录：/usr/local/springboot/EurekaServer01

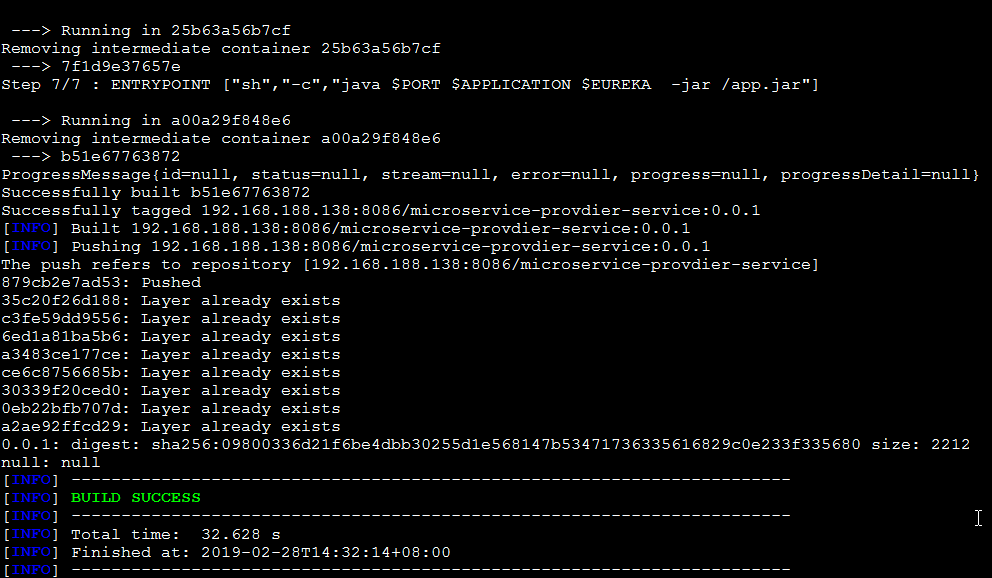
二、执行编译、打包、构建docker镜像指令

进入当前项目目录

cd /usr/local/springboot/EurekaServer01

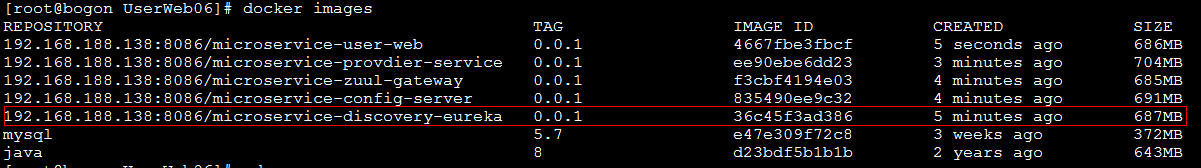
编译、打包、发布Docker镜像指令：

|  |
| --- |
| mvn clean -Dmaven.test.skip=true -DpushImage package docker:build |



查看镜像：

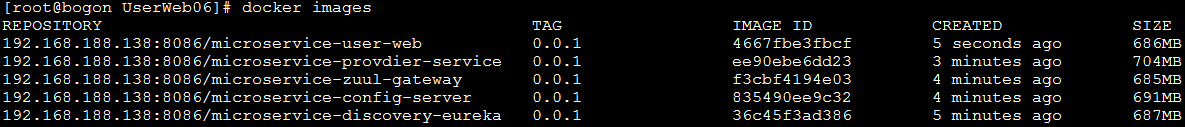
docker images



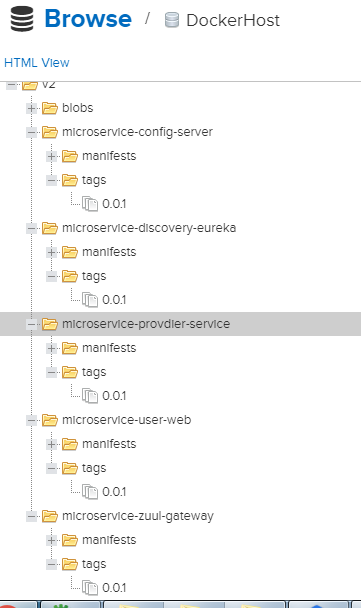
三、其他项目

其他项目ConfigServer001、ZuulGateWay、UserProvdier01、UserWeb06也和上面一样，上传，然后进入项目目录执行编译、打包、构建Docker镜像指令。

最终5个镜像全部创建ok

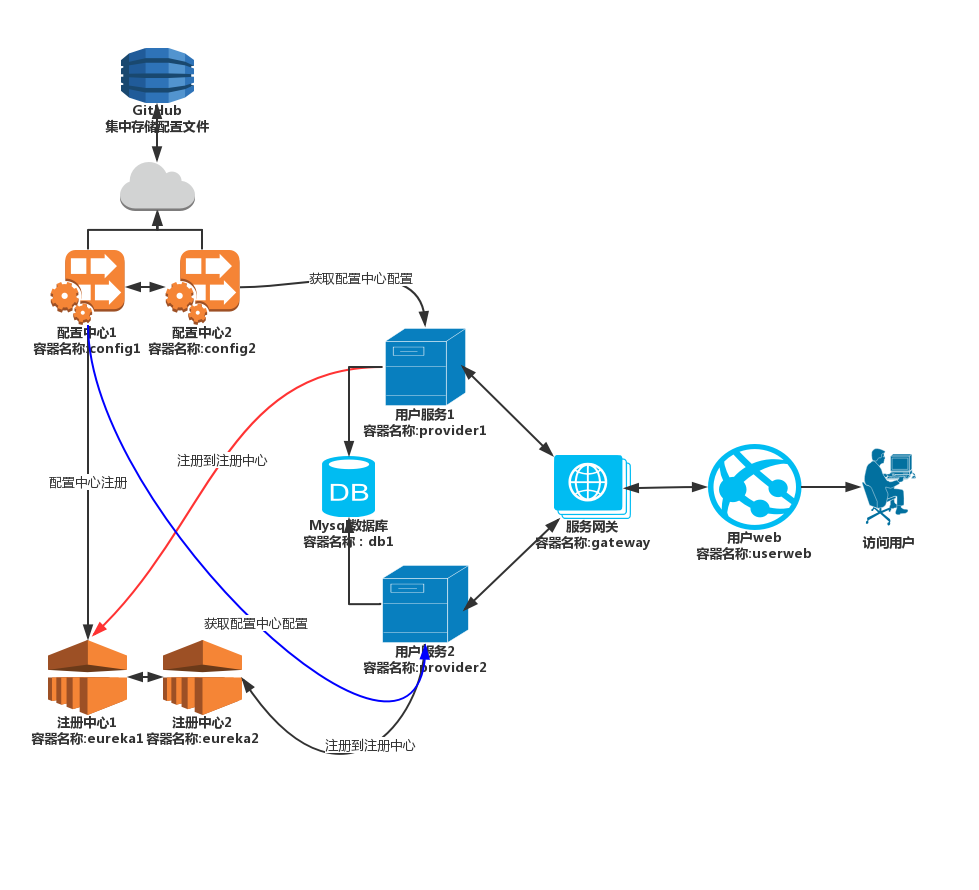


在私有仓库可以查看到镜像



## 10.5、搭建实际运行项目

### 一：项目架构图



### 二：创建docker网络

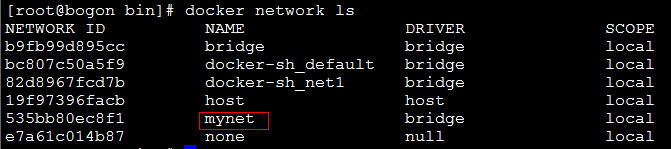
根据上面的架构图，本次需要创建一个用户自定义网络：

Docker指令:

|  |
| --- |
| docker network create mynet |

查看docker网络指令：

|  |
| --- |
| docker network ls |



构建网络成功后，所有加入到该用户自定义网络的容器即可通过容器名称相互连接。

### 三：创建运行docker容器

|  |  |  |
| --- | --- | --- |
| **容器名称** | **类型** | **对应镜像** |
| eureka1 | 注册中心 | 192.168.188.138:8086/microservice-discovery-eureka:0.0.1 |
| **容器创建指令：**  docker run #创建容器指令  -di # d守护式容器、i 运行容器  --name=eureka1 # 容器名称  --network=mynet #容器加入用户自定义网络mynet  -e PORT="-Dserver.port=10086" #指定环境参数--Eureka注册中心服务启动监听端口  #指定环境参数--Eureka注册中心地址（高可用架构）  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka2:10087/eureka"  -p 10086:10086 #容器端口号映射  192.168.188.138:8086/microservice-discovery-eureka:0.0.1 #镜像名称 | | |
| eureka2 | 注册中心 | 192.168.188.138:8086/microservice-discovery-eureka:0.0.1 |
| **容器创建指令：**  docker run #创建容器指令  -di # d守护式容器、i 运行容器  --name=eureka2 # 容器名称  --network=mynet #容器加入用户自定义网络mynet  -e PORT="-Dserver.port=10087" #指定环境参数--Eureka注册中心服务启动监听端口  #指定环境参数--Eureka注册中心地址（高可用架构，互相指向对方）  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka"  -p 10087:10087 #容器端口号映射  192.168.188.138:8086/microservice-discovery-eureka:0.0.1 #镜像名称 | | |
| config1 | 配置中心 | 192.168.188.138:8086/microservice-config-server:0.0.1 |
| **容器创建指令：**  docker run  -di  --name=config1  --network=mynet  #指定环境参数--配置中心服务启动监听端口  -e PORT="-Dserver.port=7001"  #指定Eureka注册中心地址（高可用架构）  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 7001:7001  192.168.188.138:8086/microservice-config-server:0.0.1 | | |
| config2 | 配置中心 | 192.168.188.138:8086/microservice-config-server:0.0.1 |
| **容器创建指令：**  docker run  -di  --name=config2  --network=mynet  #指定环境参数--配置中心服务启动监听端口  -e PORT="-Dserver.port=7002"  #指定Eureka注册中心地址（高可用架构）  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 7002:7002  192.168.188.138:8086/microservice-config-server:0.0.1 | | |
| db1 | mysql数据库 | mysql:5.7 |
| **容器创建指令：注意创建容器后需要修改配置文件解决中文乱码问题**  docker run  -di  --name=db1  --network=mynet  -p 3306:3306  #mysq数据库密码  -e MYSQL\_ROOT\_PASSWORD=123456  #初始化mysql数据库  -e MYSQL\_DATABASE=jpa001  #镜像名称  mysql:5.7 | | |
| gateway | 服务网关 | 192.168.188.138:8086/microservice-zuul-gateway:0.0.1 |
| **容器创建指令：**  docker run  -di  --name=gateway  --network=mynet  -e PORT="-Dserver.port=80"  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 80:80  192.168.188.138:8086/microservice-zuul-gateway:0.0.1 | | |
| provider1 | 服务提供方 | 192.168.188.138:8086/microservice-provdier-service:0.0.1 |
| **容器创建指令：**  docker run -di  --name=provider1 --network=mynet  -e PORT="-Dserver.port=8001" #监听端口  -e APPLICATION="-Dspring.application.name=UserProvider01" #应用名称用于github查找对应配置文件  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 8001:8001  192.168.188.138:8086/microservice-provdier-service:0.0.1 | | |
| **Github对应配置文件UserProvdier01-test.properties内容：**  #服务器端口号  #server.port=8001  #应用名称  spring.application.name=UserProvider  #配置显示给消费者的服务器端标识信息  ProviderVersion=UserProvider01  #数据库jdbc连接url地址,serverTimezone设置数据库时区东八区  spring.datasource.url=jdbc:mysql://db1:3306/springcloud01?serverTimezone=GMT%2B8  #数据库账号  spring.datasource.username=root  #数据库密码  spring.datasource.password=123456  #指定数据库驱动  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  #第一次加载JPA时根据model类会自动建立起表的结构（前提是先建立好数据库  #以后加载JPA时根据 model类自动更新表结构  spring.jpa.hibernate.ddl-auto=update  #显示JPA执行的sql语句  spring.jpa.show-sql=true | | |
| provider2 | 服务提供方 | 192.168.188.138:8086/microservice-provdier-service:0.0.1 |
| **容器创建指令：**  docker run -di --name=provider2 --network=mynet  -e PORT="-Dserver.port=8002" #监听端口  #应用名称用于github查找对应配置文件  -e APPLICATION="-Dspring.application.name=UserProvider02"  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 8002:8002  192.168.188.138:8086/microservice-provdier-service:0.0.1 | | |
| **Github对应配置文件UserProvdier02-test.properties内容：**  #服务器端口号  #server.port=8002  #应用名称  spring.application.name=UserProvider  #配置显示给消费者的服务器端标识信息  ProviderVersion=UserProvider02  #数据库jdbc连接url地址,serverTimezone设置数据库时区东八区  spring.datasource.url=jdbc:mysql://db1:3306/springcloud01?serverTimezone=GMT%2B8  #数据库账号  spring.datasource.username=root  #数据库密码  spring.datasource.password=123456  #指定数据库驱动  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  #第一次加载JPA时根据model类会自动建立起表的结构（前提是先建立好数据库  #以后加载JPA时根据 model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行  spring.jpa.hibernate.ddl-auto=update  #显示JPA执行的sql语句  spring.jpa.show-sql=true | | |
| userweb | 服务消费者 | 192.168.188.138:8086/microservice-user-web:0.0.1 |
| docker run -di --name=userweb --network=mynet  -e PORT="-Dserver.port=9001"  -e APPLICATION="-Dspring.application.name=UserWeb06" #应用名称用于github查找对应配置文件  -e EUREKA="-Deureka.client.service-url.defaultZone=http://eureka1:10086/eureka,http://eureka2:10087/eureka"  -p 9001:9001  192.168.188.138:8086/microservice-user-web:0.0.1 | | |

### 四：查看docker容器输出日志

通过指令：docker logs -f -t --tail 10 provider1

就可以查看对应的容器日志输出

### 五：测试运行

访问:http://192.168.188.138:9001

