# Docker

**学习目标**

* 掌握DockerMavn插件的使 用？
* 掌握持续集成工具Jenkins的安装与使用？
* 掌握容器管理工具Rancher的安装与使用？
* 掌握时间序列数据库influxDB的安装与使用？
* 掌握容器出监控工具cAdvisor的安装与使用
* Dockerfile编写创建容器的脚本？
* 掌握图表工具Grafana的使用？

# 1 DockerMavn插件？

**微服务部署有两种方法:**

(1)手动部署:首先基于源码打包生成jar包(或war包) ,将jar包 (或war包)，上传至虚拟机并拷贝至JDK容器。

(2)通过Maven插件自动部署。

对于数量众多的微服务，手动部署无疑是非常麻烦的做法，并且容易出错。所以我们这里学习如何自动部署，这也是企业实际开发中经常使用的方法。

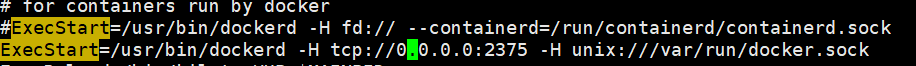
**Maven插件自动部署步骤:**

(1)修改宿主机的docker配置，让其可以远程访问

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

修改后如下:



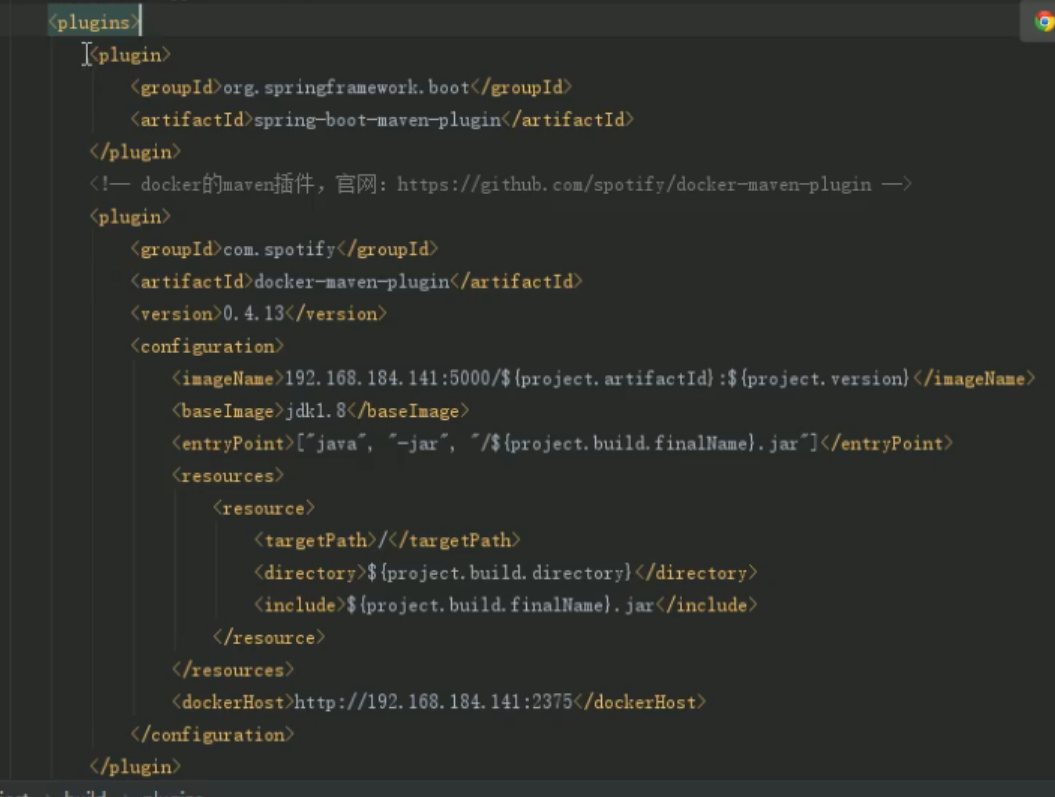
(2)刷新配置，重启服务

systemctl daemon-reload

systemctl restart docker

docker start registry

(3)在工程pom.xml增加配置



# 2 Jenkins

## 2.1 是什么？

持续集成Continuous integration,简称**CI**

随着软件开发复杂度的不断提高，团队开发成员间如何更好地协同工作以确保软件开发的质量已经慢慢成为开发过程中不可回避的问题。尤其是近些年来，敏捷(Agile)在软件工程领域越来越红火，如何能在不断变化的需求中快速适应和保证软件的质量也显得尤其的重要。

持续集成正是针对这一类问题的一 种软件开发实践。它倡导团队开发成员必须经常集成他们的工作，甚至每天都可能发生多次集成。而每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件。

**持续集成具有的特点:**

●它是一个自动化的周期性的集成测试过程，从检出代码、编译构建、运行测试、结果记录、测试统计等都是自动完成的，无需人工干预;

●需要有专门的集成服务器来执行集成构建;

●需要有代码托管工具支持,我们下一小节将介绍Git以及可视化界面Gogs的使用持续集成的作用:

**持续集成的作用:**

●保证团队开发人员提交代码的质量，诚轻了软件发布时的压力;

●持续集成中的任何-一个环节都是自动完成的，无需太多的人工干预，有利于减少重复过程以节省时间、费用和工作量;

## 2.2 简介

Jenkins,原名Hudson, 2011年改为现在的名字，它是一个开源的实现持续集成的软件工具。官方网站: http://jenkins-ci.org/.

Jenkins能实施监控集成中存在的错误，提供详细的日志文件和提醒功能，还能用图表的形式形象地展示项目构建的趋势和稳定性。

**特点:**

●易安装:仅仅一个java -jar jenkins.war, 从官网下载该文件后，直接运行，无需额外的安装，更无需安装数据库;

●易配置:提供友好的GU配置界面;

●变更支持: Jenkins能从代码仓库(Subversion/CVS) 中获取并产生代码更新列表并输出到编译输出信息中;

●支持永久链接:用户是通过web来访问Jenkins的，而这些web页面的链接地址都是永久链接地址，因此，你可以在各种文档中直接使用该链接;

●集成E-Mail/RSS/IM: 当完成一次集成时，可通过这些工具实时告诉你集成结果(据我所知，构建一次集成需要花费一定时间，有了这个功能，你就可以在等待结果过程中，干别的事情) ;

●支持第三方插件:使得Jenkins变得越来越强大

## 2.3 安装

### 2.3.1 JDK安装

(1)将jdk-8u171-linux-x64.rpm. 上传至服务器( 虚拟机)

(2)执行安装命令

**rpm -ivh jdk-8u171-linux-x64.rpm**

RPM方式安装JDK,其根目录为: /usr/java/jdk1.8.0 \_171t

### 2**.3.2 Jenkins安装与启动**

(1)下载jenkins

wget https://pkg. jenkins . io/redhat/jenkins-2.83-1.1.noarch.rpm

或将jenkins-2.83-1.1.noarch.rpm.上传至服务器

(2)安装jenkins

**rpm -ivh jenkins-2.89.3-1.1.noarch.rpm**

(3)配置jenkins

vi /etc/sysconfig/jenkins

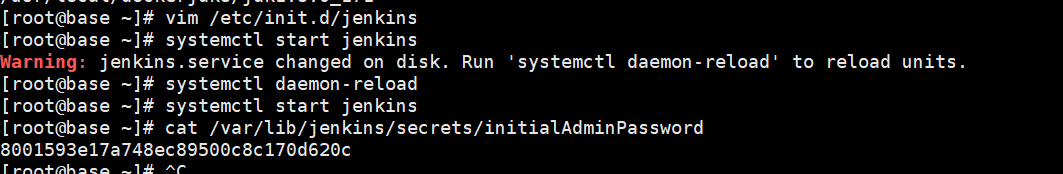
修改用，户和端口

JENKINS\_ USER="root"

JENKINS\_ PORT="8888"

(4)启动服务

systemctl start jenkins



8001593e17a748ec89500c8c170d620c

/usr/lib/jenkins/jenkins.war WAR包

/etc/sysconfig/jenkins 配置文件，“端口”，“JENKINS\_HOME”等都可以在这里配置。

/var/lib/jenkins/ 默认的JENKINS\_HOME目录

/var/log/jenkins/jenkins.log Jenkins日志文件

## 2.4 Jenkins插件安装

## 2.5 全局工具配置

## 2.6 代码上产到Git服务器

## 2.7 任务的创建与执行

# 3 常用命令

## 3.1 镜像相关命令

### 3.1.1 查详情

docker images

**REPOSITORY:** 镜像名称（可能重名）

**TAG:** 镜像标签（版本区分）

**IMAGE ID:** 镜像ID（唯一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

### 3.1.2 搜索

如果你需要从网络中查找需要的镜像，可以通过以下命令搜索

docker search 镜像名称

**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

### 3.1.3 拉取

拉取镜像就是从中央仓库中下载镜像到本地

docker pull 镜像名称

例如，我要下载centos7镜像

docker pull centos:7

### 3.1.4 删

按镜像ID删除镜像（镜像没有生成容器，不管是运行的还是停止的）

docker rmi镜像ID

删除所有镜像

docker rmi `docker images -q`

## 3.2 容器相关命令

### 3.2.1 查详情

查看正在运行的容器

docker ps

查看所有容器

docker ps -a

查看最后一次运行的容器

docker ps - l

查看停止的容器

docker ps -f status=exited

### 3.2.2 创建与启动

创建容器常用的参数说明:

创建容器命令:

docker run

**-i:**表示运行容器

**-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。

**--name :**为创建的容器命名。(--两个的一般要指定值)

**-V:** 表示目录映射关系(前者是宿主机目录，后者是映射到宿主机上的目录)，可以使用多个-v做多个目录或文件映射。注意:最好做目录映射，在宿主机上做修改，然后共享到容器上。

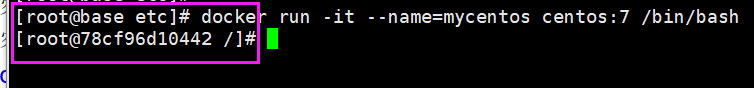
**-d;**在run后面加上-d参数,则会创建一个守护式容器在后台运行(这样创建容器后不会自动登录容器，如果只加 -t两个参数，创建后就会自动进去容器)。

**-p:** 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

**(1)交互式方式创建容器**

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

docker run -it --name=mycentos centos:7 /bin/bash



这时我们通过ps命令查看，发现可以看到启动的容器，状态为启动状态

退出当前容器(也就关闭了)

exit

**(2)守护式方式创建容器:**

docker run -di --name =容器名称 镜像名称:标签

docker run -id --name=mycentos1 centos:7

登录守护式容器方式:

docker exec -ti容器名称 /bin/bash

docker exec -it mycentos1 /bin/bash

退出当前容器(不关闭)

exit

### 3.2.3 停止与启动容器

停止容器:

docker stop 容器名称(或者容器ID)

启动容器:

docker start 容器名称(或者容器ID)

### 3.2.4 文件拷贝

如果我们需要将文件拷贝到容器内可以使用cp命令

docker cp 需要拷贝的文件或目录 容器名称:容器目录

也可以将文件从容器内拷贝出来

docker cp容器名称:容器目录 需要拷贝的文件或目录

### 3.2.5 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。

创建容器添加-v参数后边为宿主机目录:容器目录，例如:

docker run -di -v /usr/local/ myhtml:/usr/local/myhtml --name mmycentos3 centos:7

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 来解诀挂载的目录没有权限的问题

### 3.2.6 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

docker inspect 容器名称(容器ID)

也可以直接执行下面的命令直接输出IP地址

docker inspect --format='{{.NetworkSettings.IPAddress}}' 容器名称(容器ID)

docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2

### 3.2.7 删

删除指定的容器(必须是停止的):

docker rm容器名称(容器ID)

# 4 应用部署

## 4.1 MySQL

**拉取镜像:**

docker pull centos/mysql-57-centos7

5.7版本

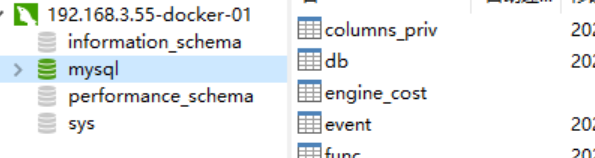
**创建容器**

docker run -di --name=tensquare\_mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 centos/mysql-57-centos7

-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

**远程登录mysql**



## 4.2 tomcat

**拉取镜像：**

docker pull tomcat:7-jre8

**创建容器：**

docker run -di --name=mytomcat -p 9000:8080 -v /usr/local/webapps:/usr/local/tomcat/webapps tomcat:7-jre8

这里用root包测试，会有一些BUG，暂时不知道原因

/usr/local/tomcat/webapps

/usr/local/tomcat/webapps

**部署项目：**

## 4.3 Nginx

**拉取镜像：**

docker pull redis

**创建容器：**

docker run -di --name=myredis -p 80:80 nginx

## 4.4 redis

**拉取镜像：**

docker pull redis

**创建容器：**

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

# 5 迁移与备份

## 5.1 容器保存为镜像

docker commit mynginx mynginx\_i

## 5.2 镜像备份

将镜像保存为tar文件

docker save -o mynginx.tar mynginx\_i

## 5.3 镜像恢复与迁移

首先我们先删除掉mynginx\_ img镜像然后执行此命令进行恢复

docker load -i mynginx. tar

-i输入的文件

执行后再次查看镜像，可以看到镜像已经恢复

# 6 Dockerfile

## 6.1 是什么

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\unbeantu）**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。

## 6.2 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,不存在会去下载 |  |
| MAINTAINER user\_name | 声明镜像的创建者 |  |
| ENV key value | 设置环境变量（可以写多个，如JDK） |  |
| RUN commond | 是Dockerfile的核心部分（可以多个，如创建目录） |  |
| ADD source\_dir/file dest\_dir/file | 将宿主机的文件复制到容器内，如果是一一个压缩文件，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 |  |

## 6.3 使用脚本创建镜像

案例，jdk1.8

基础镜像：centos7

上传; jdk1.8.0\_171-linux-x64.zip到/usr/local/dockerjdk8下

**vim Dcokerfile**

FROM centos:7

MAINTAINER lipu

WORKDIR /usr

RUN mkdir /usr/local/java

ADD jdk1.8.0\_171-linux-x64.zip /usr/local/java/

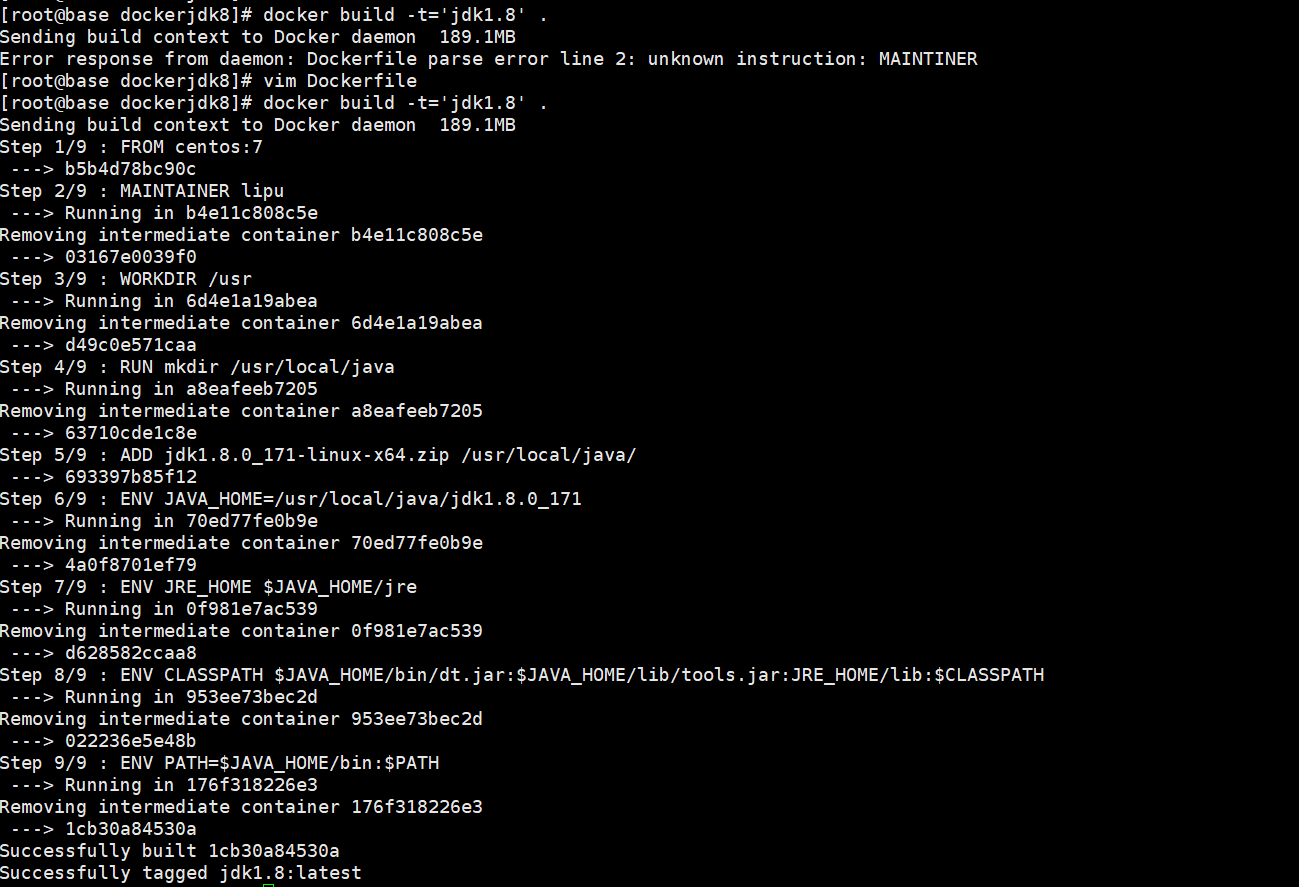
ENV JAVA\_HOME=/usr/local/java/jdk1.8.0\_171

ENV JRE\_HOME $JAVA\_HOME/jre

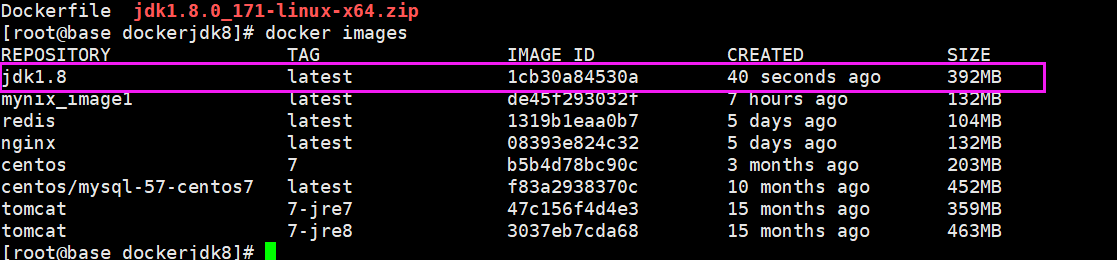
ENV CLASSPATH $JAVA\_HOME/bin/dt.jar:$JAVA\_HOME/lib/tools.jar:JRE\_HOME/lib:$CLASSPATH

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

执行创建 docker build -t='jdk1.8' .



查看：



# 7 Docker私有仓库

存储企业内部应用级的镜像

## 7.1 搭建与配置

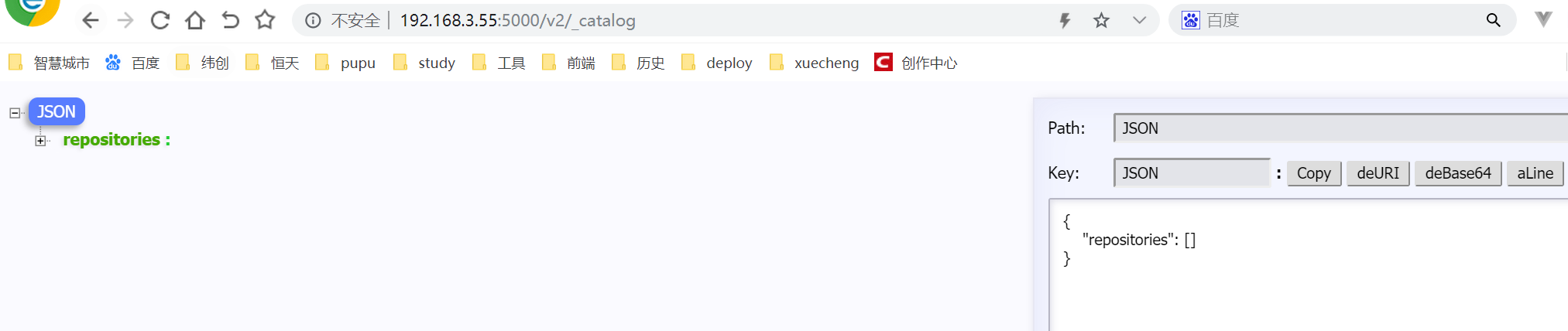
(1)拉取私有仓库镜像(此步省略)

docker pull registry

(2)启动私有仓库容器

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

(3)打开浏览器输入地址http://192.168.184.141:5000/v2/ catalog看到{"repositories":[]}表示私有仓库搭建成功并且内容为空



(4)修改daemon.json

vi /etc/docker/daemon. json

添加以下内容，保存退出。

{"insecure-registries" :["192.168.184.141:5000"]}

此步用于让docker信任私有仓库地址

{

"registry-mirrors":["https://docker.mirrors.ustc.edu.cn"],

"insecure-registries":["192.168.3.55:5000"]

}

（5）重启docker 生效配置

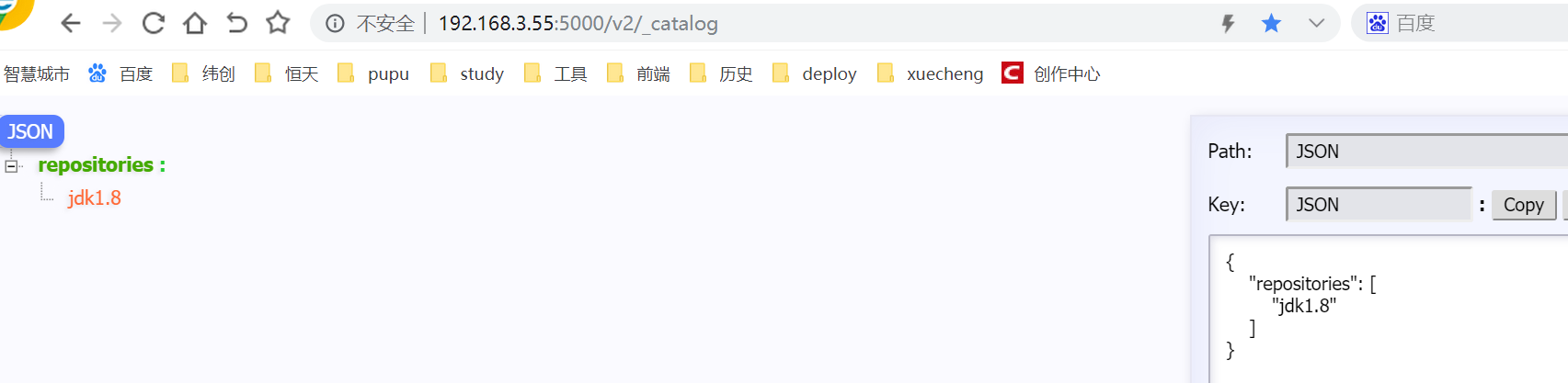
systemctl restart docker

## 7.2 镜像上传至私有仓库

dopcker tag jdk1.8 192.168.3.5:5000/jdk1.8

启动因为重启停掉的容器

docker push 192.168.3.55:5000/jdk1.8



## 7.3 下载私有仓库的镜像

(1)修改daemon.json

vi /etc/docker/daemon. json

添加以下内容，保存退出。

{"insecure-registries" :["192.168.184.141:5000"]}

此步用于让docker信任私有仓库地址

{

"registry-mirrors":["https://docker.mirrors.ustc.edu.cn"],

"insecure-registries":["192.168.3.55:5000"]

}

（2）重启docker 生效配置

systemctl restart docker

（3）重启docker 生效配置

docker pull 192.168.3.55:5000/jdk1.8