

Jenkins+Svn+Docker+SpringCloud  
实现可持续自动化微服务

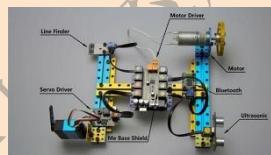


# 前言

Spring 并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

Spring Cloud 一站式解决方案能在从容应对业务发展的同时大大减少开发成本。同时，随着近几年微服务架构和 Docker 容器概念的火爆，也会让 Spring Cloud 在未来越来越“云”化的软件开发风格中立有一席之地，有效推进服务端软件系统技术水平的进步。

个人认为下面三幅图可以形象的描述 Spring\Spring Boot\Spring Cloud。



Jenkins 是一个开源软件项目，是基于 Java 开发的一种持续集成工具，用于监控持续重复的工作，旨在提供一个开放易用的软件平台，使软件的持续集成变成可能。

这是本人初次接触 Spring Cloud 微服务及 Docker 结合的学习点滴，帮助自己或他人更好地去理解微服务概念及其庞大的组件体系，给予服务开发上的便利，同时结合 Docker 虚拟化技术进行整合应用的实践过程，以作者初学的角度去理解 Spring Cloud 微服务概念、框架搭建的基本知识及 Docker 的整合应用。

# 目录

1、快速搭建虚拟机环境.....	7
1.1 软件需求.....	7
1.2 准备工作.....	7
1.2.1 安装 virtualBox .....	7
1.2.2 安装 vagrant .....	7
1.2.3 下载 box 镜像文件.....	8
1.2.4 创建虚拟机文件夹.....	8
1.3 命令详解.....	8
1.4 创建虚拟机.....	9
1.4.1 添加 box 镜像.....	9
1.4.2 查看当前 box 列表.....	9
1.4.3 初始化 box.....	9
1.4.4 编辑 Vagrantfile .....	10
1.4.5 启动本地环境.....	11
1.5 连接虚拟机.....	12
2、流程设计及规划.....	14
2.1 整体流程图.....	14
2.2 服务器规划.....	14
3、安装 Subversion(192.168.33.25).....	16
3.1 安装 HTTP 和 SVN 相关软件包.....	16
3.2 建立版本库目录.....	16
3.3 配置 httpd 文件.....	16
3.4 创建 SVN 用户 .....	16
3.5 创建并配置 SVN 库 .....	17
3.6 启动 httpd 并加入开机启动.....	17
3.7 通过浏览器尝试登陆 SVN .....	17
4、安装 Docker(192.168.33.21\22\23\24).....	19
4.1 准备工作.....	19
4.2 安装步骤.....	19
4.2.1 卸载旧版本.....	19
4.2.2 设置 REPOSITORY .....	19
4.2.3 设置 docker 稳定的源.....	19
4.2.4 更新 yum 的安装索引.....	19
4.2.5 安装 docker 版本.....	19
4.2.6 启动 docker 服务.....	20
4.3 相关命令.....	20
4.4 Docker 体验 .....	20
4.4.1 安装 nginx.....	21
4.4.2 安装 mysql.....	22
4.4.3 总结.....	24
5、搭建 docker 私有仓库.....	26

# Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

5.1 服务端操作(192.168.33.24) .....	26
5.1.1 停止并关闭防火墙.....	26
5.1.2 安装并启动 docker.....	26
5.1.3 拉取本地私有仓库 registry.....	26
5.1.4 查看 registry 镜像 .....	26
5.1.5 基于私有仓库镜像运行容器.....	27
5.1.6 查看 docker 进程.....	27
5.1.7 验证私有仓库.....	27
5.1.8 指定私有仓库 URL .....	27
5.1.9 开放管理端口映射.....	27
5.1.9 重启 docker 服务.....	28
5.1.10 从 Docker 私有仓库上拉取一个镜像测试.....	28
5.1.11 创建镜像链接为基础镜像打个标签.....	28
5.1.12 上传镜像到本地私有仓库.....	28
5.1.13 查看私有仓库镜像.....	28
5.2 客户端操作.....	29
5.2.1 停止并关闭防火墙.....	29
5.2.2 安装并启动 docker.....	29
5.2.3 修改 Docker 配置文件 .....	29
5.2.4 重启 docker 服务.....	29
5.2.5 从私有仓库中拉取 nginx 镜像 .....	29
6、搭建 DockerSwarm 模式集群 .....	31
6.1 集群规划.....	31
6.2 所有节点操作.....	31
6.2.1 安装 docker 的 CE 版本.....	31
6.2.2 编辑 hosts 文件 .....	31
6.3 创建 swarm 模式集群.....	31
6.3.1 swarm 主节点操作(192.168.33.21) .....	31
6.3.2 swarm 从节点操作(192.168.33.22) .....	32
6.3.3 swarm 主节点操作(192.168.33.21) .....	32
7、安装 jenkins .....	34
7.1 安装 jenkins 服务器(192.168.33.23).....	34
7.1.1 软件需求.....	34
7.1.2 下载 Jenkins 的 war 包.....	34
7.1.3 上传软件.....	34
7.1.4 解压软件.....	35
7.1.5 添加环境变量.....	35
7.2 启动 jenkins .....	35
7.3 初始化 jenkins .....	36
7.4 配置 jenkins 的 Jdk 和 Maven .....	37
7.5 体验 jenkins .....	38
7.5.1 构建一个 maven 项目 .....	38
8、搭建 Portainer 可视化界面.....	43
8.1 swarm 主节点操作(192.168.33.21) .....	43

# Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

8.1.1 下载 portainer 镜像 .....	43
8.1.2 启动 portainer .....	43
8.1.3 查看 docker 进程.....	43
8.2 浏览 portainer 界面 .....	43
8.2.1 设置密码.....	44
8.3Portainer 体验.....	47
8.3.1 创建 nginx 服务 .....	47
9、Spring Boot 快速入门.....	52
9.1 搭建第一个 Spring Boot 程序.....	52
9.1.1 创建父项目.....	52
9.1.2 创建子项目.....	53
9.1.3 编写 pom.xml .....	55
9.1.4 更新 maven 项目 .....	56
9.1.5 便捷配置.....	57
9.1.6 测试 Spring Boot 应用.....	58
9.2 单元测试.....	59
9.3 优化配置.....	60
9.3.1 定制 Banner.....	60
9.3.2 关闭 Banner.....	61
9.3.3 配置文件.....	61
9.4 构建可执行 jar 包 .....	64
9.5 运行.....	65
10、Spring Cloud 快速入门.....	67
10.1 服务的注册与发现（Eureka）.....	67
10.1.1 创建服务注册中心.....	67
10.1.2 创建服务提供者.....	70
10.2 配置服务中心(Config) .....	71
10.2.1 创建 SVN 配置文件仓库 .....	71
10.2.2 创建配置服务端.....	72
10.2.3 创建使用配置服务的客户端.....	73
11、SpringCloud 与 Docker 结合 .....	77
11.1 编辑 POM 文件 .....	77
11.2 编译\打包\上传.....	77
12、Jemeter 压力测试.....	80
12.1 安装 Jemeter .....	80
12.2 解压文件.....	80
12.3 启动 jemeter.....	80
12.4 体验 jemeter.....	80
12.4.1 准备一个待测试的服务 .....	80
12.4.2 创建 jemeter 压测内容 .....	81
12.4.3 开始运行.....	83
12.4.4 测试结果.....	83
13、jenkins 自动化构建 .....	85

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

13.1 构建 maven 项目 docker 镜像.....	85
13.1.1 创建 maven 工程.....	85
13.1.2 添加 docker-maven 插件.....	85
13.1.3 在 jenkins 中构建一个 maven 项目 .....	86
13.1.4 配置构建项目 .....	86
13.1.5 配置构建项目要执行的命令 .....	86
13.1.6 完成构建项目的配置.....	87
13.1.7 立即构建.....	87
13.1.8 验证构建成功的镜像文件.....	88
13.2 设置自动化构建.....	89
13.3 jenkins 状态通知.....	90
13.3.1 与钉钉关联.....	90
13.4 可能遇到的问题.....	92
14、Portainer 管理微服务容器.....	94

# 第一章

## 快速搭建虚拟机环境



Vagrant 是一个基于 Ruby 的工具，用于创建和部署虚拟化开发环境。它使用 Oracle 的开源 VirtualBox 虚拟化系统，使用 Chef 创建自动化虚拟环境。非常适合 php/python/ruby/java 这类语言开发 web 应用，“代码在我机子上运行没有问题”这种说辞将成为历史。

我们可以通过 Vagrant 封装一个 Linux 的开发环境，分发给团队成员。成员可以在自己喜欢的桌面系统（Mac/Windows/Linux）上开发程序，代码却能统一在封装好的环境里运行，非常霸气。

# 1、快速搭建虚拟机环境

## 1.1 软件需求

- 1) vagrant\_1.9.6\_x86\_64.msi
- 2) VirtualBox-5.1.24-117012-Win.exe
- 3) vagrant-centos-7.2.box

	vagrant-centos-7.2.box	2017/7/8 7:03	BOX 文件	616,238 KB
	vagrant_1.9.6_x86_64.msi	2017/7/7 14:28	Windows Install...	228,780 KB
	VirtualBox-5.1.24-117012-Win.exe	2017/7/24 9:57	应用程序	120,891 KB

## 1.2 准备工作

### 1.2.1 安装 virtualBox

到官网 <https://www.virtualbox.org/wiki/Downloads> 下载最新版的 Virtualbox，然后双击安装，一直点击确认完成。

### 1.2.2 安装 vagrant

官网 <https://www.vagrantup.com/downloads.html> 下载最新版的 Vagrant，然后双击安装，一直点击确认完成。在 windows 下安装 vagrant，为了写入相应配置到环境变量，系统可能会要求重新启动。在命令行中，输入 vagrant，查看程序是不是已经运行了。如果不行，请检查一下\$PATH 里面是否包含 vagrant 所在的路径。

```
E:\wmos>vagrant -v  
Vagrant 1.9.6  
  
E:\wmos>
```

### 1.2.3 下载 box 镜像文件

box 是什么？接下来，我们需要选择使用何种操作系统，这里以 centos7.2 为例。以前基于虚拟机的工作流，我们需要下载 ISO 镜像，安装系统，设置系统等操作。而 Vagrant 开源社区提供了许多已经打包好的操作系统，我们称之为 box。你可以从 box 下载地址（下文列出），找到你想要的 box，当然你也可以自己制作一个。

- 官方仓库：<https://atlas.hashicorp.com/boxes/search>
- 官方镜像：<https://vagrantcloud.com/boxes/search>
- 第三方仓库：<http://www.vagrantbox.es/>

### 1.2.4 创建虚拟机文件夹

为了方便管理，我们把虚拟机统一放在一个文件夹下，这里我创建的目录为：E:\vmos，没有特殊要求，可以随意创建。

## 1.3 命令详解

命令	作用
vagrant box add	添加box的操作
vagrant init	初始化box的操作，会生成vagrant的配置文件Vagrantfile
vagrant up	启动本地环境
vagrant ssh	通过ssh登录本地环境所在虚拟机
vagrant halt	关闭本地环境
vagrant suspend	暂停本地环境
vagrant resume	恢复本地环境
vagrant reload	修改了Vagrantfile后，使之生效（相当于先 halt，再 up）
vagrant destroy	彻底移除本地环境
vagrant box list	显示当前已经添加的box列表
vagrant box remove	删除相应的box
vagrant package	打包命令，可以把当前的运行的虚拟机环境进行打包
vagrant plugin	用于安装卸载插件
vagrant status	获取当前虚拟机的状态
vagrant global-status	显示当前用户Vagrant的所有环境状态

## 1.4 创建虚拟机

### 1.4.1 添加 box 镜像

命令 : vagrant add box centos7.2 E:\U\ vagrant-centos-7.2.box , 命令中最好不要含有中文。

```
E:\vmos>vagrant box add centos7.2 E:\U\vagrant-centos-7.2.box
=> box: Box file was not detected as metadata. Adding it directly...
=> box: Adding box 'centos7.2' <v0> for provider:
    box: Unpacking necessary files from: file:///E:/U/vagrant-centos-7.2.box
    box: Progress: 100% <Rate: 603M/s, Estimated time remaining: --:--:-->
=> box: Successfully added box 'centos7.2' <v0> for 'virtualbox'!
```

### 1.4.2 查看当前 box 列表

命令 : vagrant box list

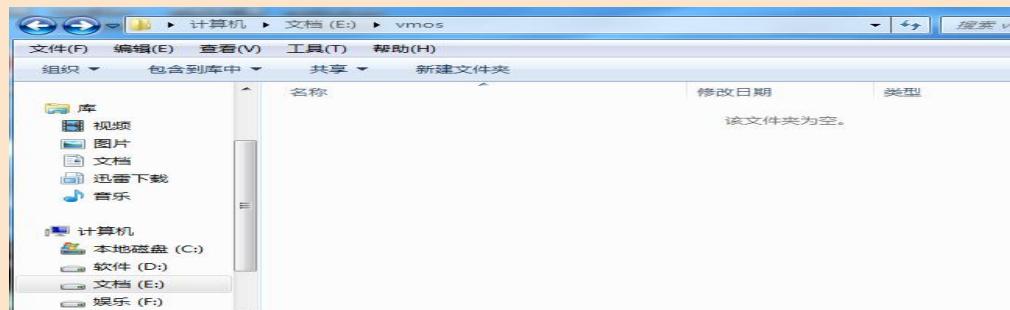
```
E:\vmos>vagrant box list
centos7  <virtualbox, 0>
centos7.2 <virtualbox, 0>

E:\vmos>
```

### 1.4.3 初始化 box

命令 : vagrant init

初始化 box 的操作 , 会生成 vagrant 的配置文件 Vagrantfile , 生成之前 ,  
文件夹里面为空 :

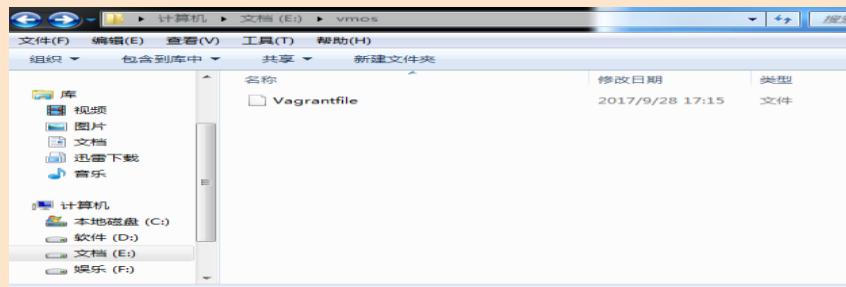


初始化操作：

```
E:\vmos>vagrant init
A 'Vagrantfile' has been placed in this directory. You are now
ready to 'vagrant up' your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
'vagrantup.com' for more information on using Vagrant.

E:\vmos>
```

生成之后，多了一个文件：



## 1.4.4 编辑 Vagrantfile

打开刚刚生成的 Vagrantfile 文件，默认内容如下：

```
1 # -*- mode: ruby -*-
2 # vim: set ft=ruby :
3
4 # All Vagrant configuration is done below. The "2" in Vagrant.configure
5 # configures the configuration version (we support older styles for
6 # backwards compatibility). Please don't change it unless you know what
7 # you're doing.
8 Vagrant.configure("2") do |config|
9   # The most common configuration options are documented and commented below.
10  # For a complete reference, please see the online documentation at
11  # https://docs.vagrantup.com
12
13  # Every Vagrant development environment requires a box. You can search for
14  # boxes at https://vagrantcloud.com/search.
15  config.vm.box = "base"
16
17  # Disable automatic box update checking. If you disable this, then
18  # boxes will only be checked for updates when the user runs
19  # vagrant box updated. This is not recommended.
20  config.vbox.check_update = false
21
22  # Create a forwarded port mapping which allows access to a specific port
23  # within the machine from a port on the host machine. In the example below,
24  # port 8080 on the host machine will map to port 80 on the guest machine.
25  # NOTE: This will enable public access to the opened port.
26  config.vm.network "forwarded_port", guest: 80, host: 8080
27
28  # Create a forwarded port mapping which allows access to a specific port
29  # within the machine from a port on the host machine and only allow access
30  # via 127.0.0.1 to disable public access
31  # config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"
32
33  # Create a private network, which allows host-only access to the machine
34  # using a specific IP.
35  # config.vm.network "private_network", ip: "192.168.33.10"
36
37  # Create a public network, which generally matches to bridged network.
# config.vm.network "public_network"
```

修改后的内容如下：

```
VAGRANTFILE_API_VERSION = "2"          指定vagrant版本，默认为2，不用修改
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|          可循环创建几个相同配置的虚拟机

(1..).each do |i|          开始循环
  config.vm.define vm_name = "hbase#{i}" do |config|
    config.vm.provider "virtualbox" do |v|
      v.customize ["modifyvm", :id, "--name", vm_name, "--memory", "2048", '--cpus', 2]          指定虚拟机的hostname
    end          虚拟机的一些配置，如：内存、CPU等
    config.vm.box = "centos7"          指定所使用的box镜像名字
    config.vm.hostname = vm_name
    config.ssh.username = "vagrant"          指定要创建的虚拟机用户名
    config.vm.network :private_network, ip: "192.168.33.1#{i}"          指定虚拟机的IP地址
  end
end
end
```

## 1.4.5 启动本地环境

命令 : vagrant up

上一步完成之后，接下来我们就可以启动创建所需要的虚拟机了。

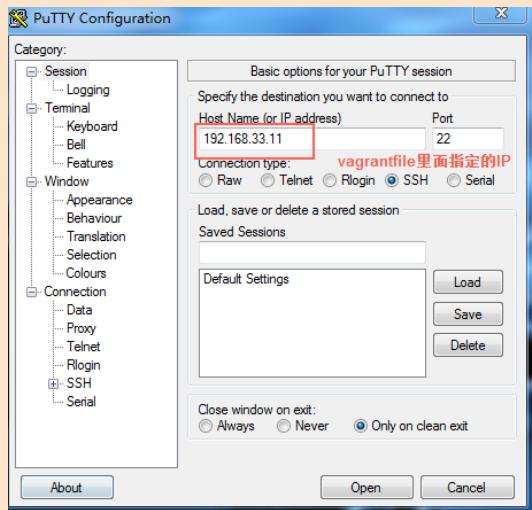
```
E:\vmos>vagrant up
Bringing machine 'centos1' up with 'virtualbox' provider...
==> centos1: Importing base box 'centos7'...
Progress: 90%
```

创建成功后，正在尝试 SSH 连接

至此虚拟机已经创建成功了。

## 1.5 连接虚拟机

虚拟机创建成功之后，我们就可以使用 SSH 连接工具连接上去。新创建好的用户包含（root 和 vagrant，默认的密码为 vagrant），我使用的是 putty 工具进行连接。



点击“open”，输入用户名和密码

```

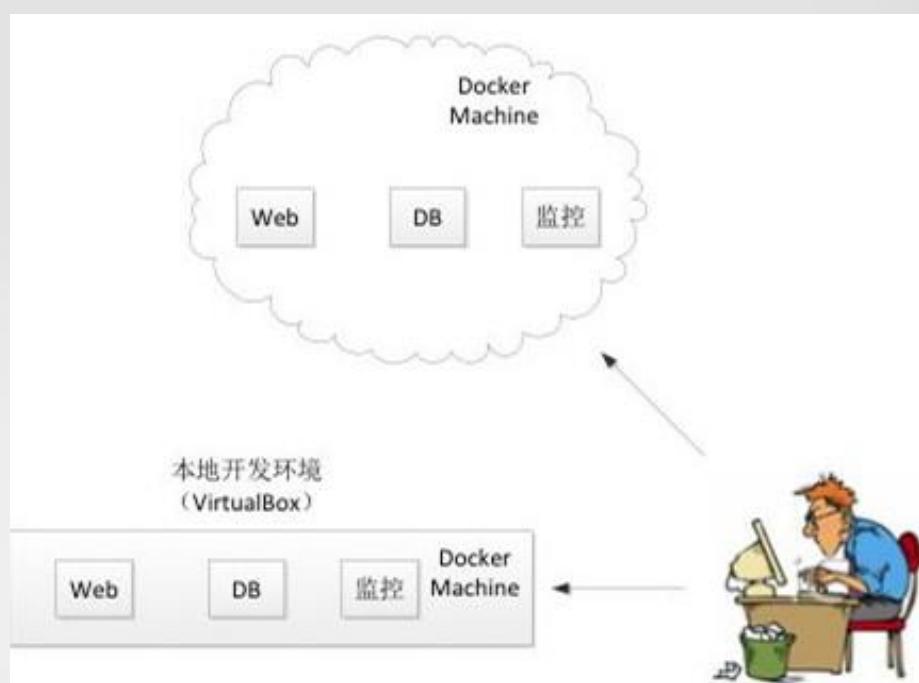
root@hbase1:~#
login as: root
root@192.168.33.11's password:
Last login: Thu Sep 28 10:35:31 2017 from 192.168.33.1
[root@hbase1 ~]# hostname
hbase1
[root@hbase1 ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    link/ether 08:00:27:d6:d8:7d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 85916sec preferred_lft 85916sec
    inet6 fe80::a00:27ff:fed6:d87d/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    link/ether 08:00:27:d6:d8:7d brd ff:ff:ff:ff:ff:ff
    inet 192.168.33.11/24 brd 192.168.33.255 scope global enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed6:d87d/64 scope link
        valid_lft forever preferred_lft forever
[root@hbase1 ~]#

```

可以看到，已经成功连接到这台新创建的 centos 系统，所有的配置都是 vagrantfile 文件里面指定好的，非常的便捷。虚拟机的相关操作命令，请参照前面的内容执行。

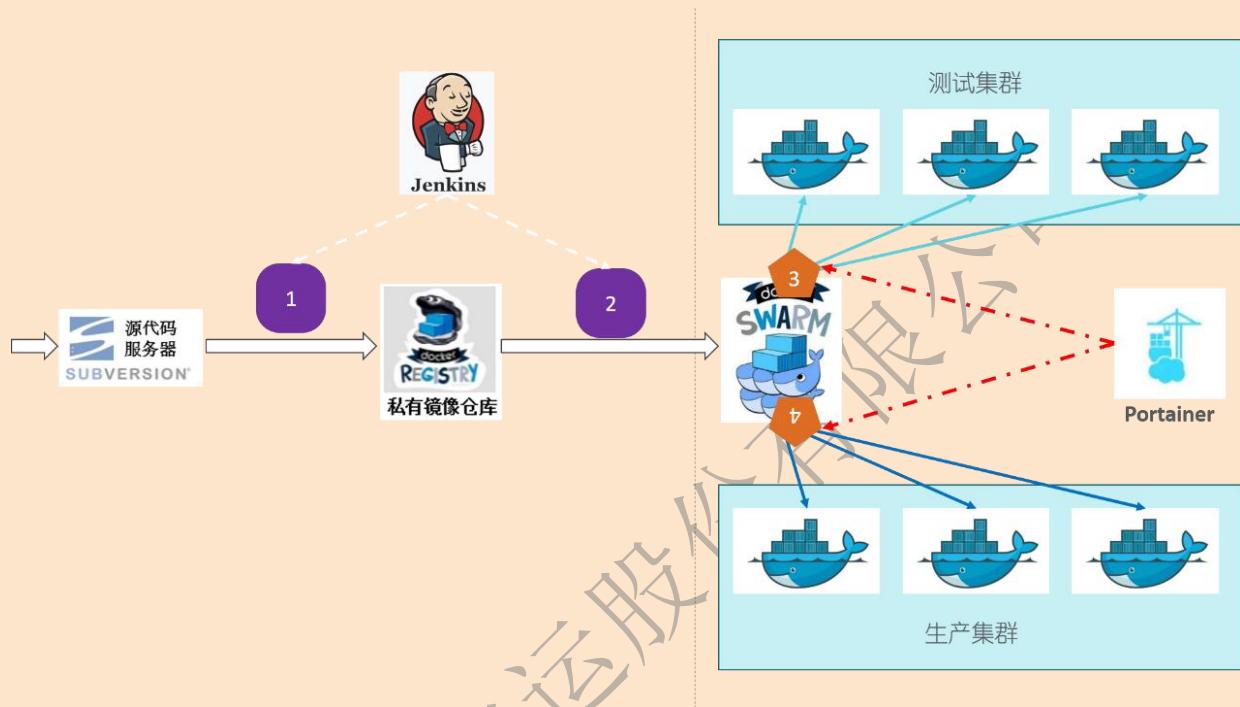
# 第二章

## 流程设计及规划



## 2、流程设计及规划

### 2.1 整体流程图



- ✧ 通过 Jenkins 管理代码的编译、打包和上传；
- ✧ 通过 Portainer 操作 Docker Swarm 模式集群，管理发布、升级等；

### 2.2 服务器规划

IP	Hostname	应用程序
192.168.33.21	Swarm1	Swarm 主节点、Portainer
192.168.33.22	Swarm2	Swarm 从节点
192.168.33.23	Swarm3	Jenkins
192.168.33.24	Swarm4	Registry
192.168.33.25	Swarm5	Subversion

# 第三章

## 安装 Subversion



Subversion 是一个自由开源的版本控制系统。在 Subversion 管理下，文件和目录可以超越时空。Subversion 将文件存放在中心版本库里，这个版本库很像一个普通的文件服务器，不同的是，它可以记录每一次文件和目录的修改情况，这样就可以借此将数据恢复到以前的版本，并可以查看数据的更改细节。正因为如此，许多人将版本控制系统当作一种神奇的“时间机器”。

### 3、安装 Subversion(192.168.33.25)

#### 3.1 安装 HTTP 和 SVN 相关软件包

命令 : yum install -y httpd subversion mod\_dav\_svn

#### 3.2 建立版本库目录

命令 : mkdir -p /data/svn/

#### 3.3 配置 httpd 文件

命令 : vi /etc/httpd/conf/httpd.conf , 末尾添加如下信息 :

```
<Location /svn>
    DAV svn
    SVNParentPath /data/svn/
    AuthType Basic
    AuthName "SVN Repository"
    AuthUserFile /etc/svn-auth-accounts
    Require valid-user
</Location>
```

#### 3.4 创建 SVN 用户

利用 httpd 包生成的命令 “htpasswd” 来创建, 把用户名 “test” 换成自定义的用户名

命令 : htpasswd -cm /etc/svn-auth-accounts test

说明 : 选项 “-c” 是用来创建密码文件/etc/svn-auth-accounts ;

选项 “-m” 是用来给用户创建 MD5 加密密码 ;

注意 : 如创建第二个用户时 ,请勿使用 “-c” 选项 ,否则会重新生成并覆盖原文件 ;

### 3.5 创建并配置 SVN 库

命令 : svnadmin create /data/svn/repo

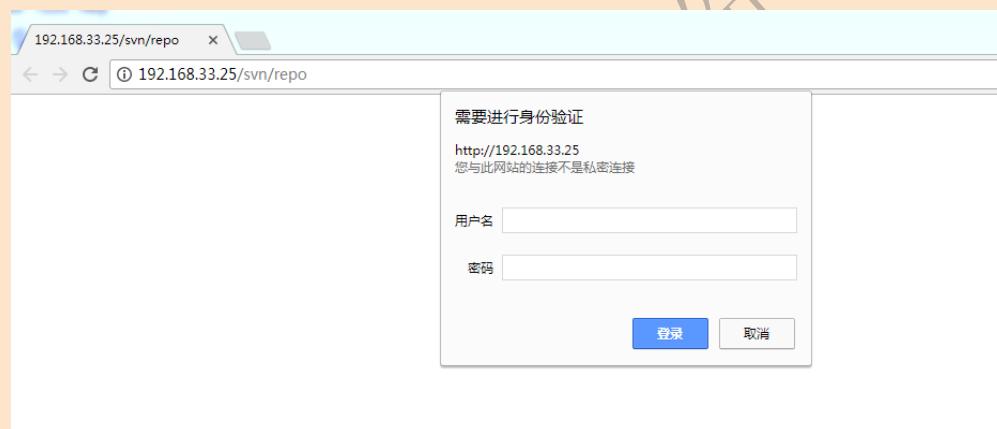
```
chown -R apache:apache /data/svn/repo
```

### 3.6 启动 httpd 并加入开机启动

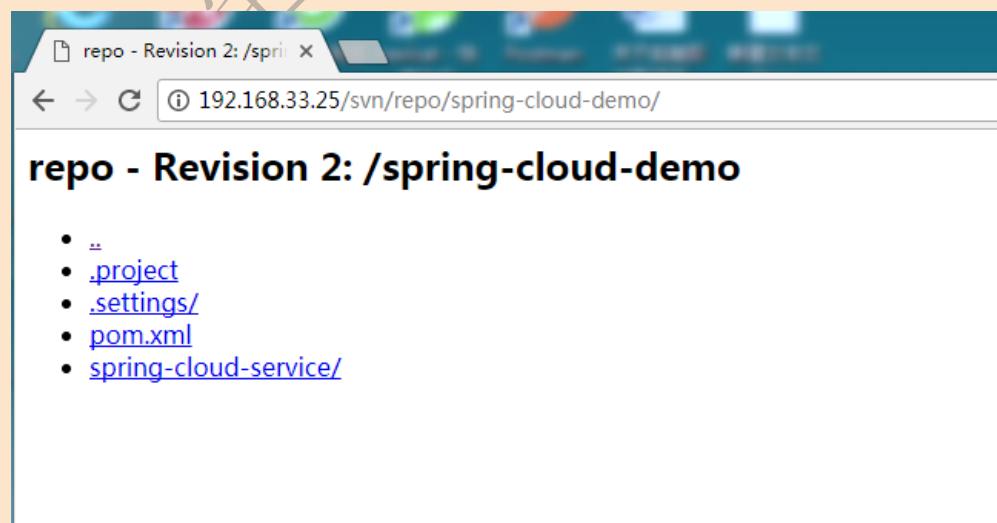
命令 : systemctl restart httpd && systemctl enable httpd

### 3.7 通过浏览器尝试登陆 SVN

在浏览器输入 : <http://192.168.33.25/svn/repo>

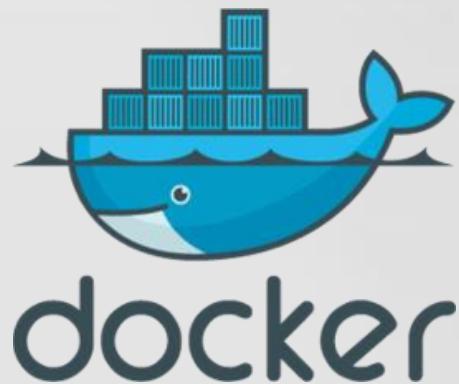


输入账号和密码，点击确定即可登录成功！然后我们再把项目上传到 svn 中。



# 第四章

## Centos7 安装 docker



Docker 是一个开源的引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。开发者在笔记本上编译测试通过的容器可以批量地在生产环境中部署，包括 VMs ( 虚拟机 )、bare metal、OpenStack 集群和其他的基础应用平台。

Docker 通常用于如下场景：

- web 应用的自动化打包和发布；
- 自动化测试和持续集成、发布；
- 在服务型环境中部署和调整数据库或其他的后台应用；
- 从头编译或者扩展现有的 OpenShift 或 Cloud Foundry 平台来搭建自己的 PaaS 环境。

## 4、安装 Docker(192.168.33.21\22\23\24)

### 4.1 准备工作

- linux 系统，不建议 windows，这里我们选择了 centos7
- docker 最新版本

### 4.2 安装步骤

#### 4.2.1 卸载旧版本

命令 : yum remove -y docker docker-common docker-selinux docker-engine

#### 4.2.2 设置 REPOSITORY

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

#### 4.2.3 设置 docker 稳定的源

命令 : yum-config-manager --add-repo \  
<https://download.docker.com/linux/centos/docker-ce.repo>

#### 4.2.4 更新 yum 的安装索引

命令 : yum makecache fast

#### 4.2.5 安装 docker 版本

命令 : yum install -y docker-ce

### 4.2.6 启动 docker 服务

命令：systemctl start docker.service，作用：启动 docker 服务。

命令：systemctl enable docker.service，作用：使 docker 服务开机自启动。

```
[root@centos1 ~]# systemctl start docker.service
[root@centos1 ~]# systemctl enable docker.service
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
```

至此，我们在 centos7 系统上成功安装了 docker，测试一下 docker 的优势吧。

### 4.3 相关命令

Docker 的命令不是很复杂，首先我们应该了解一些常用的一些命令，如下：

镜像管理

```
docker images: 列出本地所有镜像
docker search <IMAGE_ID/NAME>: 查找image
docker pull <IMAGE_ID>: 下载image
docker push <IMAGE_ID>: 上传image
docker rmi <IMAGE_ID>: 删除image
```

容器管理

```
docker run -i -t <IMAGE_ID> /bin/bash: -i: 标准输入给容器    -t: 分配一个虚拟终端    /bin/bash: 执行bash脚本
-d: 以守护进程方式运行（后台）
-p: 默认匹配 docker 容器的 5000 端口到宿主机的 49153 到 65535 端口
-p <HOST_PORT>:<CONTAINER_PORT>: 指定端口号
--name: 指定容器的名称
--rm: 退出时删除容器

docker stop <CONTAINER_ID>: 停止container
docker start <CONTAINER_ID>: 重新启动container
docker ps - Lists containers.
    -l: 显示最后启动的容器
    -a: 同时显示停止的容器, 默认只显示启动状态

docker attach <CONTAINER_ID> 连接到启动的容器
docker logs <CONTAINER_ID> : 输出容器日志
    -f: 实时输出

docker cp <CONTAINER_ID>:path hostpath: 复制容器内的文件到宿主机目录上
```

参考：

<http://www.runoob.com/docker/docker-command-manual.html>

[http://blog.csdn.net/zhang\\_jiayu/article/details/42611469](http://blog.csdn.net/zhang_jiayu/article/details/42611469)

### 4.4 Docker 体验

前面已经在 centos7 上成功安装了 docker，那么 docker 到底有什么优势呢？我们现在来体验一下吧。

### 4.4.1 安装 nginx

#### 4.4.1.1 下载镜像到本地

命令 : docker pull nginx

```
[root@centos1 ~]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
afeb2bfd31c0: Downloading [====>] 2.096 MB/22.49 MB
7ff5d10493db: Downloading [====>] 2.063 MB/21.87 MB
d2562f1ae1d0: Download complete
```

#### 4.4.1.2 确认镜像已被获取

命令 : docker images nginx

```
[root@centos1 ~]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
afeb2bfd31c0: Pull complete
7ff5d10493db: Pull complete
d2562f1ae1d0: Pull complete
Digest: sha256:af32e714a9cc3157157374e68c818b05eb9e0737aac06b55a09da374209a8f9
[root@centos1 ~]# docker images nginx
REPOSITORY          TAG           IMAGE ID            CREATED             SIZE
docker.io/nginx      latest        da5939581ac8   2 weeks ago        108.3 MB
```

#### 4.4.1.3 运行容器

命令 : docker run -d -p 8080:80 nginx

```
[root@centos1 ~]# docker run -d -p 8080:80 nginx
95fd5a2081864540b32637c10366c521eea9f08c52496e7ea10d239beaa44515
[root@centos1 ~]#
```

非常快，几乎是秒级就成功了，相比普通的搭建 nginx 环境便捷多了，这就是它的优势。真的创建成功了吗？我们用浏览器来测试一下吧，在地址栏输入：

<http://192.168.33.24:8080>，后回车，看见 nginx 了吗？



## 4.4.1.4 再启动一个 nginx

命令：docker run -d -p 8081:80 nginx

```
[root@centos1 ~]# docker run -d -p 8081:80 nginx
19a24da0a0b16619d113755962632530bbd540b7ffd34329f43c580760b46b39
[root@centos1 ~]#
```

还是秒级完成了，在浏览器中输入 <http://192.168.33.24:8081> 后回车测试一下，也看到了熟悉的 nginx。



## 4.4.2 安装 mysql

### 4.4.2.1 查找 mysql 镜像

命令：docker search mysql

```
[root@centos1 ~]# docker search mysql
NAME                               DESCRIPTION                                     STARS   OFFICIAL   AUTOMATED
docker.io/docker                MySQL is a widely used, open-source relati... 5015    [OK]
maria                            MariaDB is a community-developed fork of M... 1044    [OK]
mysql                             MySQL is a widely used, open-source relati... 1033    [OK]
mariadb                           MariaDB Server is a fork of the MySQL rela... 299     [OK]
mariadb/mariadb                  MariaDB Server is a fork of MySQL. This im... 13      [OK]
mariadb/persona                  Persona Server is a fork of the MySQL rela... 293     [OK]
zabbix/zabbix                   Zabbix Server with MySQL database support 94     [OK]
imagecontaining/mysql            Image containing MySQL, Optimized no inn... 93     [OK]
mariadb/mariadb57                MariaDB Server is a fork of MySQL. This im... 77     [OK]
zabbix/zabbix                   Zabbix Frontend based on MySQL web-server ... 35     [OK]
mariadb/mariadb57                MariaDB Server is a fork of MySQL. This im... 34     [OK]
ubuntu-14.04-nginx-php-pgsql     ubuntu-14.04-nginx-php-pgsql-admin-mysq... 16     [OK]
mariadb/mariadb57-backup         Backup MySQL to file supports periodic bac... 16     [OK]
mariadb/mariadb57-centos7       A Docker Image for MySQL 5.7 running on Ce... 12     [OK]
mariadb/mariadb57-centos7-backup MySQL 5.7 SQL database server with bac... 11     [OK]
mariadb/mariadb57-centos7-backup MySQL 5.7 SQL database server with bac... 9      [OK]
mariadb/mariadb57-centos7       DEPRECATED: A CentOS7 based MySQL V5.5 ima... 6      [OK]
mariadb/mariadb57-centos7-backup Backup all DBs in a MySQL server 3      [OK]
mariadb/mariadb57-centos7       A Docker Image for MySQL 5.7 SQL database s... 3      [OK]
mariadb/mariadb57-centos7-backup MySQL 5.7 SQL database server, open-source relat... 0      [OK]
mariadb/mariadb57-centos7       MySQL 5.7 SQL database server 0      [OK]
mariadb/mariadb57-centos7-backup Images used in CI of cf-mysql-release 0      [OK]
mariadb/mariadb57-centos7       Improved MySQL service with support for ... 0      [OK]
mariadb/mariadb57-centos7-backup Docker Royal package 0      [OK]
```

### 4.4.2.2 下载镜像到本地

命令：docker pull mysql

```
[root@centos1 ~]# docker pull mysql
Using default tag: latest
Trying to pull repository docker.io/library/mysql ...
latest: Pulling from docker.io/library/mysql
ed1d0a3a64e4: Downloading [==>          ] 1.607 MB/52.6 MB
fdb8d83dece3: Download complete
75b6cc7b50d3: Download complete
ed1d0a3a64e4: Download complete
8eb36a82c85b: Waiting
41be6f1a1c40: Waiting
0e1b414ea7c1: Waiting
914c20654a91: Waiting
587693eb988c: Waiting
b183c3585729: Waiting
315e21657aa4: Waiting
```

### 4.4.2.3 确认镜像已被下载

命令 : docker images mysql

```
[root@centos1 ~]# docker pull mysql
Using default tag: latest
Trying to pull repository docker.io/library/mysql ...
latest: Pulling from docker.io/library/mysql ...
aa18ad1a0d33: Pull complete
fd8bd8d3dece3: Pull complete
75b5cce7b50d3: Pull complete
e1d10a3a64e4: Pull complete
8eb36a82c85b: Pull complete
41be6f1a1c40: Pull complete
0e1b414eac71: Pull complete
931a2a2a2a2a: Pull complete
587693cb988c: Pull complete
b183c1585729: Pull complete
315e21657ea4: Pull complete
Digest: sha256:0dc3dab751ef46a6647234abdec2d47400f0dfbe77ab490b02bffdःae57846ed
[root@centos1 ~]# docker images mysql
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
docker.io/mysql     latest   b4e78b89bcf3   9 days ago   412.3 MB
[root@centos1 ~]#
```

### 4.4.2.4 运行容器

命令 :

```
docker run --name mysql-master-001 \
-d -p 13306:3306 -e MYSQL_ROOT_PASSWORD=dockertest \
--restart=always mysql
```

```
[root@centos1 ~]# docker run --name mysql-master-001 -d -p 13306:3306 -e MYSQL_ROOT_PASSWORD=dockertest --restart=always mysql
88e9b5d78ba8calb23171ba8d4c1792c964b19116cd94048fa61d71db8df9d48
[root@centos1 ~]#
```

也是秒级启动了 , 操作非常简单。

说明 :

- 1 ) --name 为容器指定一个名称
- 2 ) -p 将容器的 3306 端口映射到主机的 13306 端口
- 3 ) -e 设置环境变量 , 此处设置了 mysql 的 root 用户密码为 dockertest.
- 4 ) 使用在 docker run 的时候使用-restart 参数来设置。

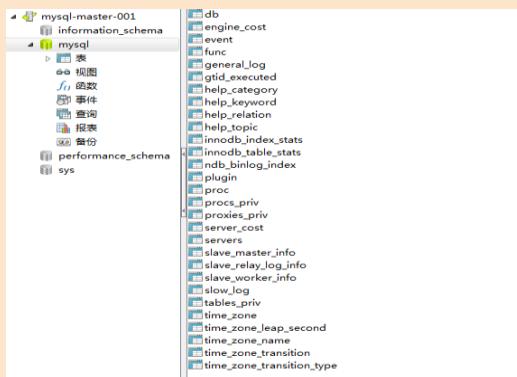
no - container : 不重启

on-failure - container:退出状态非 0 时重启

always:始终重启

#### 4.4.2.5 测试 mysql 容器

通过 navicate 等客户端可以连接这个 mysql。



可以看到，创建一个 mysql 的容器是多么的方便，按照常规的操作，安装一个 mysql 的步骤非常繁琐，还很容易出现问题。用 docker，一个命令，几秒钟就搞定了，这就是它的优势所在。

#### 4.4.3 总结

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。

Docker 核心解决的问题是利用 LXC 来实现类似 VM 的功能，从而利用更加节省的硬件资源提供给用户更多的计算资源。

云计算、大数据，移动技术的快速发展，加之企业业务需求的不断变化，导致企业架构要随时更改以适合业务需求，跟上技术更新的步伐。毫无疑问，这些重担都将压在企业开发人员身上；团队之间如何高效协调，快速交付产品，快速部署应用，以及满足企业业务需求，是开发人员亟需解决的问题。Docker 技术恰好可以帮助开发人员解决这些问题。

# 第五章

## 搭建 docker 私有仓库



Docker Hub 是一个很好的用于管理公共镜像的地方，我们可以在上面找到想要的镜像（Docker Hub 的下载量已经达到数亿次）；而且我们也可以把自己的镜像推送上去。但有的时候，使用场景需要我们有一个私有的镜像仓库用于管理自己的镜像，这个时候我们就通过 Registry 来实现此目的。

## 5、搭建 docker 私有仓库

### 5.1 服务端操作(192.168.33.24)

#### 5.1.1 停止并关闭防火墙

```
systemctl stop firewalld.service
```

```
systemctl disable firewalld.service
```

#### 5.1.2 安装并启动 docker

参照 4.2 的步骤进行操作

#### 5.1.3 拉取本地私有仓库 registry

命令 : docker pull registry

```
[root@k8s1 ~]# docker pull registry
Using default tag: latest
Trying to pull repository docker.io/library/registry ...
latest: Pulling from docker.io/library/registry
90f4dba627d6: Pull complete
b3e11d7b4f5e: Pull complete
1f032f3c8932: Pull complete
425585e7aedb: Pull complete
f45f535a83d2: Pull complete
Digest: sha256:0f8fe61fa337b8ef02217702ba979b47a7d68717d4628f31592ebff85915f3ba
```

#### 5.1.4 查看 registry 镜像

命令 : docker images

```
[root@k8s1 ~]# docker images
REPOSITORY          TAG           IMAGE ID            CREATED         SIZE
docker.io/registry  latest        28525f9a6e46   2 weeks ago    33.19 MB

[root@k8s1 ~]# mkdir -p /opt/data/registry
[root@k8s1 ~]# cd /opt/data/registry
[root@k8s1 registry]# ll
total 0
[root@k8s1 registry]#
```

### 5.1.5 基于私有仓库镜像运行容器

命令 : docker run -it -d -p 5000:5000 --name registry registry

```
[root@k8s1 registry]# docker run -d -p 5000:5000 --name registry --restart always -v /opt/data/registry:/tmp/registry docker.io/registry
a20f9472243feee4e629c3430f2664d7a701d827c9c61260d676d7171824b5b4e
[root@k8s1 registry]#
```

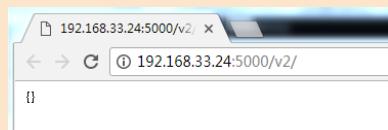
### 5.1.6 查看 docker 进程

命令 : docker ps -a

```
[root@k8s1 registry]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
a20f9472243f      docker.io/registry   "/entrypoint.sh /etc/"   3 minutes ago    Up About a minute   0.0.0.0:5000->5000
/tcp   registry
```

### 5.1.7 验证私有仓库

在浏览器中，输入 <http://192.168.33.24:5000/v2/>，看到以下信息，是正常的



### 5.1.8 指定私有仓库 URL

命令 : vi /etc/docker/daemon.json

```
{
  "insecure-registries":["192.168.33.24:5000"]
}
```

### 5.1.9 开放管理端口映射

管理端口在 /lib/systemd/system/docker.service 文件中修改

```
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2375 \
-H unix:///var/run/docker.sock -H tcp://0.0.0.0:7654
```

(此处默认 2375 为主管理端口, unix:///var/run/docker.sock 用于本地管理, 7654 是备用端口)

将管理地址写入 /etc/profile, 使 profile 生效。

```
echo 'export DOCKER_HOST=tcp://0.0.0.0:2375' >> /etc/profile
source /etc/profile
```

### 5.1.9 重启 docker 服务

命令 : systemctl restart docker

```
[root@k8s1 registry]# systemctl restart docker
[root@k8s1 registry]#
```

### 5.1.10 从 Docker 私有仓库上拉取一个镜像测试

命令 : docker pull nginx

```
[root@swarm4 ~]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
a1d9ee2e25fc3: Pull complete
9bda7d5afdf39: Pull complete
Digest: sha256:9fc103a62af6db7f188ac3376c60927db41f8b8d2354bf02d2290a672dc425
Status: Downloaded newer image for nginx:latest
```

命令 : docker images

```
[root@swarm4 ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
nginx              latest   40960efd7b8f   2 weeks ago   108MB
registry           latest   a07e3f32a779   2 weeks ago   33.3MB
```

### 5.1.11 创建镜像链接为基础镜像打个标签

命令 : docker tag nginx 192.168.33.24:5000/nginx

命令 : docker images

```
[root@swarm4 ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
192.168.33.24:5000/nginx  latest   40960efd7b8f   2 weeks ago   108MB
nginx              latest   40960efd7b8f   2 weeks ago   108MB
registry           latest   a07e3f32a779   2 weeks ago   33.3MB
[root@swarm4 ~]#
```

### 5.1.12 上传镜像到本地私有仓库

命令 : docker push 192.168.33.24:5000/nginx

```
[root@swarm4 ~]# docker push 192.168.33.24:5000/nginx
The push refers to a repository [192.168.33.24:5000/nginx]
4cc0336d6a9: Pushed
0ba7659aae2e: Pushed
0cc7821cd3f3: Pushed
latest: digest: sha256:278fefc722ffelc36f6dd64052758258d441dcdb5elbbbed0670485af2413c9f size: 948
```

### 5.1.13 查看私有仓库镜像

命令 : curl -X GET <http://192.168.33.24:5000/v2/nginxbox/tags/list>

```
[root@swarm4 ~]# curl -X GET http://192.168.33.24:5000/v2/nginx/tags/list
[{"name": "nginx", "tags": ["latest"]}
[root@swarm4 ~]#
```

## 5.2 客户端操作

### 5.2.1 停止并关闭防火墙

```
systemctl stop firewalld.service
```

```
systemctl disable firewalld.service
```

### 5.2.2 安装并启动 docker

参照 4.2 的步骤进行操作

### 5.2.3 修改 Docker 配置文件

命令 : vi /etc/docker/daemon.json

```
{  
  "insecure-registries":["192.168.33.24:5000"],  
  "registry-mirrors": ["192.168.33.24:5000"]  
}
```

### 5.2.4 重启 docker 服务

命令 : systemctl restart docker

### 5.2.5 从私有仓库中拉取 nginx 镜像

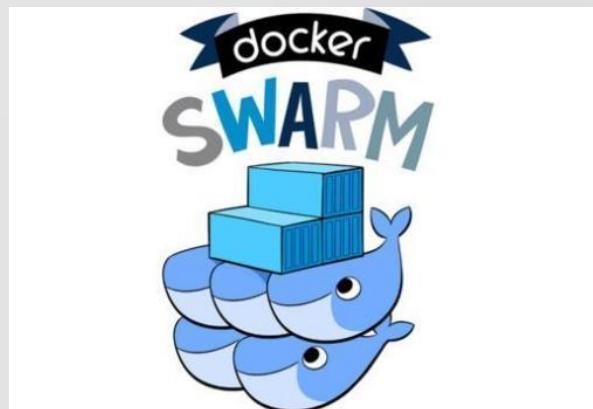
命令 : docker pull nginx && docker images

```
[root@swarm1 ~]# docker pull 192.168.33.24:5000/nginx  
Using default tag: latest  
latest: Pulling from nginx  
bc95e04b23c0: Pull complete  
a21d9ee25fc3: Pull complete  
9bda7d5afdf9: Pull complete  
Digest: sha256:278fefc722ffe1c36f6dd64052758258d441dcdb5e1bbbed0670485af2413c9f  
Status: Downloaded newer image for 192.168.33.24:5000/nginx:latest  
[root@swarm1 ~]# docker images  
REPOSITORY          TAG           IMAGE ID        CREATED       SIZE  
192.168.33.24:5000/nginx   latest        40960efd7b8f   2 weeks ago   108MB
```

至此 , docker 私有仓库搭建完成!

# 第六章

## 搭建 Docker Swarm 模式集群



Swarm 是 Docker 公司在 2014 年 12 月初发布的一套较为简单的工具，用来管理 Docker 集群，它将一群 Docker 宿主机变成一个单一的，虚拟的主机。Swarm 使用标准的 Docker API 接口作为其前端访问入口，换言之，各种形式的 Docker Client(docker client in go, docker\_py, docker 等)均可以直接与 Swarm 通信。

## 6、搭建 DockerSwarm 模式集群

### 6.1 集群规划

主机	IP	描述
Swarm1	192.168.33.21	Swarm 主节点
Swarm2	192.168.33.22	Swarm 从节点

### 6.2 所有节点操作

#### 6.2.1 安装 docker 的 CE 版本

参照 4.2 进行操作

#### 6.2.2 编辑 hosts 文件

命令 : vi /etc/hosts

192.168.33.21 swarm1

192.168.33.22 swarm2

### 6.3 创建 swarm 模式集群

#### 6.3.1 swarm 主节点操作(192.168.33.21)

##### 6.3.1.1 初始化集群

命令 : docker swarm init --advertise-addr 192.168.33.21

```
[root@swarm1 ~]# docker swarm init --advertise-addr 192.168.33.21
Swarm initialized: current node (uyyfd97x5yxtqqfl2rxpoqv4w) is now a manager.

To add a worker to this swarm, run the following command:
  docker swarm join --token SWMTKN-1-1odnpwns3owgzo7qm0ikssi1i01ikgcb5ldn4qns0ppt3b7ywq-bfaun39rnjcfpi9bbbshmclk0 192.168.33.21:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

在 docker swarm init 完了之后，会提示如何加入新机器到集群，如果当时没有注意到，也可以通过下面的命令来获知如何加入新机器到集群。

docker swarm join-token worker

### 6.3.2 swarm 从节点操作(192.168.33.22)

#### 6.3.2.1 加入新机器到集群

docker swarm join --token \

SWMTKN-1-

```
1odnpwns3owgzo7qm0ikssi1i01ikgcb5ldn4qns0ppt3b7ywq- \
bfaun39rnjcfpi9bbbshmclk0 192.168.33.21:2377
```

### 6.3.3 swarm 主节点操作(192.168.33.21)

#### 6.3.3.1 查看集群信息

命令：docker node ls，查看加入到 swarm 集群的节点。

```
[root@swarm1 ~]# docker node ls
ID          HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
uyyfd97x5yxtqqfl2rxpoqv4w *  swarm1    Ready   Active        Leader
gv7wejjjs09egeggj7w65xh5s      swarm2    Ready   Active
[root@swarm1 ~]#
```

至此，swarm 模式集群已经搭建完毕！

参考：

<http://www.cnblogs.com/gtarcoder/p/6419033.html>

# 第七章

## 安装 jenkins



Jenkins 是一个开源软件项目，是基于 Java 开发的一种持续集成工具，允许持续集成和持续交付项目，无论用的是什么平台。这是一个免费的源代码，可以处理任何类型的构建或持续集成。集成 Jenkins 可以用于一些测试和部署技术。

## 7、安装 jenkins

### 7.1 安装 jenkins 服务器(192.168.33.23)

#### 7.1.1 软件需求

- jdk-8u152-linux-x64.tar.gz (自行下载并上传)
- apache-maven-3.5.2-bin.tar.gz (自行下载并上传)
- jenkins 的 war 包

#### 7.1.2 下载 Jenkins 的 war 包

访问 <https://jenkins.io/download/>, 下载 Long-term Support (LTS) 版本的 war 包

#### 7.1.3 上传软件

把下载好的三个软件上传至 192.168.33.23 上

### 7.1.4 解压软件

```
cd /opt  
tar -xvf jdk-8u152-linux-x64.tar.gz  
tar -xvf apache-maven-3.5.2-bin.tar.gz
```

### 7.1.5 添加环境变量

命令 : vi /etc/profile

```
export JAVA_HOME=/opt/jdk1.8.0_152  
export MAVEN_HOME=/opt/apache-maven-3.5.2  
export PATH=$PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin  
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

命令 : source /etc/profile

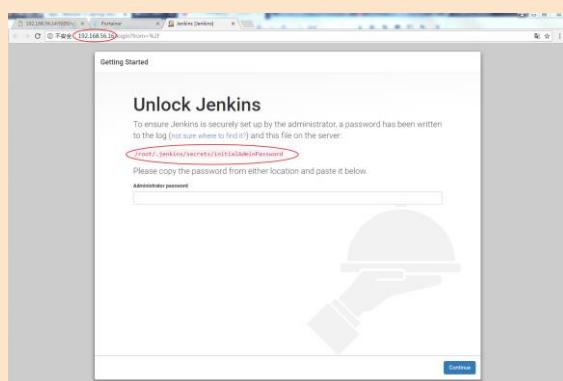
执行完成之后，用下面两个命令验证一下：

```
java -version  
mvn -v
```

## 7.2 启动 jenkins

```
mkdir jenkins  
mv jenkins.war jenkins/jenkins.war  
cd jenkins  
nohup java -jar jenkins.war --httpPort=80 >/dev/null &
```

在浏览器中访问 <http://192.168.33.23>

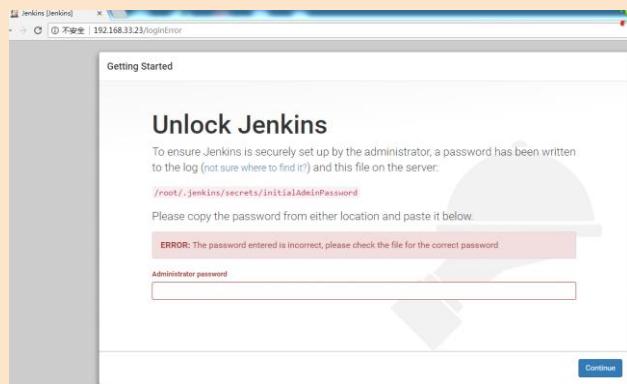


### 7.3 初始化 jenkins

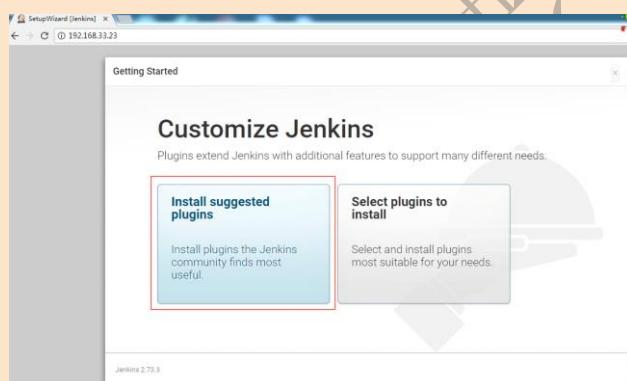
初次使用，显示了 Administrator 密码的文件位置，通过以下命令查看初始密码 cat /root/.jenkins/secrets/initialAdminPassword

```
[root@swarm6 jenkins]# cat /root/.jenkins/secrets/initialAdminPassword  
ba8b53bc23a34307a0e8a181eda4b8e9  
[root@swarm6 jenkins]#
```

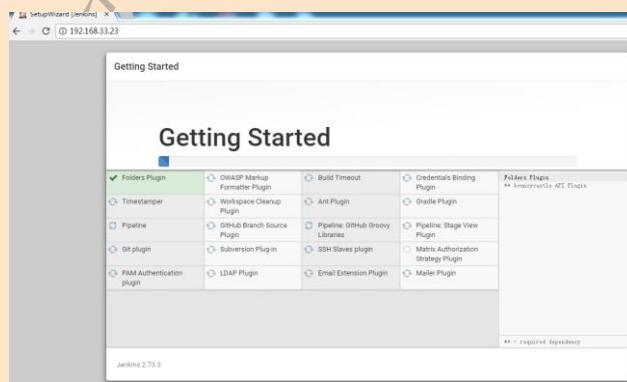
输入找到的密码后，



首次进入会弹出插件安装界面



点击安装常用的插件后



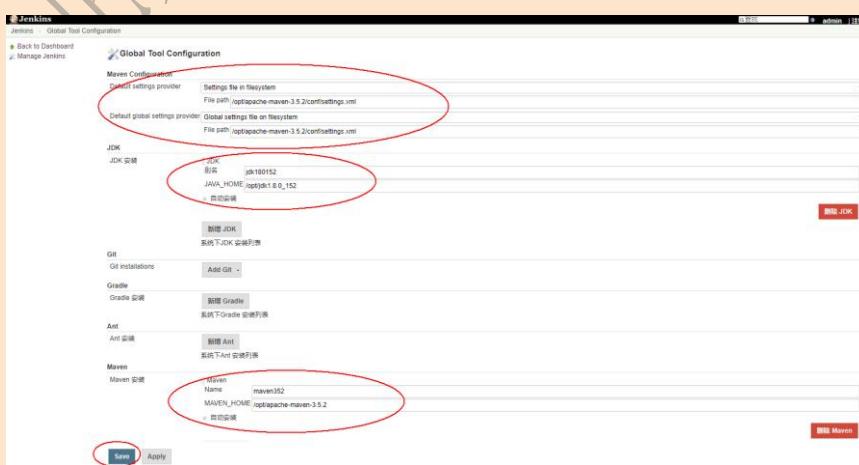
插件安装完成之后，点击“contine”进入主界面



## 7.4 配置 jenkins 的 Jdk 和 Maven



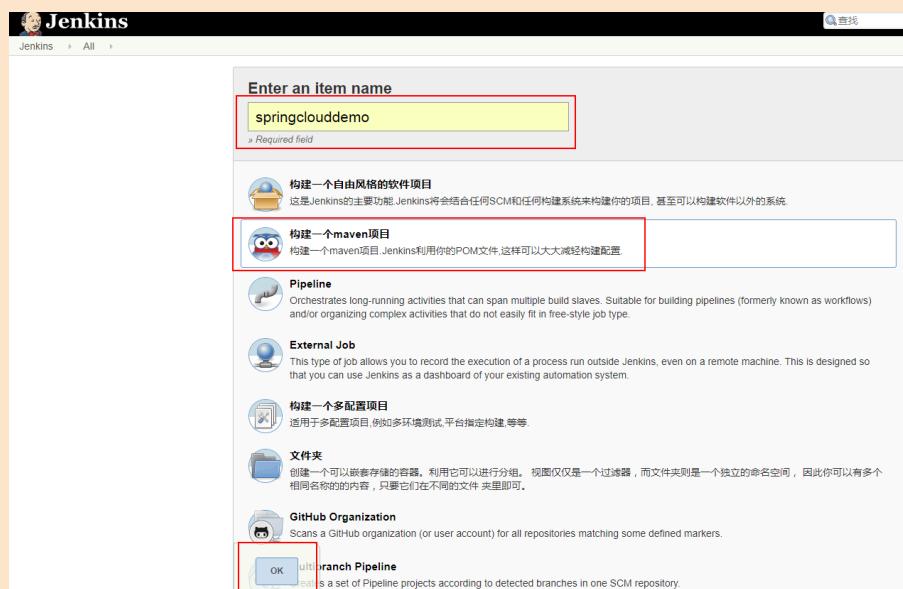
点击“系统管理”=>“全局设置”，按图配置，并保存



## 7.5 体验 jenkins

### 7.5.1 构建一个 maven 项目

填写项目名称，选择构建一个 maven 项目



源码管理部分，选择 subverion，输入项目地址，此时我们需要添加 SVN

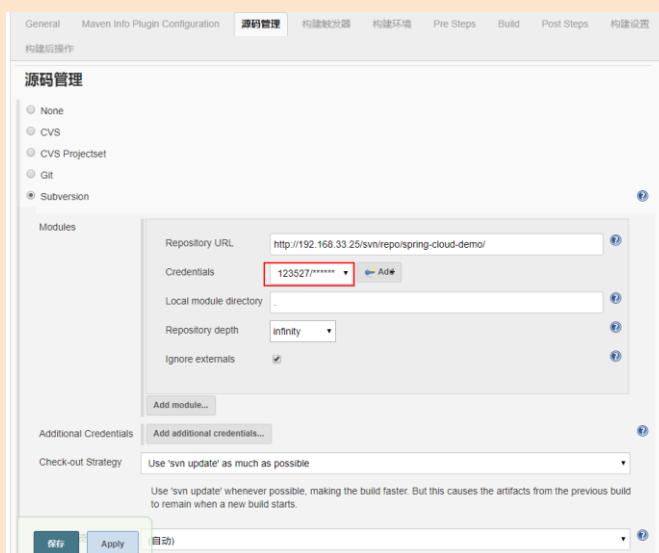
账号信息，选择“ADD”

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

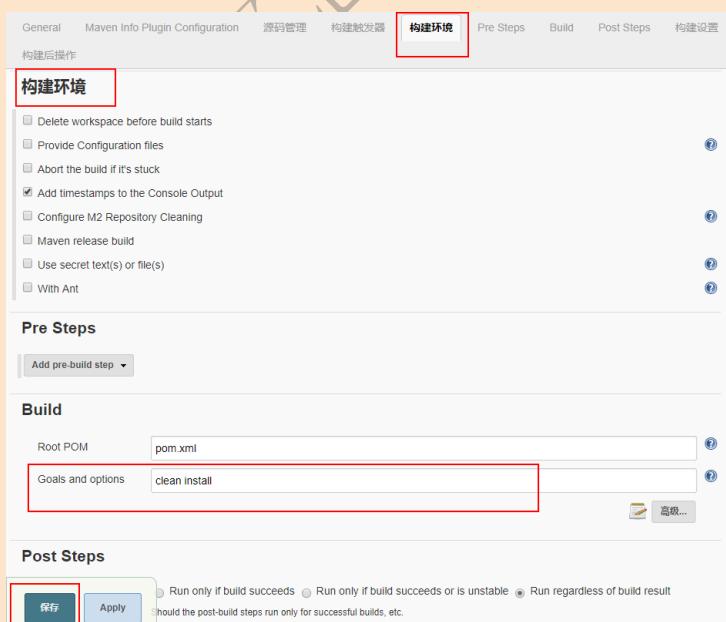
在弹出的页面，输入 SVN 账号信息，并点击“ADD”



现在我们就可以选择刚刚添加的 SVN 的账号信息了



在“Build”中，输入“clean install”，最后点击“保存”，OK。



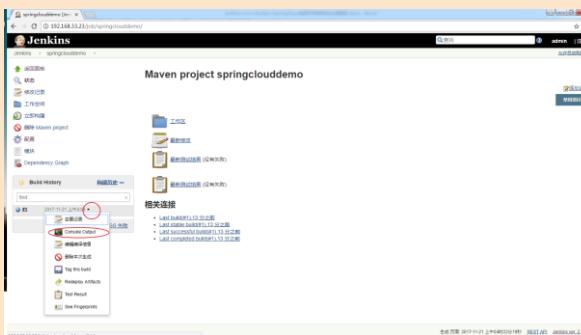
# Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

按照上面的步骤，我们已经创建了一个“Maven 构建项目”，如下所示：



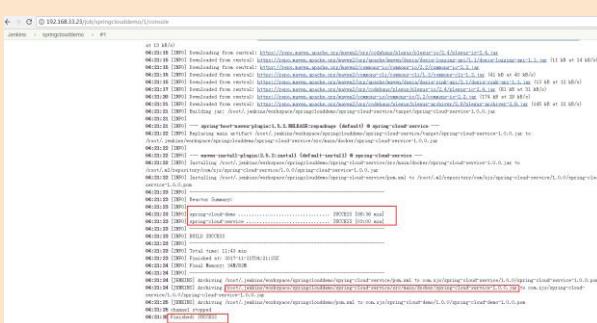
点击上图中的“立即构建”，jenkins 就开始为我们执行了，按下图选择，

即可看到构建的整个过程



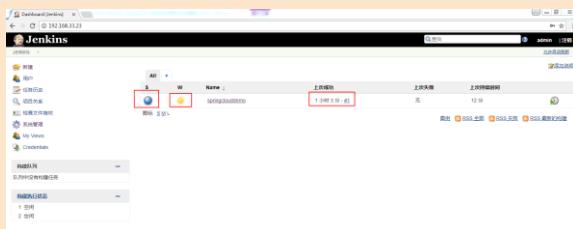
这就是我们看到的 jenkins 构建过程日志信息，非常的清晰

最后，我们的项目构建成功了！



## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

主界面也能看到相应的信息



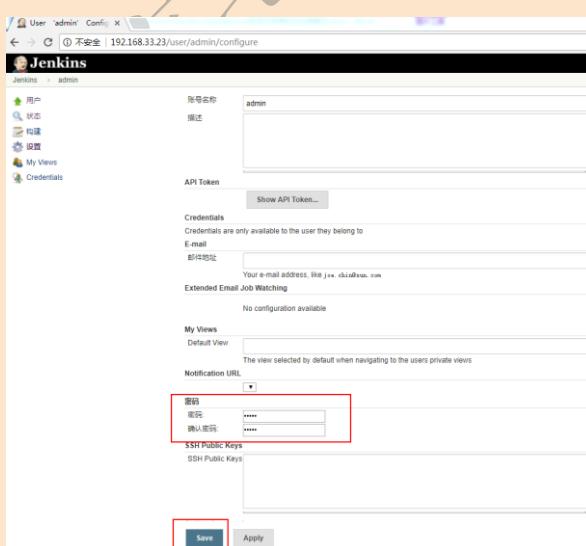
接下来，我们还可以修改一下 admin 的登录密码



点击设置

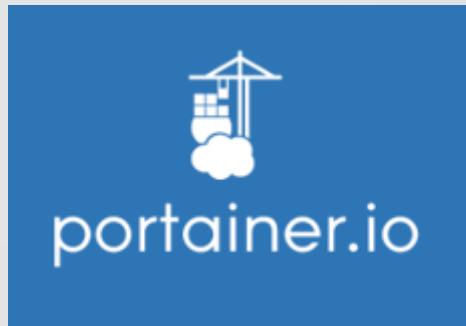


输入新的密码，保存即可。



# 第八章

## 安装 Protainer



## 8、搭建 Portainer 可视化界面

### 8.1 swarm 主节点操作(192.168.33.21)

#### 8.1.1 下载 portainer 镜像

命令 : docker pull portainer/portainer

#### 8.1.2 启动 portainer

命令 :

```
docker service create --name portainer --publish 9000:9000 \
--constraint 'node.role == manager' \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
portainer/portainer -H unix:///var/run/docker.sock
```

```
[root@swarm1 ~]# docker service create --name portainer --publish 9000:9000 --constraint 'node.role == manager' --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock portainer/portainer -H unix:///var/run/docker.sock
0jqfppd6y4fa5bb20zdtkzhah
Since --detach-raise was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
[root@swarm1 ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
f4de6707a65b7      portainer/portainer:latest   "/portainer -H uni..."   13 seconds ago    Up 13 seconds     9000/tcp            portainer.1.v02bdog6xqothba8q1clxt9wh
```

#### 8.1.3 查看 docker 进程

命令 : docker ps -a

```
[root@k8s2 ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
92ee239b49cfa      192.168.33.21:5000/portainer:latest   "/portainer -H uni..."   About an hour ago  Up About an hour   9000/tcp            portainer.1.7avv0biqjj7ip
n422cxd7ccjd
```

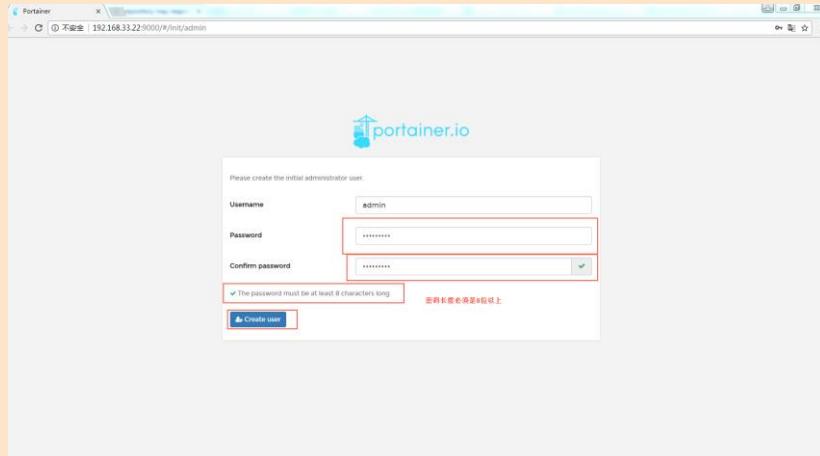
已成功启动 !

### 8.2 浏览 portainer 界面

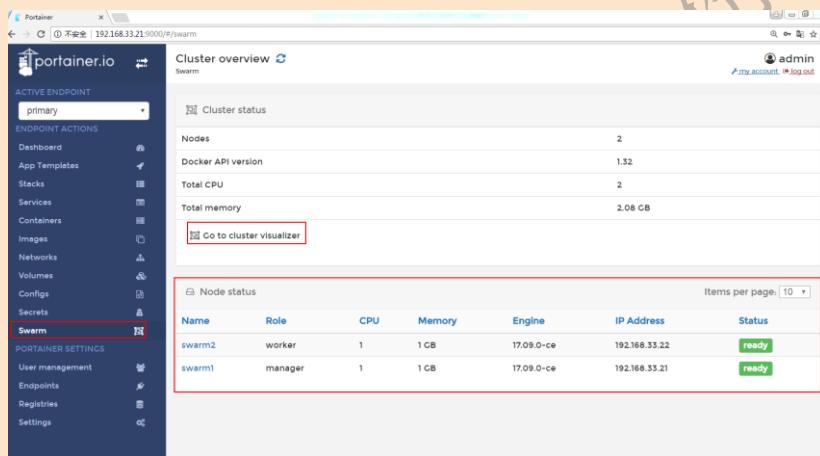
在浏览器中访问 <http://192.168.33.21:9000> , 终于可以看到 portainer 界面了。

### 8.2.1 设置密码

第一次使用需要设置 admin 用户的密码。

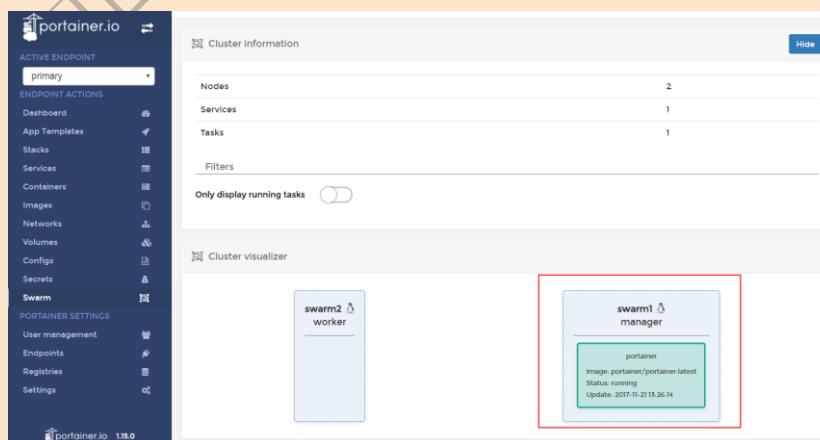


我们看一下集群信息，就是第六章里面创建的 docker swarm 模式集群，



Name	Role	CPU	Memory	Engine	IP Address	Status
swarm2	worker	1	1 GB	17.09.0-ce	192.168.33.22	ready
swarm1	manager	1	1 GB	17.09.0-ce	192.168.33.21	ready

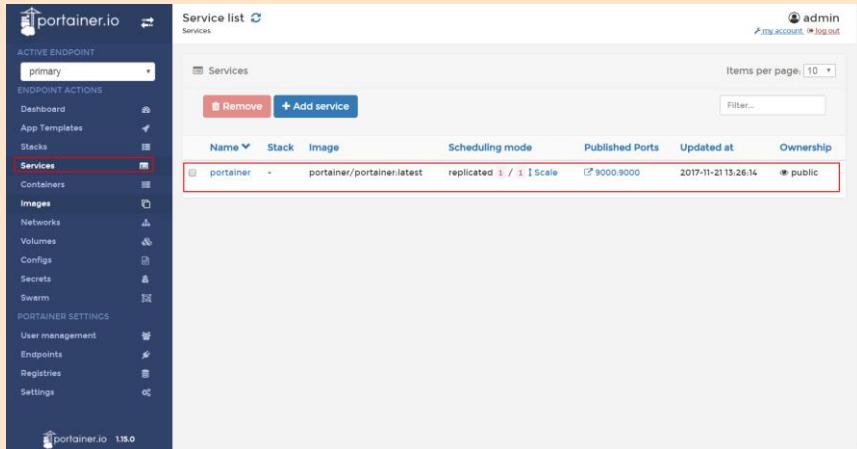
点击“Go to cluster visualizer”，我们就可以很直观的看到集群的状态已经运行的容器信息



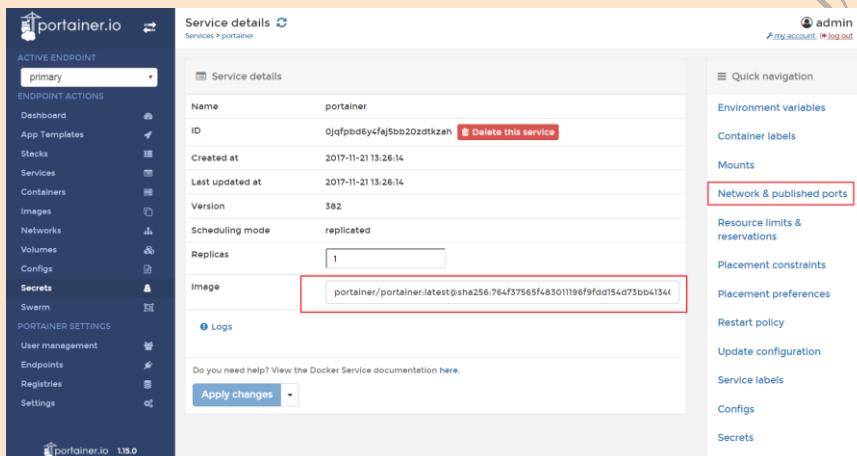
portainer	Image: portainer/portainer:latest	Status: running	Update: 2017-11-21 13:26:14
-----------	-----------------------------------	-----------------	-----------------------------

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

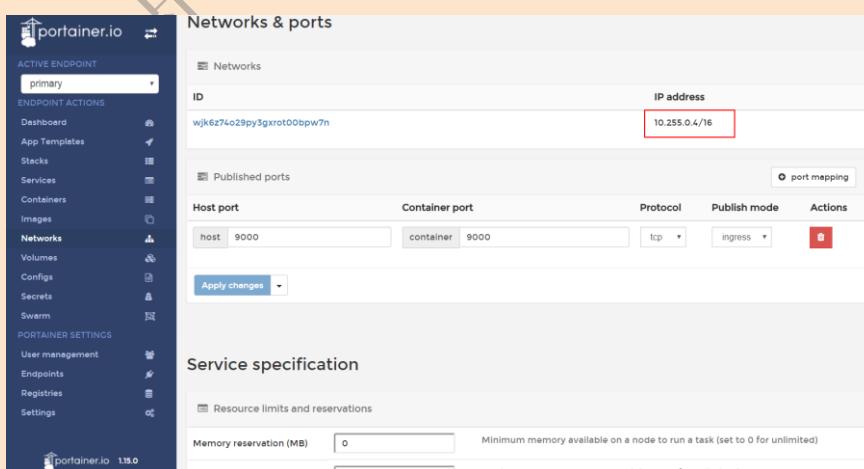
### 查看 service



点击当前的 service , 接下来的界面就是该服务的相信信息



Portainer 是 Docker swarm 的管理界面 , 执行的都是 swarm 相关的指令。而 swarm 一个重要的内容就是网络相关的信息 , 点击上面的 “Network & published ports” , 我们就能清楚的了解该服务的网络相关的信息。



## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

上图中有一个 ID : **wjk6z74o29py3gxrot00bpw7n** , 这是怎么产生的呢 ?

我们看一下 swarm 的网络 , 在里面可以看见 “ingress” , 它就是 swarm 自动为容器创建的一张虚拟网。如无指定具体网络的情况下 , 这就是容器的默认网段。

The screenshot shows the Portainer.io interface under the 'Networks' section. It lists several networks: bridge, docker\_gwbridge, host, ingress, and none. The 'ingress' network is highlighted with a red border. The table columns include Name, Stack, Scope, Driver, IPAM Driver, IPAM Subnet, IPAM Gateway, and Ownership.

Name	Stack	Scope	Driver	IPAM Driver	IPAM Subnet	IPAM Gateway	Ownership
bridge	-	local	bridge	default	172.17.0.0/16	172.17.0.1	public
docker_gwbridge	-	local	bridge	default	172.18.0.0/16	172.18.0.1	public
host	-	local	host	default	-	-	public
<b>ingress</b>	-	swarm	overlay	default	10.255.0.0/16	10.255.0.1	public
none	-	local	null	default	-	-	public

通过 “ip 子网掩码计算器” 工具 , 可以计算这个网段的可用 VIP( 虚拟 IP )数量。

The screenshot shows an IP subnet calculator tool. The input field 'IP/掩码位' contains '10 255 0 1 /16'. The result table shows the following information:

可用地址	65534	这里就是可用的VIP个数
掩码:	255 255 0 0	
网络:	10 255 0 0	
第一个可用:	10 255 0 1	
最后可用:	10 255 255 254	
广播:	10 255 255 255	

说明: 在网络掩码“位格式”也被称为CIDR格式 (CIDR=无类别域间路由选择)。

点击 “ingress” , 是否看到 Portainer ? 它的 vip : 10.255.0.5 , 就是上面网段的一个 VIP。

The screenshot shows the detailed configuration of the 'ingress' network. Key settings include:

- ID: wjk6z74o29py3gxrot00bpw7n
- Driver: overlay
- Scope: swarm
- Subnet: 10.255.0.0/16
- Gateway: 10.255.0.1
- Access control: Ownership (public)
- Network options: com.docker.network.driver.overlay.xxlan0\_list
- Containers in network: portainer (IP: 10.255.0.5/16, MacAddress: 02:43:0e:ff:00:05)

这个网络是 swarm 自行创建的 , 可以支持创建 65534 个 VIP , 一个网络达到这样的量级 , 那是非常可怕的一件事情。也就是每台机器的网络信息要在 65534 上共享 , 性能非常差 ! 在生产环境中 , 一定要创建贴合实际的网络 , 避免因此而影响集群的性能。

## 8.3 Portainer 体验

### 8.3.1 创建 nginx 服务

#### 8.3.1.1 查看镜像信息

The screenshot shows the Portainer.io dashboard with the 'Images' tab selected. The main area displays a table of images. One row for the 'nginx:latest' image has a red circle around its 'Remove' button.

ID	Tags	Size	Created
sha256:4c09e0d6...	portainer/portainer:latest	33.1 MB	2017-11-13 17:20:05
sha256:4090e0d9...	192.168.51.24:5000/nginx:latest	108.4 MB	2017-11-05 02:41:09

#### 8.3.1.2 创建自定义网络

The screenshot shows the Portainer.io dashboard with the 'Networks' tab selected. The main area displays a table of networks. A red circle highlights the '+ Add network' button at the top of the table.

Name	Stack	Scope	Driver	IPAM Driver	IPAM Subnet	IPAM Gateway	Ownership
bridge	-	local	bridge	default	172.17.0.0/16	172.17.0.1	* public
docker_gwbridge	*	local	bridge	default	172.18.0.0/16	172.18.0.1	* public
host	-	local	host	default	-	-	* public
ingress	*	swarm	overlay	-	10.255.0.0/16	10.255.0.1	* public
none	-	local	null	default	-	-	* public

我们创建一个“10.254.255.0/16”的网络，避免与“10.255.0/16”冲突，之后  
点击“Create the network”

The screenshot shows the 'Create network' dialog. The 'Name' field is set to 'net\_nginx'. The 'Driver' dropdown is set to 'overlay'. In the 'Labels' section, there is a label 'name: nginx'. The 'Restrict external access to the network' toggle switch is turned off. In the 'Access control' section, the 'Administrators' checkbox is checked.

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

在主界面的“Network list”中，我们可以看到刚刚创建的网络，但是好像有点问题，“ownership”不是public。

The screenshot shows the Portainer.io interface under the 'Networks' section. A table lists various Docker networks. One row for 'net\_nginx' is highlighted with a red box. The 'Ownership' column for this network shows 'administrators'. Other networks like 'bridge', 'docker\_gwbridge', 'host', and 'ingress' have 'public' selected.

点击“net\_nginx”，可以进行修改。

The screenshot shows the 'Network details' page for 'net\_nginx'. In the 'Ownership' section, a dropdown menu is open, showing three options: 'Administrators', 'Restricted', and 'Public'. The 'Public' option is highlighted with a red box.

选择“public”，点击“update ownership”即可。

The screenshot shows the 'Network details' page for 'net\_nginx' after selecting 'Public' ownership. The 'Update ownership' button at the bottom left is highlighted with a red box.

再次刷新“Network list”，新创建的网络就是我们想要的。

The screenshot shows the 'Network list' page again. The 'net\_nginx' network is now listed with 'public' selected in the 'Ownership' column, indicating the change was successful.

## 8.3.1.3 部署 nginx 服务

接下来，我们以上面新建的网络来部署 nginx 服务

Name	Stack	Image	Scheduling mode	Published Ports	Updated at	Ownership
portainer	portainer	portainer/portainer:latest	replicated	9000:9000	2017-11-21 13:26:14	public

填写相关的一些参数

**Name:** nginx

**Image configuration:**

**Name:** nginx:latest      **Registry:** DockerHub

**Scheduling:**

**Scheduling mode:** Global    Replicated

**Replicas:** 2

**Ports configuration:**

**Port mapping:** host 8000 → container 80    TCP    Ingress    Host

**Access control:**

**Enable access control:**

**Administrators:** I want to restrict the management of this resource to administrators only

**Restricted:** I want to restrict the management of this resource to a set of users and/or teams

**Actions:**

**Port mapping:** host 8000 → container 80    TCP    UDP    Ingress    Host

**Access control:**

**Enable access control:**

**Administrators:** I want to restrict the management of this resource to administrators only

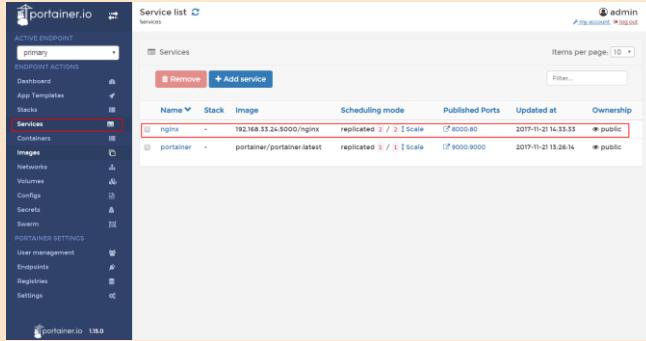
**Restricted:** I want to restrict the management of this resource to a set of users and/or teams

**Network:** net\_nginx

**Extra networks:**

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

按照上面的内容填写好之后，点击“Create service”，稍等片刻之后，在“service”界面就能看见刚刚创建的服务了。



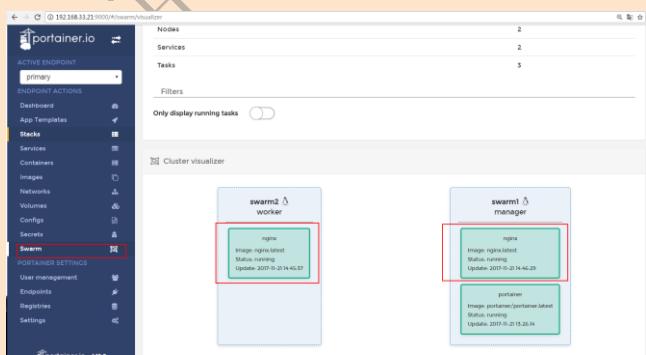
在浏览器中访问 <http://192.168.33.21:8000>



在浏览器中也可以访问 <http://192.168.33.22:8000>



查看一下集群的状态，不难发现 2 个容器成功的分布到集群的节点上了。



Portainer 的功能还是非常全面的，赞一个！其他的功能自行尝试吧。

# 第九章

# SpringBoot 快速入门



Spring 家族发展到今天，已经很庞大了。作为一个开发者，想使用这一系列的技术，需要一个一个的搞配置，然后还有个版本兼容性问题，其实挺麻烦的，偶尔也会出现小坑，影响着开发进度。

Spring Boot 提供可插拔的设计，就是各种 starter，把一些常用的基础框架组合起来，提供默认的配置，方便使用。开发者可以先不关心如何配置，快速启动开发，编写业务逻辑。

Spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具，它为基于 JVM 的云应用开发中的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等操作提供了一种简单的开发方式。

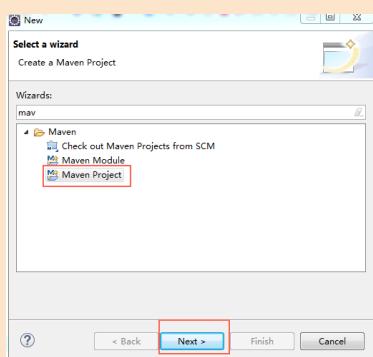
## 9、Spring Boot 快速入门

### 9.1 搭建第一个 Spring Boot 程序

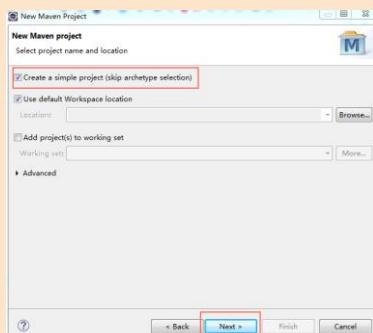
可以在 start.spring.io 上建项目，也可以用 Eclipse 构建。这里采用 Eclipse。

#### 9.1.1 创建父项目

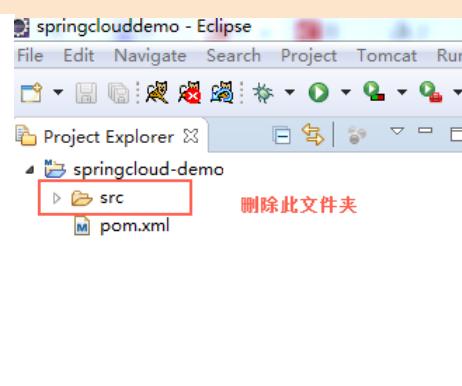
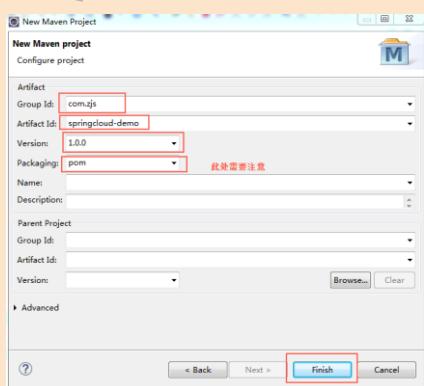
打开 Eclipse ,点击 File->New->Other,在弹出对话框中 选中 Maven Project.



点击 Next 按钮 , 出现下图 , 在弹出界面中选择 “Create a simple project”



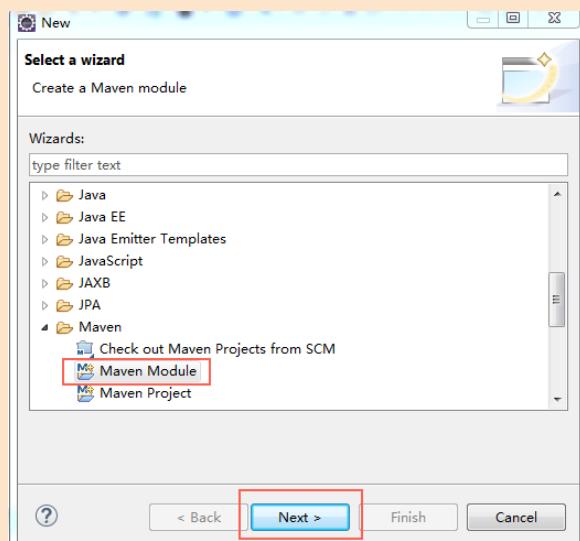
设置工程的参数 , 见下图



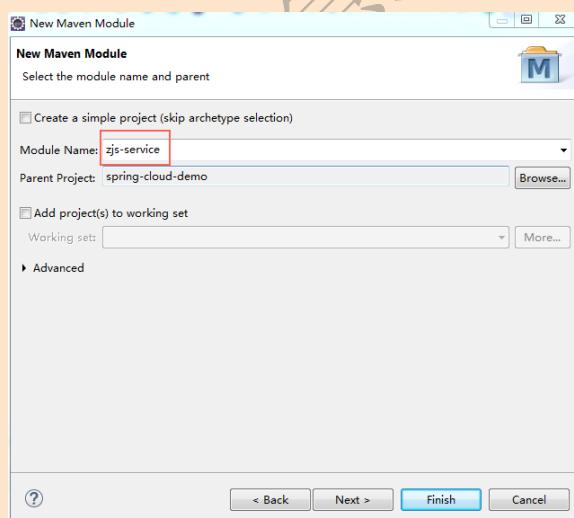
这样，我们按常规模版创建了一个 Maven 工程。因为，这是一个父项目，不需要有什么源码。那么我们在 Eclipse 中将这个工程下的不用的目录都删除，仅留下 pom.xml 文件就行了。

### 9.1.2 创建子项目

选中刚建的父项目，在弹出菜单中点击 New -> Maven Module;

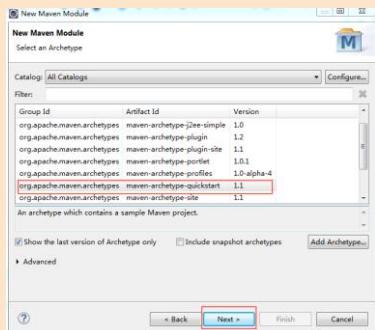


如图配置

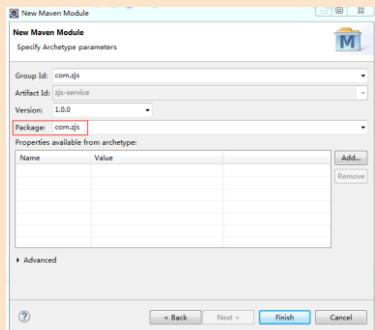


使用默认的 Archetype (默认：GroupId:org.apache.maven.archetypes,Artifact

Id:maven-archetype-quickstart )

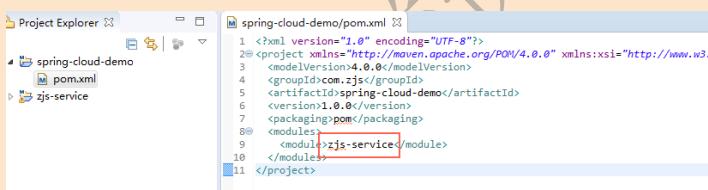


完成工程配置，见下图



这样一个子项目就创建完成了，在文件系统中，子项目会建在父项目的目录中。在父目录中运行 mvn test 等命令，所有的子项目都会按顺序执行。

细心一点的人，可能会发现，通过这个步骤创建子项目的同时，会修改父项目的 pom.xml，增加了类似下面的信息：



这个信息，就是标记有哪些子模块。重复创建子项目的步骤，可以创建多个子项目。

按上面步骤创建的子项目，在 pom.xml 中有个 parent 节点，所以，他可以继承父项目的相关信息。没错，父子项目中存在继承关系。

在子项目的 pom.xml 中，子项目的 groupId 和 version 一般和父项目相同，那么可以把子项目的这两个参数删除，这样会自动继承父项目的取值。

同样，如果其他的一些属性，所有子项目都是一样的，那么可以上移到父项

目中设置，子项目中无需重复设置。比如：

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

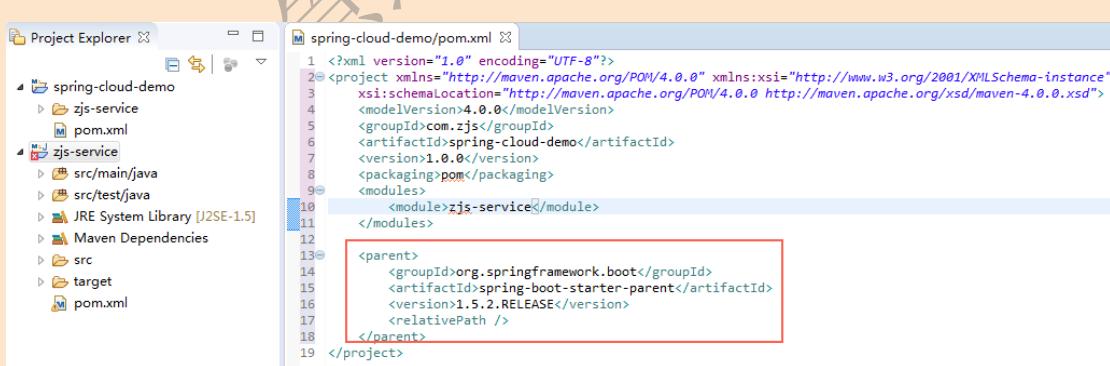
可以仅在父项目中设置一次。

除了这种情况以外，还有一种情况，就是依赖和插件。依赖和插件的情况是这样，某一个依赖或插件可能会被大部分子项目所使用，但是也可能有些子项目不需要使用，这样使用上述的方式，简简单单地进行继承就不合适了。

Manen 提供 dependencyManagement 和 pluginManagement 两个标签。使用这两个标签，可以在父项目中统一管理依赖和插件的配置参数，比如版本号啥的。而在子项目中，仅需列出需要使用的依赖和插件的 groupId 和 artifactId 就可以了，其他信息会自动从父项目管理的信息里面获取。

### 9.1.3 编写 pom.xml

在 parent 部分使用 spring-boot-starter-parent。spring-boot-starter-parent 是重要的默认的父工程，它提供了 dependency-management 部分。



#### 添加 spring-boot-starter-web 依赖

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```

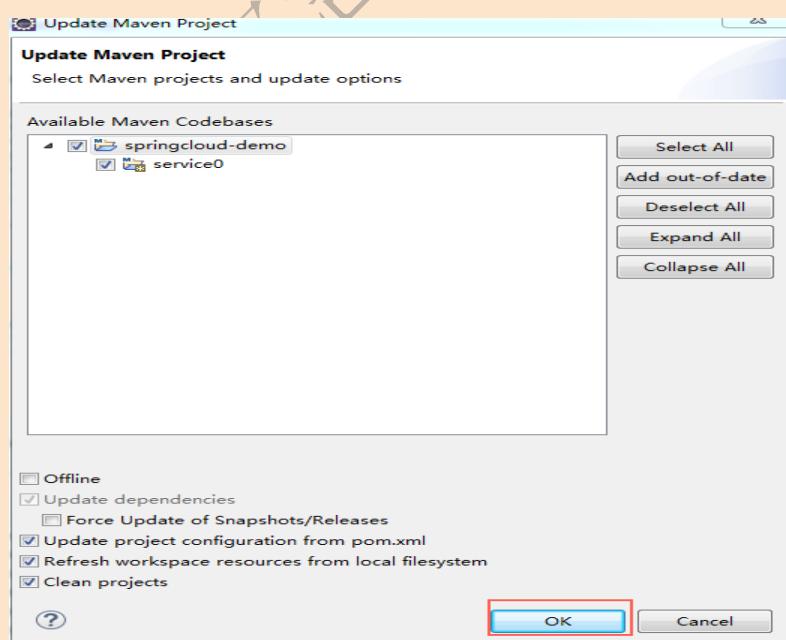
完整 POM 文件如下：

```
10      <module>zjs-service</module>
11  </modules>
12
13  <parent>
14      <groupId>org.springframework.boot</groupId>
15      <artifactId>spring-boot-starter-parent</artifactId>
16      <version>1.5.2.RELEASE</version>
17      <relativePath />
18  </parent>
19
20  <dependencies>
21      <dependency>
22          <groupId>org.springframework.boot</groupId>
23          <artifactId>spring-boot-starter-web</artifactId>
24      </dependency>
25
26      <dependency>
27          <groupId>junit</groupId>
28          <artifactId>junit</artifactId>
29          <version>3.8.1</version>
30          <scope>test</scope>
31      </dependency>
32  </dependencies>
33  <build>
34      <plugins>
35          <plugin>
36              <groupId>org.springframework.boot</groupId>
37              <artifactId>spring-boot-maven-plugin</artifactId>
38          </plugin>
39
40          <plugin>
41              <groupId>org.apache.maven.plugins</groupId>
42              <artifactId>maven-compiler-plugin</artifactId>
43              <version>3.6.0</version>
44              <configuration>
45                  <source>1.8</source>
46                  <target>1.8</target>
47                  <encoding>UTF-8</encoding>
48              </configuration>
49          </plugin>
50      </plugins>
51  </build>
52 </project>
```

## 9.1.4 更新 maven 项目

上述操作之后，项目有报错信息，按照提示，我们更新一下 maven 项目。

在父项目上右击，选择 Maven->Update Project (如项目无报错，则不需要)，在弹出界面选择 OK。



### 9.1.5 便捷配置

maven 是个项目管理工具，如果我们不告诉它我们的代码要使用什么样的 jdk 版本编译的话，它就会用 maven-compiler-plugin 默认的 jdk 版本来进行处理，这样就容易出现版本不匹配的问题，以至于可能导致编译不通过的问题。例如代码中要是使用上了 jdk1.7 的新特性，但是 maven 在编译的时候使用的是 jdk1.6 的版本，那这一段代码是完全不可能编译成.class 文件的。为了处理这种情况的出现，在构建 maven 项目的时候，第一步就是配置 maven-compiler-plugin 插件。

compiler 插件能解决：

- maven 2.1 默认用 jdk 1.3 来编译，maven 3 貌似是用 jdk 1.5，如果项目用的 jdk 1.6 也会有问题，compiler 插件可以指定 JDK 版本为 1.6。
- windows 默认使用 GBK 编码，java 项目经常编码为 utf8，也需要在 compiler 插件中指出，否则中文乱码可能会出现编译错误。

pom 增加插件配置

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <encoding>UTF-8</encoding>
  </configuration>
</plugin>
```

接下来就是配置 spring-boot-maven-plugin 插件。

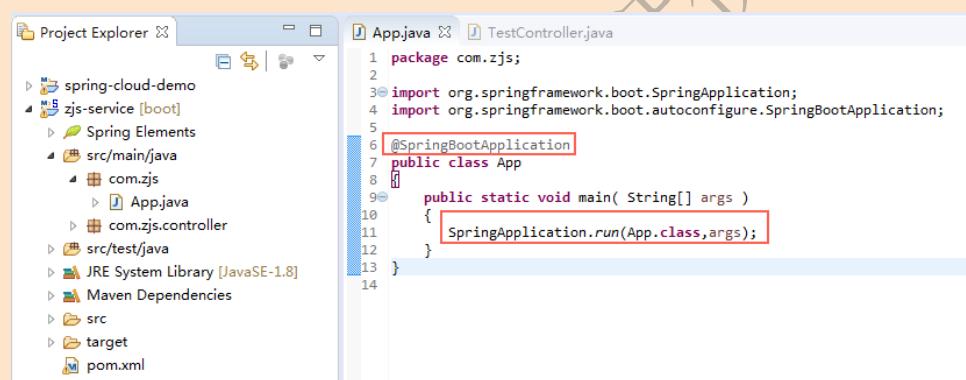
Spring boot 加上这个插件，才可以使用 Java -jar 命令来启动 jar 包，并且有了这个插件，打的包里面才会有 maven 依赖的 jar 包和 spring boot 的启动类，而且 MANIFEST.MF 文件里面也会有启动类的信息。但是如果不去加这个插件，则

打的包里面就只有 class 文件，没有依赖的 Jar 包，MANIFEST.MF 文件里面也没有启动类的信息，所以如果不加这个插件就不能独立启动。

**注意：在用 Eclipse 调试的时候加不加插件都可以启动，看不出来不同，所以必须要独立启动 jar 包才可以看出来。而且如果用了 Spring Boot 但是不需要独立启动，就不要加这个插件，否则 Spring Boot 会因为找不到启动类而导致报错。**

### 9.1.6 测试 Spring Boot 应用

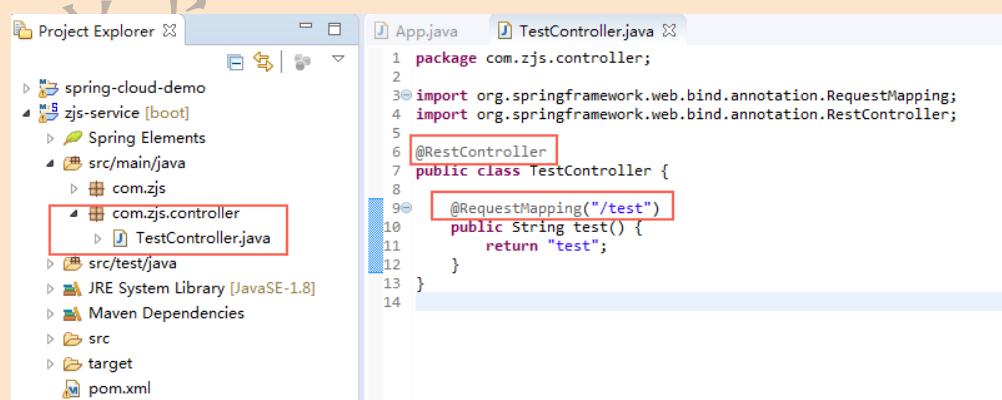
在 zjs-service 子项目的 com.zjs 包下有一个 App 类，我们把它改造成启动类，编辑后如下：



The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, which lists the project structure: spring-cloud-demo, zjs-service [boot], Spring Elements, src/main/java (containing com.zjs with App.java) and com.zjs.controller, src/test/java, JRE System Library [JavaSE-1.8], Maven Dependencies, src, target, and pom.xml. On the right is the code editor for App.java, which contains the following code:

```
1 package com.zjs;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class App {
8
9     public static void main( String[] args )
10    {
11        SpringApplication.run( App.class, args );
12    }
13 }
```

接下来，我们再添加一个 com.zjs.controller 包，然后在包下面创建 TestController 类

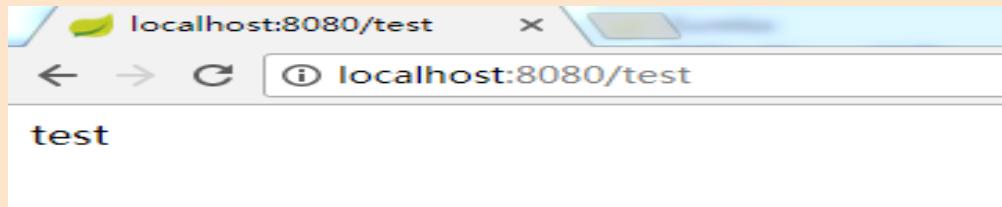


The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, which lists the project structure: spring-cloud-demo, zjs-service [boot], Spring Elements, src/main/java (containing com.zjs with com.zjs.controller and TestController.java), and com.zjs.controller. On the right is the code editor for TestController.java, which contains the following code:

```
1 package com.zjs.controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class TestController {
8
9     @RequestMapping( "/test" )
10    public String test()
11    {
12        return "test";
13    }
14 }
```

启动程序，右键App.java文件，选择run as -> Java Application。

如果没有报错的话，在浏览器中输入：<http://localhost:8080/test>



## 9.2 单元测试

创建一个普通的Java类，在Junit4中不再需要继承TestCase类了。因为我们是Web项目，所以在创建的Java类中添加注解：

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes=App.class)
```

接下来就可以编写测试方法了，测试方法使用@Test注解标注即可。

在该类中我们可以像平常开发一样，直接@Autowired注入要测试的类实例。

下面是完整代码：

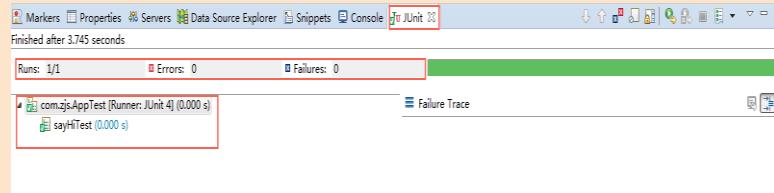
```
1 package com.zjs;
2
3 import org.junit.Test;
4 import org.junit.runner.RunWith;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
8
9 import com.zjs.controller.TestController;
10
11 @RunWith(SpringJUnit4ClassRunner.class)
12 @SpringBootTest(classes=App.class)
13 public class AppTest {
14
15     @Autowired
16     TestController testController;
17
18     @Test
19     public void test() {
20         String testResult=testController.test();
21
22         System.out.println(testResult);
23     }
24 }
```

之后，我们来运行一下，选中AppTest.java,选择Run As->Junit Test，测试结果图如下：

Console面板：

```
2017-10-18 15:07:33.284 INFO 8080 --- [           main] com.zjs.AppTest : Starting AppTest on FC290ZHZLYLSUL with PID 8080
2017-10-18 15:07:33.284 INFO 8080 --- [           main] com.zjs.AppTest : No active profile set, falling back to default profiles
2017-10-18 15:07:33.523 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Refreshing org.springframework.web.context.support.GenericWebApplicationContext@533553: startup date [2017-10-18T15:07:33.523+08:00]; root of context hierarchy
2017-10-18 15:07:33.697 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Initialising StandardEnvironment
2017-10-18 15:07:33.700 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Initialising StandardEnvironment with PropertySources [systemProperties,systemProperties,systemProperties]
2017-10-18 15:07:33.700 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Adding [inline] PropertySource with highest search precedence
2017-10-18 15:07:33.700 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Adding inlined properties to environment: {spring.jmx.enabled=true}
2017-10-18 15:07:33.700 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Adding [Inlined Test Properties] PropertySource with highest search precedence
2017-10-18 15:07:33.700 DEBUG 8080 --- [           main] o.s.w.c.s.GenericWebApplicationContext : Adding inlined properties to environment: {spring.jmx.enabled=true}
2017-10-18 15:07:36.366 INFO 8080 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Mapped "{[/test]}" onto public java.lang.String com.zjs.controller.TestController.test()
2017-10-18 15:07:36.372 INFO 8080 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{"error"}]" onto public org.springframework.http.ResponseEntity<org.springframework.util.MultiValueMap<java.lang.String,java.util.List<java.lang.String>>> org.springframework.web.error.ErrorHandler.handleError(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2017-10-18 15:07:36.433 INFO 8080 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-10-18 15:07:36.433 INFO 8080 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.DispatcherServlet]
2017-10-18 15:07:36.830 INFO 8080 --- [           main] com.zjs.AppTest : Started AppTest in 4.108 seconds (JVM running for 4.108 seconds)
2017-10-18 15:07:36.900 INFO 8080 --- [           Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.Generi
```

Junit面板：

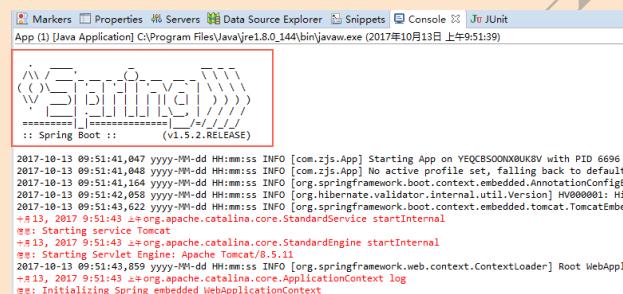


OK，我的第一个Spring Boot程序已经问世了！比起之前的Spring开发，是否简化了不少了？这里只是给出一个很简单的入门教程，Spring Boot的功能还是很强大的，值得慢慢研究！

## 9.3 优化配置

### 9.3.1 定制 Banner

在启动Spring Boot项目时，控制台会默认输出一个启动图案，如下：



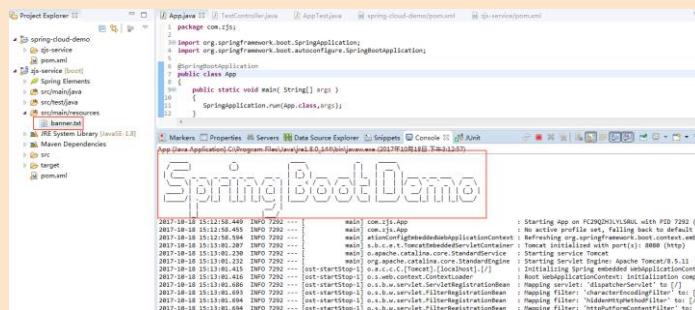
当然，这个图案是可以修改的，修改方式很简单：

- 在 src/main/resources 下新建一个 banner.txt 文档
- 通过 <http://patorjk.com/software/taag> 网站生成所需字符，将字符拷贝到步骤 1 所创建的 txt 文档中，比如我这里为 Spring Boot Demo 生成字符，如下：



点击左下角的选择和拷贝按钮，将这个字符拷贝到txt文档中，然后再启动项目，

这个时候控制台输出的文本就会自动改变，如下：



### 9.3.2 关闭 Banner

可以修改当然也可以关闭，关闭Banner需要我们稍微修改一下main方法中的代

码，如下：

```

1 package com.zjs;
2
3 import org.springframework.boot.Banner;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.boot.builder.SpringApplicationBuilder;
6
7 @SpringBootApplication
8 public class App
9 {
10     public static void main( String[] args )
11     {
12         // SpringApplication.run(App.class,args);
13         SpringApplicationBuilder builder=new SpringApplicationBuilder(App.class);
14         builder.bannerMode(Banner.Mode.OFF).run(args);
15     }
16 }

```

### 9.3.3 配置文件

Spring Boot 使用一个全局的配置文件 application.properties 或者 application.yml，配置文件放在src/main/resources目录下。properties是我们常见的一种配置文件，Spring Boot不仅支持properties这种类型的配置文件，也支持yaml语言的配置文件，我这里以properties类型的配置文件为例来看几个案例。

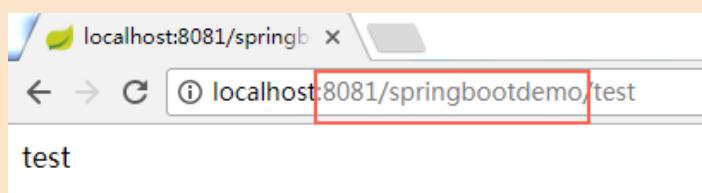
### 9.3.3.1 修改 Tomcat 默认端口和默认访问路径

Tomcat 默认端口是 8080 , 我将之改为 8081 , 默认访问路径是 <http://localhost:8080> , 我将之改为<http://localhost:8081/helloboot>, 我们来看看这两个需求要怎么样通过简单的配置来实现。

很简单，在application.properties文件中添加如下代码：

```
server.context-path=/springbootdemo
server.port=8081
```

然后再启动Project，在浏览器中就得这样来访问了：



### 9.3.3.2 常规属性配置

如何在使用Spring容器框架下注入properties文件里的值。如果我们使用了 Spring Boot , 这项工作将会变得更加简单 , 我们只需要在 application.properties中定义属性即可。

```
book.author=罗贯中
book.name=三国演义
book.pinyin=sanguoyanyi
```

然后在变量中通过@Value直接注入就行了，修改TestController的方法，使之返回这些值：

```
7 @RestController
8 public class TestController {
9     @Value(value = "${book.author}")
10    private String author;
11
12    @Value(value = "${book.name}")
13    private String name;
14
15    @Value(value = "${book.pinyin}")
16    private String pinyin;
17
18    @RequestMapping("/test")
19    public String test() {
20        return "test" + " author:" + author + " name:" + name + " pinyin:" + pinyin;
21    }
22 }
```

重新启动，然后在浏览器中访问，结果如下：



非常简单！

### 9.3.3.3 类型安全的配置

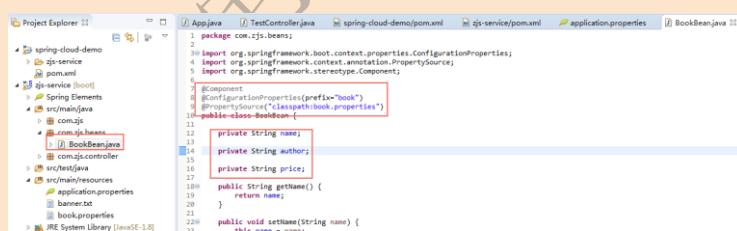
刚刚说的这种方式我们在实际项目中使用的时候工作量略大，因为每个项目要注入的变量的值太多了，这种时候我们可以使用基于类型安全的配置方式，就是将properties属性和一个Bean关联在一起，这样使用起来会更加方便。我么来看看这种方式怎么实现。

在src/main/resources文件夹下创建文件book.properties

```
book.name=红楼梦
book.author=曹雪芹
book.price=28
```

创建 Book Bean，并注入 properties 文件中的值，prefix 是指前缀，

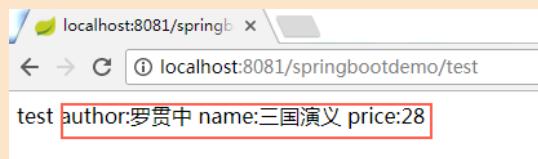
@PropertySource指定要注入文件的位置



在TestController中添加代码注入BookBean



重新启动，然后在浏览器中访问，结果如下：



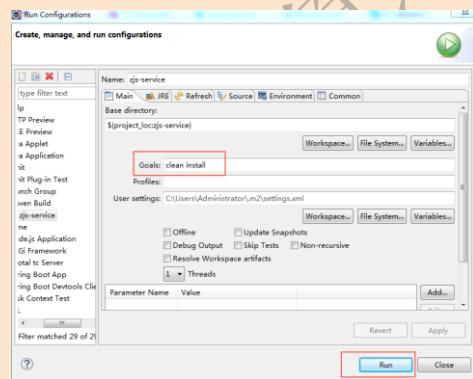
### 9.3.3.4 日志配置

默认情况下Spring Boot使用Logback作为日志框架，也就是我们前面几篇博客中用到的打印日志方式，当然如果有需要我们可以手动配置日志级别以及日志输出位置，相比于我们在Spring容器中写日志输出代码，这里的配置简直就是小儿科了，只需要在application.properties中添加如下代码：

```
logging.file=log.log  
logging.level.org.springframework.web=debug
```

## 9.4 构建可执行 jar 包

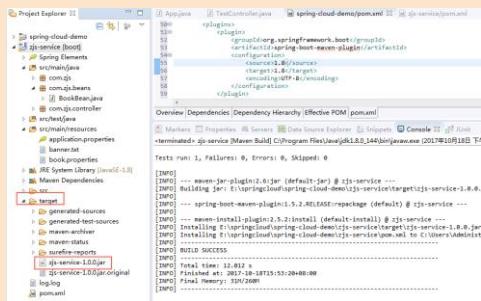
右击项目，选择run as -> Maven Builder，在Goals处输：clean install命令打包，以我的项目工程为例：



开始run，

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
[INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ zjs-service ---  
[INFO] Building jar: E:\springcloud\spring-cloud-demo\zjs-service\target\zjs-service-1.0.0.jar  
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:repackage (default) @ zjs-service ---  
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ zjs-service ---  
[INFO] Installing E:\springcloud\spring-cloud-demo\zjs-service\target\zjs-service-1.0.0.jar to C:\Users\Administrator.m2\repository\com\zjs\zjs-service\zjs-service\1.0.0\zjs-service-1.0.0.jar  
[INFO] BUILD SUCCESS  
[INFO] Total time: 12.812 s  
[INFO] Finished at: 2017-10-18T15:53:20Z  
[INFO] -----
```

执行成功之后，刷新一下项目，jar包出现了



## 9.5 运行

Spring Boot里，很吸引人的一个特性是可以直接把应用打包成为一个jar/war，然后这个jar/war是可以直接启动的，不需要另外配置一个Web Server。单独的JAR包，然后通过java -jar <name>.jar命令运行。以这个项目为例：

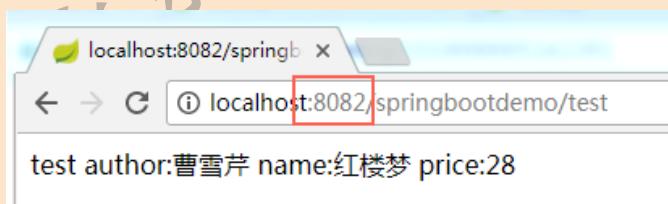
`java -jar zjs-service-1.0.0.jar --server.port=8082`

```

管理员: C:\Windows\system32\cmd.exe -java -jar zjs-service-1.0.0.jar --server.port=8082
anExporter : Registering beans for JMX exposure on startup
2017-10-18 16:06:24.365 DEBUG 8132 --- [main] o.s.u.s.resource.Resource
ceUrlProvider : Looking for resource handler mappings
2017-10-18 16:06:24.366 DEBUG 8132 --- [main] o.s.u.s.resource.Resource
ceUrlProvider : Found resource handler mapping: URL pattern="/**/favicon.ico",
locations=[ServletContext resource [/], class path resource [META-INF/resources/], class path resource [resources/], class path resource [static/], class path resource [public/], class path resource [/], resolvers=[org.springframework.web.servlet.resource.PathResourceResolver@df27fae]]
2017-10-18 16:06:24.367 DEBUG 8132 --- [main] o.s.u.s.resource.Resource
ceUrlProvider : Found resource handler mapping: URL pattern="/*/*", locations=[class path resource [META-INF/resources/webjars/*/*], locations=[ServletContext resource [/], class path resource [META-INF/resources/webjars/*/*], resolvers=[org.springframework.web.servlet.resource.PathResourceResolver@24a35978]
2017-10-18 16:06:24.368 DEBUG 8132 --- [main] o.s.u.s.resource.Resource
ceUrlProvider : Found resource handler mapping: URL pattern="/*/*", locations=[ServletContext resource [/], class path resource [META-INF/resources/*/*], class path resource [resources/*/*], class path resource [static/*/*], class path resource [public/*/*], resolvers=[org.springframework.web.servlet.resource.PathResourceResolver@16f7c8c1]
2017-10-18 16:06:24.469 DEBUG 8132 --- [main] s.h.c.c.c.TomcatEmbedded
dServletContainer : Tomcat started on port(s): 8082 [http]
2017-10-18 16:06:24.471 DEBUG 8132 --- [main] o.s.u.c.s.StandardServer

```

是否成功？在浏览器中访问 “<http://localhost:8082/springbootdemo/test>”



当然也支持其他的方式运行，请自行研究。

# 第十章

## SpringCloud 快速入门



按照官方的话说：Spring Cloud 为开发者提供了在分布式系统（如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性 Token、全局锁、决策竞选、分布式会话和集群状态）操作的开发工具。最关键的是它足够简单，一般的开发人员只需要几天时间就可以学会它的基本用法。

SpringCloud分布式开发五大神兽：

- 服务发现——Netflix Eureka
- 客服端负载均衡——Netflix Ribbon
- 断路器——Netflix Hystrix
- 服务网关——Netflix Zuul
- 分布式配置——Spring Cloud Config

## 10、Spring Cloud 快速入门

第六章已经讲解了Spring Boot的快速入门，Spring Boot的功能还有很多，很强大，想熟练使用它，得有“悬梁刺股”的精神！



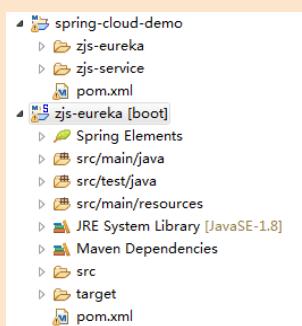
Spring Cloud是在Spring Boot上发展起来的，下面我们以eureka（服务注册与发现）开始它的蜕变之旅吧！

### 10.1 服务的注册与发现（Eureka）

#### 10.1.1 创建服务注册中心

##### 10.1.1.1 创建子项目

在spring-cloud-demo父项目上右击，选择创建“Maven Module”，子项目名称为“zjs-eureka”。

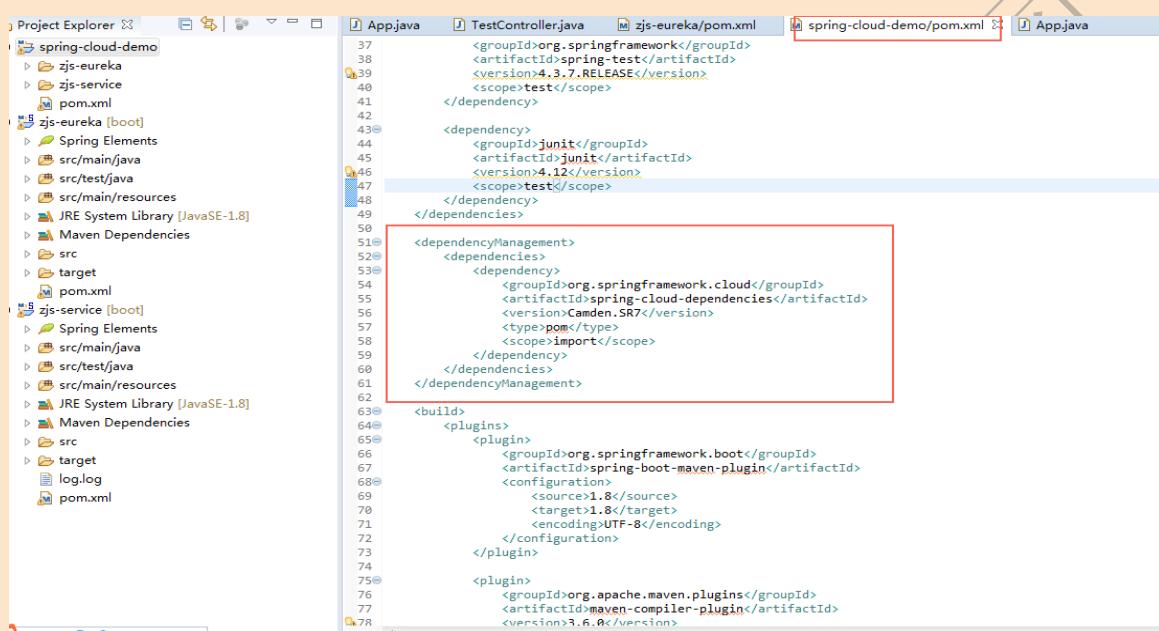


## 10.1.1.2 编辑 POM 文件

怎么将一个SpringBoot改造成SpringCloud呢？其实很简单，现在开始改造吧！

### 10.1.1.2.1 编辑父项目 POM 文件

编辑父项目spring-cloud-demo的POM文件，增加以下内容：



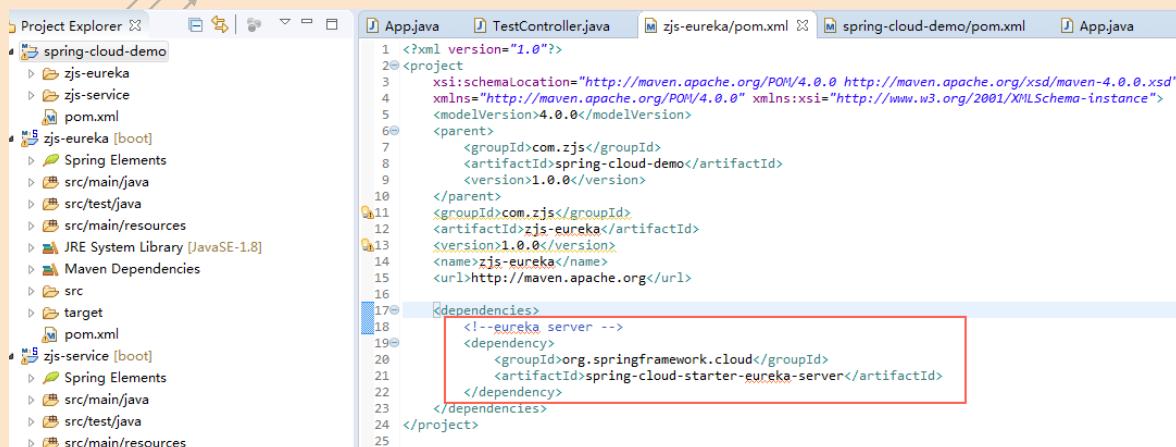
```

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Camden.SR7</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.0</version>
        </plugin>
    </plugins>
</build>

```

### 10.1.1.2.2 编辑子项目 POM 文件

编辑子项目zjs-eureka的POM文件，增加以下内容：



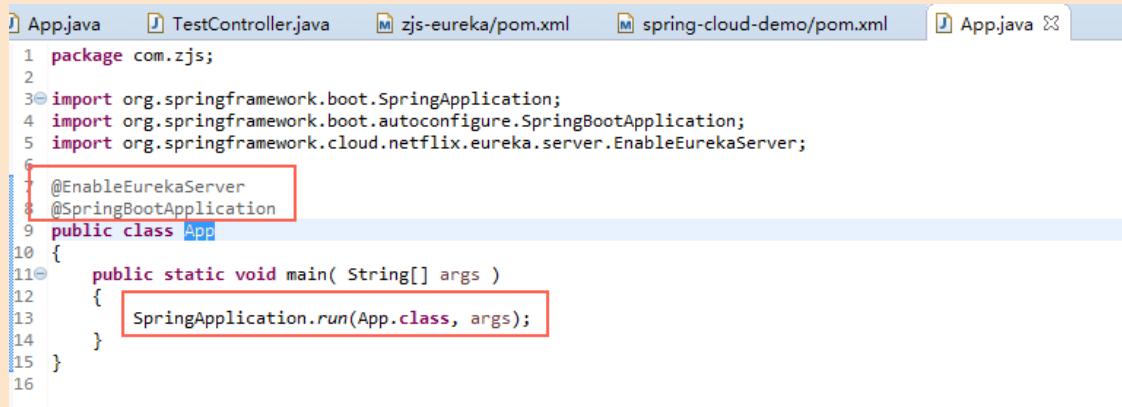
```

<parent>
    <groupId>com.zjs</groupId>
    <artifactId>spring-cloud-demo</artifactId>
    <version>1.0.0</version>
</parent>
<groupId>com.zjs</groupId>
<artifactId>zjs-eureka</artifactId>
<version>1.0.0</version>
<name>zjs-eureka</name>
<url>http://maven.apache.org</url>
<dependencies>
    <!--eureka server -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
    </dependency>
</dependencies>

```

## 10.1.1.3 编辑启动类

编辑子项目zjs-eureka的启动类App.java，内容如下：

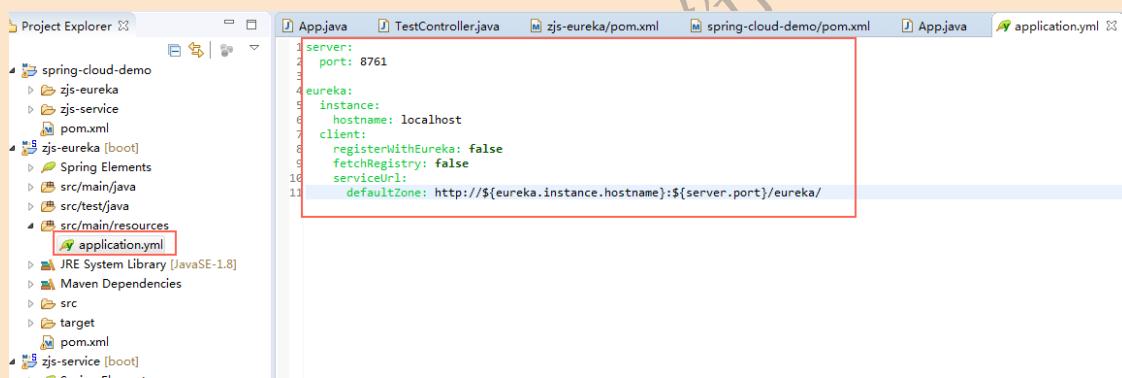


```

1 package com.zjs;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @EnableEurekaServer
8 @SpringBootApplication
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         SpringApplication.run(App.class, args);
14     }
15 }
16

```

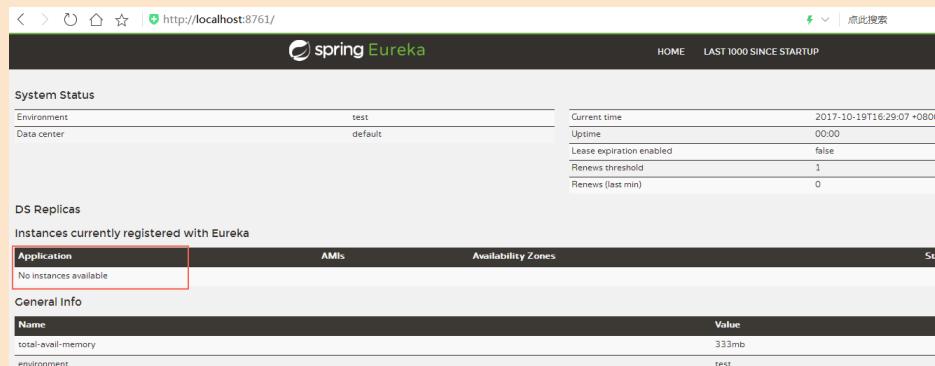
## 10.1.1.4 增加 application.yml 文件



## 10.1.1.5 启动服务注册中心

在子项目zjs-eureka上右键App.java文件，选择run as -> Java Application。

如果没有报错的话，在浏览器中输入：<http://localhost:8761>



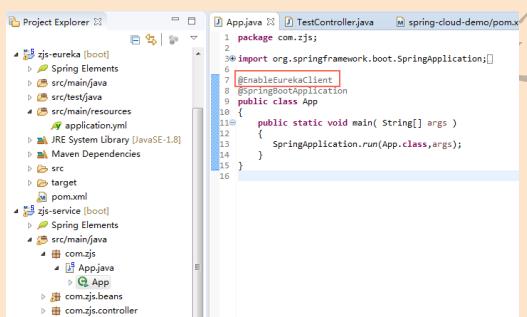
### 10.1.2 创建服务提供者

先前，我们已经创建了一个“zjs-service”服务，运行也是正常的，现在我们动一下手术，让它成为服务的提供者。

#### 10.1.2.1 编辑 POM 文件

```
App.java  TestController.java  spring-cloud-demo/pom.xml  App.java  jzis-service/pom.xml  3
4  	<dependency>
5  	<groupId>http://maven.apache.org/POM/4.0.0</groupId>
6  	<artifactId>maven-assembly-plugin</artifactId>
7  	<version>2.5.4</version>
8  	<configuration>
9  	<groupIds>jzis.com.zjs</groupIds>
10 	<artifactId>spring-cloud-demo</artifactId>
11 	<version>1.0.0</version>
12 	</configuration>
13 	<executions>
14 	<execution>
15 	<phase>package</phase>
16 	<goals><goal>single</goal></goals>
17 	</execution>
18 	</executions>
19 	</dependency>
20 	<dependency>
21 	<groupId>org.springframework.cloud</groupId>
22 	<artifactId>spring-cloud-starter-eureka</artifactId>
23 	</dependency>
24 	<dependency>
25 	<groupId>org.springframework.boot</groupId>
26 	<artifactId>spring-boot-starter-web</artifactId>
27 	</dependency>
28 	</dependencies>
29 </project>
```

### 10.1.2.2 编辑启动类



### 10.1.2.3 编辑配置文件

仅仅@EnableEurekaClient是不够的，还需要在配置文件中注明自己的服务注册中心的地址，application.properties配置文件如下：

```
1 server.context-path=/springbootdemo
2 server.port=8081
3
4 logging.file=log.log
5 logging.level.org.springframework.web=debug
6
7 eureka.client.serviceUrl.defaultZone= http://localhost:8761/eureka/
8 spring.application.name= zjs-service
```

需要指明spring.application.name,这个很重要，这在以后的服务与服务之间相互调用一般都是根据这个name。

启动工程，再次刷新http://localhost:8761

The screenshot shows the Eureka dashboard at http://localhost:8761. At the top, it says 'HOME LAST 1000 SINCE STARTUP'. Below that, the 'System Status' section shows environment 'test' and data center 'default'. The 'DS Replicas' section lists a single instance of 'ZJS-SERVICE' with status 'UP(1) - FC280ZJULYLSRU/zjs-service:8081'. The 'General Info' section provides detailed metrics like total available memory (333mb), environment (test), and current memory usage (82mb, 24%).

你会发现一个服务已经注册在服务中了，服务名为ZJS-SERVICE,端口为8081，这时打开 http://localhost:8081/springbootdemo/test，你会在浏览器上看到：

The browser window shows the URL http://localhost:8081/springbootdemo/test. The page content displays a single record: 'test author:曹雪芹 name:红楼梦 price:28'.

## 10.2 配置服务中心(Config)

### 10.2.1 创建 SVN 配置文件仓库

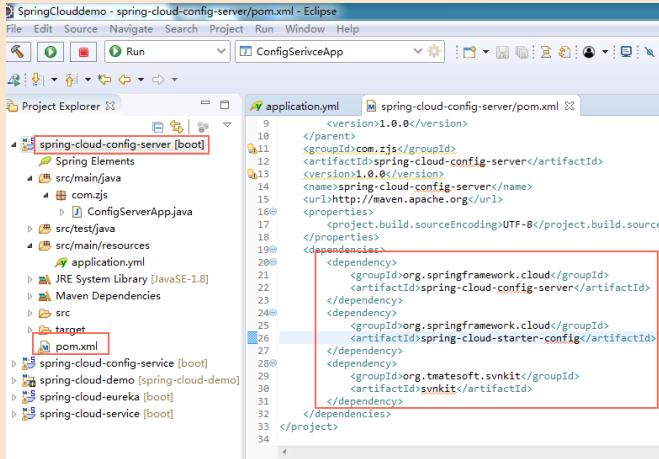
在SVN服务器上创建springcloudconfig，并创建config文件夹，然后写入一个配置文件cloud-config-dev.properties也可以是cloud-config-dev.yml

The screenshot shows a command-line interface with the URL 192.168.33.23/svn/repo/springcloudconfig/config/. Below the URL, it says 'repo - Revision 37: /springcloudconfig/config'. Underneath, there is a list with two items: a dot and a link to 'cloud-config-dev.properties'.

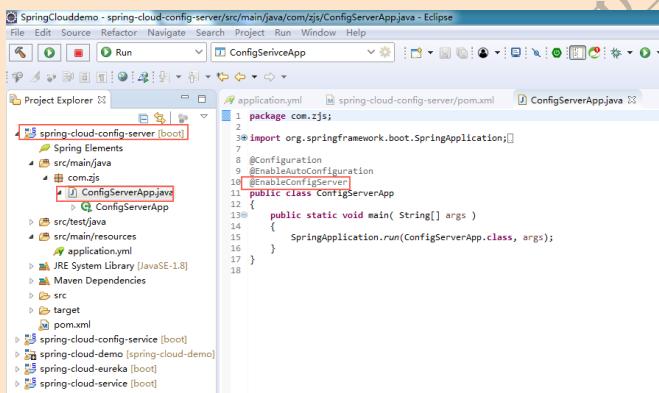
## 10.2.2 创建配置服务端

在spring-cloud-demo下创建spring-cloud-config-server的子项目，并在

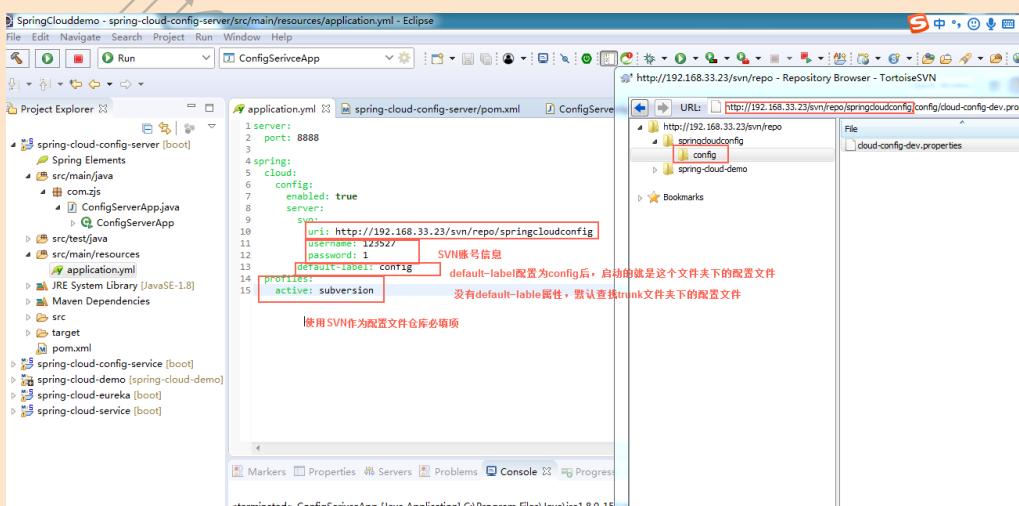
POM文件中添加



创建启动类

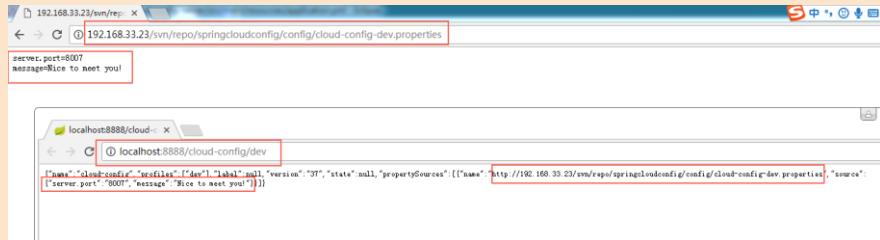


添加配置文件application.yml



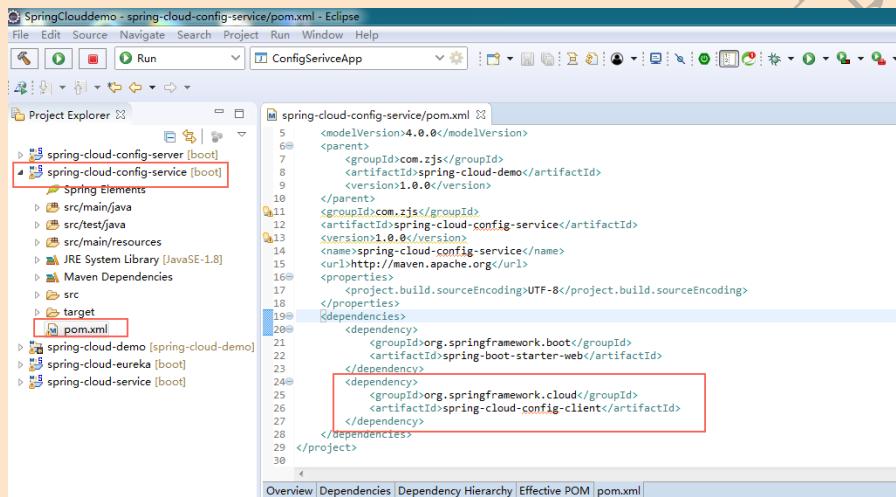
## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

启动程序后，我们访问 <http://localhost:8888/cloud-config/dev>，就可以看到配置信息。

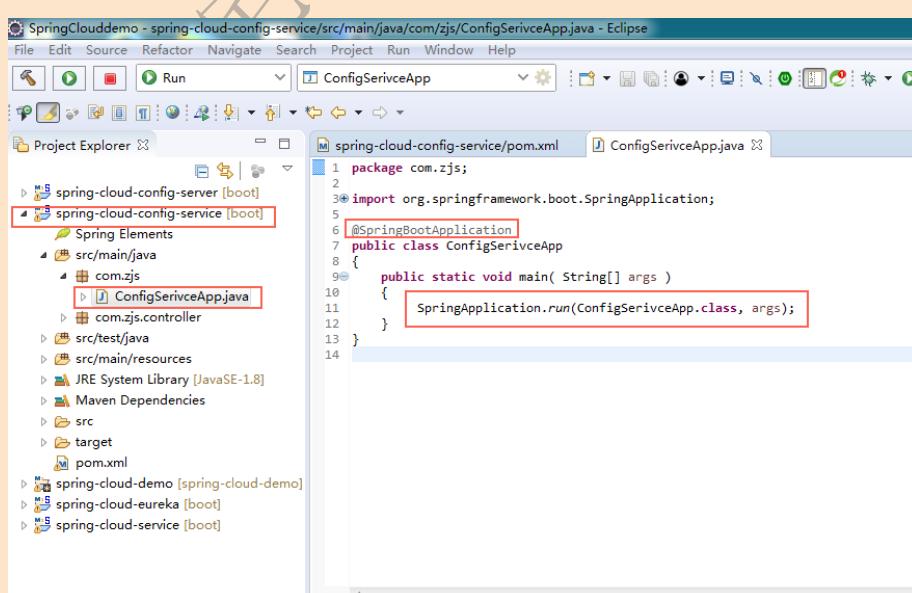


### 10.2.3 创建使用配置服务的客户端

在spring-cloud-demo下创建spring-cloud-config-server的子项目，并在POM文件中添加

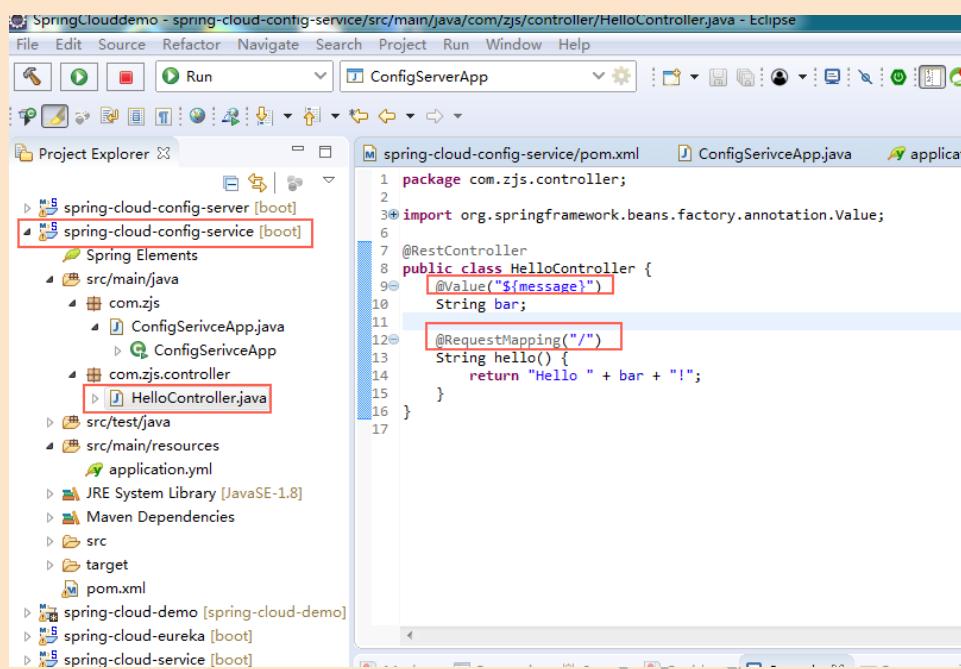


添加启动类



## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

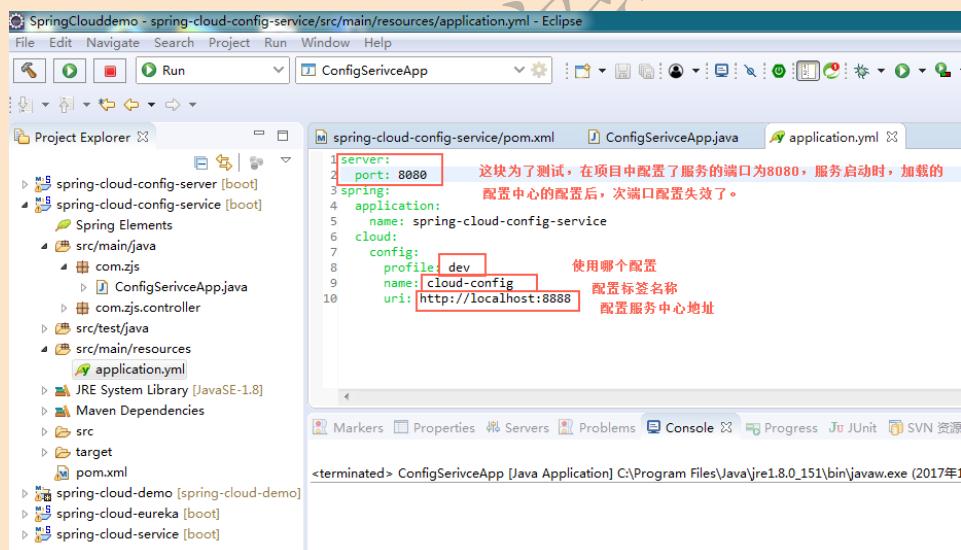
为了直观看到配置中心内容 ,添加HelloController文件 ,里面用到了\${message}变量 ,在此项目中 ,我们并不配置这个值 ,需要从配置中心获取该值。



The screenshot shows the Eclipse IDE interface with the project 'spring-cloud-config-service' selected in the Project Explorer. The code editor displays the 'HelloController.java' file, which contains the following Java code:

```
1 package com.zjs.controller;
2
3 import org.springframework.beans.factory.annotation.Value;
4
5 @RestController
6 public class HelloController {
7     @Value("${message}")
8     String bar;
9
10    @RequestMapping("/")
11    String hello() {
12        return "Hello " + bar + "!";
13    }
14
15 }
16
17
```

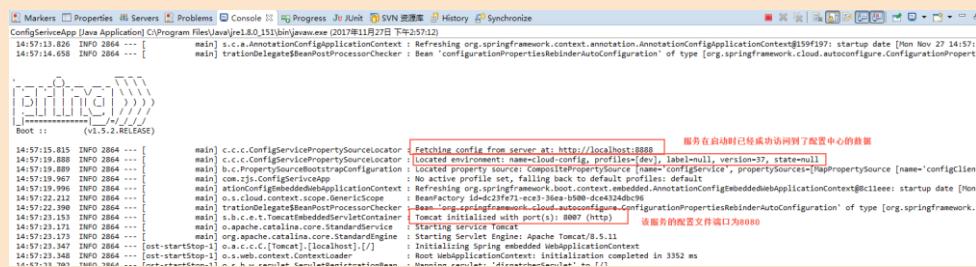
添加配置文件application.yml



The screenshot shows the Eclipse IDE interface with the 'application.yml' file selected in the Project Explorer. The code editor displays the following YAML configuration:

```
server:
  port: 8080 这块为了测试，在项目中配置了服务的端口为8080，服务启动时，加载的
spring: 配置中心的配置后，次端口配置失效了。
application:
  name: spring-cloud-config-service
cloud:
  config:
    profile: dev 使用哪个配置
    name: cloud-config 配置标签名称
    uri: http://localhost:8888 配置服务中心地址
```

接下来 , 启动该服务



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The log output shows the application starting up and connecting to the configuration service:

```
14:57:15.815 INFO 2864 --- [main] c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at: http://localhost:8888 服务在启动时已经成功访问到了配置中心的数据
14:57:15.888 INFO 2864 --- [main] c.c.c.ConfigServicePropertySourceLocator : Located environment name=cloudConfig, profile=dev, label=null, version=37, state=null
14:57:15.901 INFO 2864 --- [main] c.c.c.ConfigServicePropertySourceLocator : Located environment name=cloudConfig, profile=dev, label=null, version=37, state=null
14:57:15.967 INFO 2864 --- [main] com.zjs.ConfigServiceApp : No active profile set, falling back to default profiles: default
14:57:15.996 INFO 2864 --- [main] o.s.b.SpringApplication : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@159f197: startup date [Mon Nov 27 14:57:15 CST 2017]; root of context hierarchy
14:57:22.399 INFO 2864 --- [main] o.s.b.SpringApplication : Bean 'org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration' of type [org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration] initialized with port(8080) : 8080 (http)
14:57:22.415 INFO 2864 --- [main] o.s.b.SpringApplication : Bean 'org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration' of type [org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration] initialized with port(8080) : 8080 (http)
14:57:22.437 INFO 2864 --- [main] o.s.b.SpringApplication : Tomcat initialized with port(s): 8080 (http)
14:57:22.437 INFO 2864 --- [main] o.s.b.SpringApplication : Starting Servlet Engine: Apache Tomcat/8.5.11
14:57:23.347 INFO 2864 --- [ost-startStop-1] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
14:57:23.348 INFO 2864 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 3352 ms
14:57:23.769 INFO 2864 --- [main] o.s.web.context.ContextLoader : Context initialized for context path ''
```

## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

最后服务成功启动了

在浏览器中输入<http://localhost:8007/>

The screenshot shows the Eclipse IDE interface with the following components:

- Project Explorer:** Shows two projects: "spring-cloud-config-server [boot]" and "spring-cloud-config-service [boot]". The "spring-cloud-config-service" project contains a "src/main/java" package with "com.zjs" and "HelloController.java".
- Code Editor:** Displays the content of `HelloController.java`. A red box highlights the line `@Value("${message}")` and the variable `String bar;`. Another red box highlights the return value of the `hello()` method: `"Hello " + bar + "!"`.
- Browser:** A browser window titled "localhost:8007" shows the URL `http://localhost:8007/`. The page content is "Hello Nice to meet you!!".
- Terminal:** Three terminal windows are shown:
  - localhost:8888/cloud-config/dev: Shows the command `curl localhost:8888/cloud-config/dev` and the JSON response:

```
{"name": "cloud-config", "profiles": ["dev"], "label": null, "version": "37", "state": null, "propertySources": [{"name": "http://192.168.33.23/svn/repo/springcloudconfig/config", "server.port": "8007", "message": "Nice to meet you!"}]}
```
  - 192.168.33.23/svn/repo: Shows the command `curl 192.168.33.23/svn/repo` and the properties:

```
server.port=8007
message=Nice to meet you!
```

## SpringCloud有了这个功能后

- 配置信息可以集中管理；
  - 启动服务时，指定配置更加方便；

本章只是介绍了Spring Cloud 的一个方面，其他组件请自行研究。

# 第十一章

SpringCloud 与 Docker 结合

# 11、SpringCloud 与 Docker 结合

下面我们把第六章的两个Spring Cloud项目与docker结合起来，实现程序的编译、打包、发布一体化，前提需要Docker私有仓库。

## 11.1 编辑 POM 文件

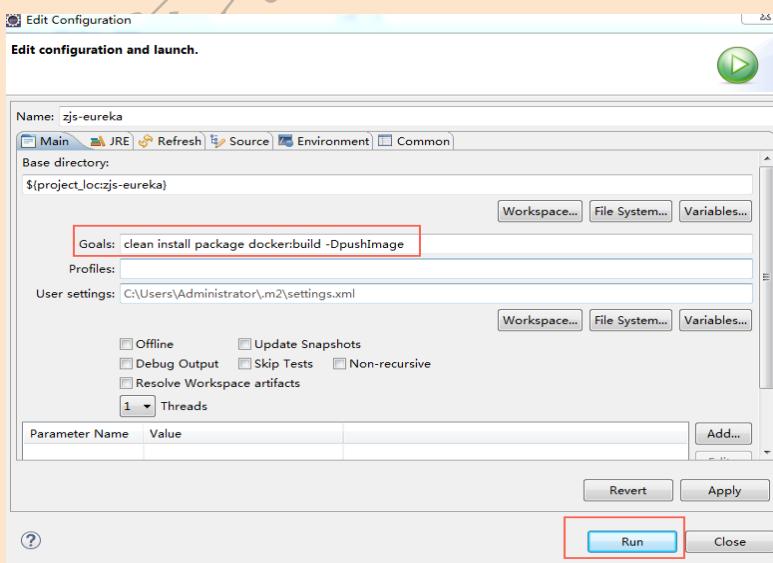
在父子项目中，都增加以下内容：

```

25    <build>
26        <plugins>
27            <plugin>
28                <groupId>com.spotify</groupId>
29                <artifactId>docker-maven-plugin</artifactId>
30                <version>1.0.0</version>
31                <configuration>
32                    <dockerHost>http://192.168.33.21:2375</dockerHost>
33                    <imageName>192.168.33.21:5000/${project.artifactId}:${project.version}</imageName>
34                    <baseImage>java:8-jre-alpine</baseImage>
35                    <maintainer>张永刚</maintainer>
36
37                    <entryPoint>"java", "-jar", "${project.name}-${project.version}.jar"</entryPoint>
38                    <resources>
39                        <resource>
40                            <targetPath>/</targetPath>
41                            <directory> ${project.build.directory} </directory>
42                            <include>${project.name}-${project.version}.jar</include>
43                        </resource>
44                    </resources>
45                </configuration>
46            </plugin>
47
48        </plugins>
49    </build>
50 </project>
```

## 11.2 编译\打包\上传

在父子项目上各自执行,右键项目,选择 run as -> Maven Builder, 在 Goals 处填写“clean install package docker:build -DpushImage”，选择 Run



经过一段时间的等待，终于上传成功了！

```
<terminated> zjs-eureka [Maven Build] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年10月19日 下午5:11:28)
2dd874c2ab: Pushing [=====] 98.71 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 99.23 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 99.79 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 100.3 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 100.9 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 101.4 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 102 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 102.5 MB/103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 103 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 103.6 MB
[288] 3AB[2K
2dd874c2ab: Pushing [=====] 103.8 MB
[288] 3AB[2K
2dd874c2ab: Pushed
[281.0.0: digest: sha256:480c098aa10a68f67f4778b127b166d26d3f2101877fc06e490e3c5058584fc size: 1159
null: null
[INFO]
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 36.134 s
[INFO] Finished at: 2017-10-19T17:12:06+08:00
[INFO] Final Memory: 53M/388M
[INFO]
```

查看以下 Docker 私有仓库的镜像

看到这些信息，恭喜您！

接下来，我们以容器的方式启动服务注册中心以及服务提供者

docker run -d -p 8761:8761 192.168.33.21:5000/zjs-eureka:1.0.0 (先运行)

docker run -d -p 8081:8081 192.168.33.21:5000/zjs-service:1.0.0

docker ps -a

```
root@k8s1 ~]# docker run -d -p 8761:8761 192.168.33.21:5000/zjs-eureka:1.0.0
f5cbcede7cbe9718a8a8c906c1ab524c599cc861a7ad25656037e5550c2e1c
root@k8s1 ~]# docker run -d -p 8081:8081 192.168.33.21:5000/zjs-service:1.0.0
e8449557414a5641ef510ee152b5470a34052a809bd7de74f860b4ea6a
root@k8s1 ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS                         NAMES
e8449557414a      192.168.33.21:5000/zjs-service:1.0.0   "java -jar /zjs-servi"   14 seconds ago    Up 9 seconds          0.0.0.0:8081->8081/tcp   determined_kowalev
c1                 f5cbcede7cbe      "java -jar /zjs-eurek"   24 seconds ago    Up 23 seconds         0.0.0.0:8761->8761/tcp   boring_meitner
1104047100b1       docker.io/registry     "/entrypoint.sh /etc/"  11 minutes ago     Up 11 minutes        0.0.0.0:5000->5000/tcp   registry
root@k8s1 ~]#
```

在浏览器中访问 <http://192.168.33.21:8761>

Environment	test	Current time
Data center	default	Uptime
		Lease expiration enabled
		Renews threshold
		Renews (last min)

**DS Replicas**

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones
-------------	------	--------------------

好了，Spring Cloud 和 Docker 结合就是这样子！

参考：<https://my.oschina.net/u/162754/blog?sort=time&p=5>

# 第十二章

## Jmeter 压力测试



Apache JMeter 是 Apache 组织开发的基于 Java 的压力测试工具。用于对软件做压力测试，它最初被设计用于 Web 应用测试，但后来扩展到其他测试领域。它可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器，等等。 JMeter 可以用于对服务器、网络或对象模拟巨大的负载，来自不同压力类别下测试它们的强度和分析整体性能。另外，JMeter 能够对应用程序做功能/回归测试，通过创建带有断言的脚本来验证你的程序返回了你期望的结果。为了最大限度的灵活性，JMeter 允许使用正则表达式创建断言。

# 12、Jemeter 压力测试

## 12.1 安装 Jemeter

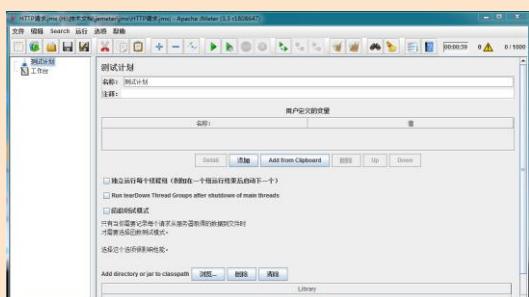
下载地址: <http://jmeter.apache.org>

## 12.2 解压文件

下载解压到某个盘, 我放在了 D 盘

## 12.3 启动 jemeter

在安装目录的 bin 文件夹里, 点击 “jmeter.bat”, 启动成功后的界面



如果是英文版请点击 Options>Choose Language>Chinese(Simplified), 切换成简体中文。

## 12.4 体验 jemeter

### 12.4.1 准备一个待测试的服务

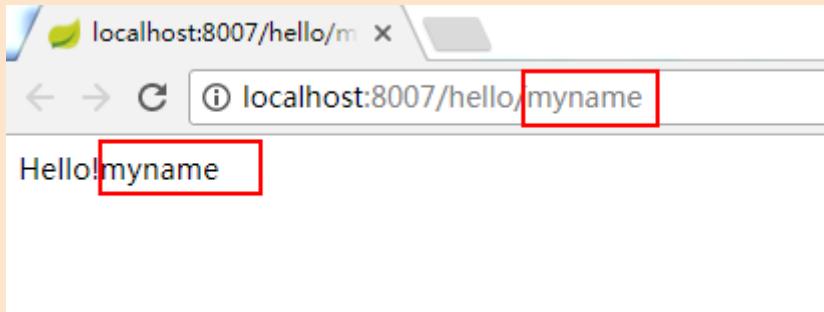
我们用 eclipse 开发一个 spring-cloud-service 的服务

 A screenshot of the Eclipse IDE. On the left, the 'Project Explorer' view shows a project named 'spring-cloud-demo' with various Java files and resources. On the right, the 'HelloController.java' code editor is open. The code defines a REST controller with a single endpoint that prints the current time and user name to the console. Red boxes highlight the annotations and the println statement.

```

1 package com.zjs.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 public class HelloController {
9
10    @GetMapping("/hello/{user}")
11    public String sayHello(@PathVariable("user") String user) {
12        System.out.println("现在时间: " + System.currentTimeMillis() + " " + user);
13        return "Hello!" + user;
14    }
15
16 }
    
```

启动后，访问浏览器 <http://localhost:8007/hello/myname>



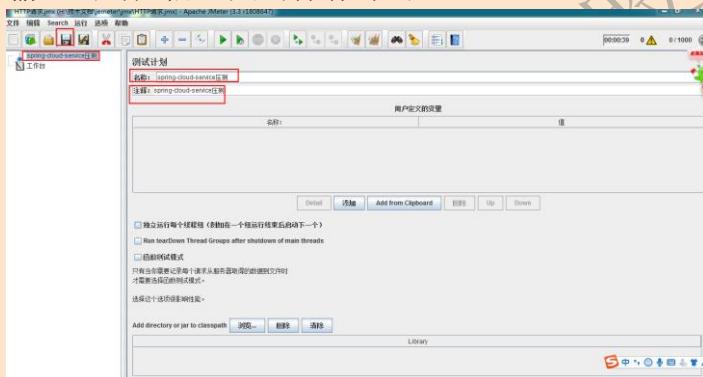
可以看到，这个服务有一个参数 user 是需要的。

## 12.4.2 创建 jemeter 压测内容

接下来，我们模拟 1000 用户访问该服务

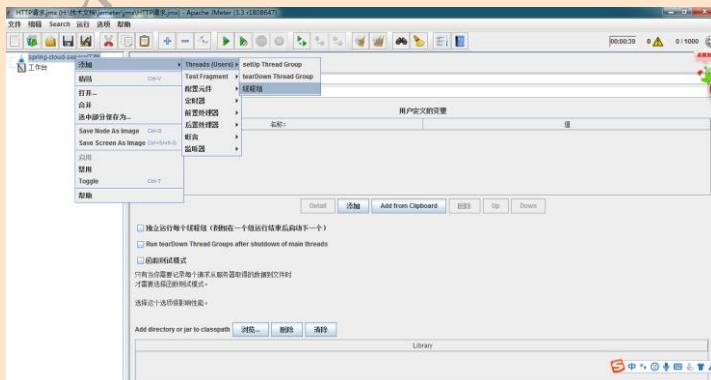
### 12.4.2.1 创建测试计划

输入“名称”信息点击保存即可。

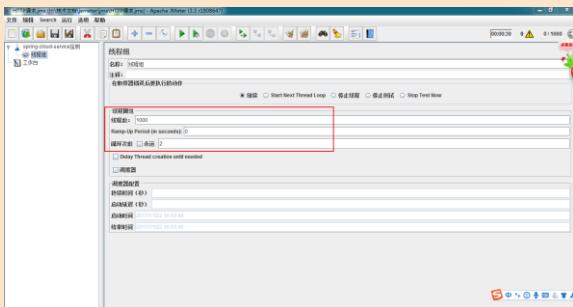


### 12.4.2.2 创建线程组

右键点击“spring-cloud-service 压测”，添加>Threads(Users)>线程组



## 12.4.2.3 设置线程数

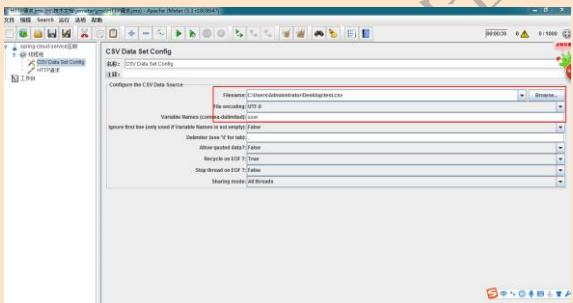


- Number of Threads:一个用户占一个线程，1000个线程就是模拟1000个用户
- Ramp-Up Period(in seconds):设置线程需要多长时间全部启动。如果线程数为1000，准备时长为10，那么需要1秒钟启动100个线程。如果设置为0，就是并发；
- Loop Count: 每个线程发送请求的次数。如果线程数为1000，循环次数为2，那么每个线程发送2次请求。总请求数为 $1000 \times 2 = 2000$ 。如果勾选了“永远”，那么所有线程会一直发送请求，直到选择停止运行脚本。

## 12.4.2.4 添加 CSV Data Set Config

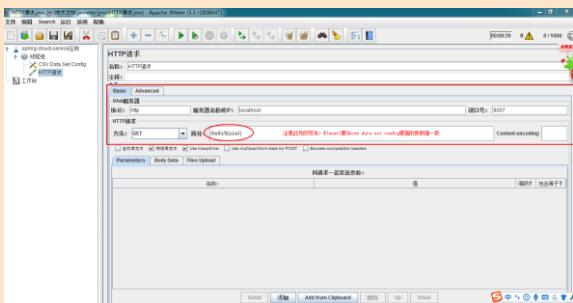
我们的服务需要传递一个参数值，那么我们可以先创建一个 csv 文件，里面只有一列信息。

右键点击线程组>添加>配置元件>CSV Data Set Config，并填写相关信息



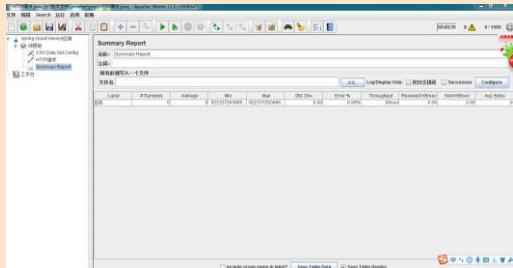
## 12.4.2.4 创建 HTTP 请求

右键点击线程组>添加>Sampler>HTTP 请求，并填写相关参数



## 12.4.2.5 添加 Summary Report

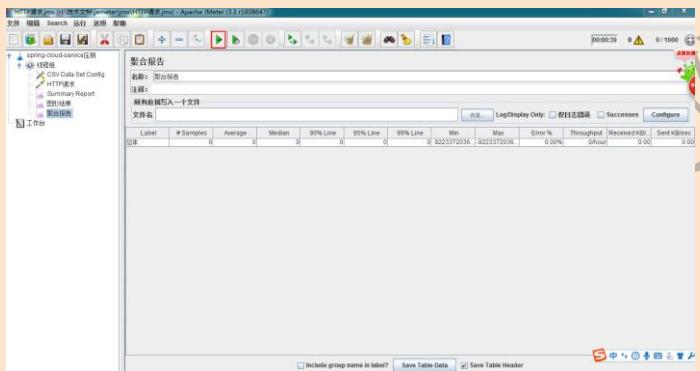
右键点击线程组>添加>监听器> Summary Report，用来查看测试结果



还可以添加很多监听器的内容，如“图形结果”、“聚合报告”等等。

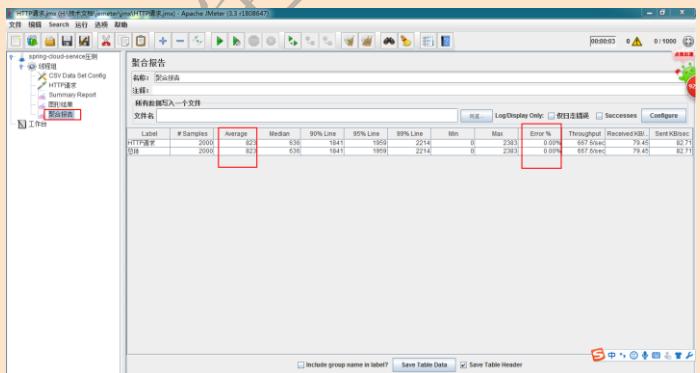
## 12.4.3 开始运行

到目前为止，脚本就全写好了，点击开始进行压力测试。



## 12.4.4 测试结果

压力测试结果很快就出现了。



Jmeter 用来做轻量级的压力测试，非常合适，只需要十几分钟，就能把压力测试需要的脚本写好。测试脚本被保存后，可以重复使用，也可以被导入到其他的电脑上运行。



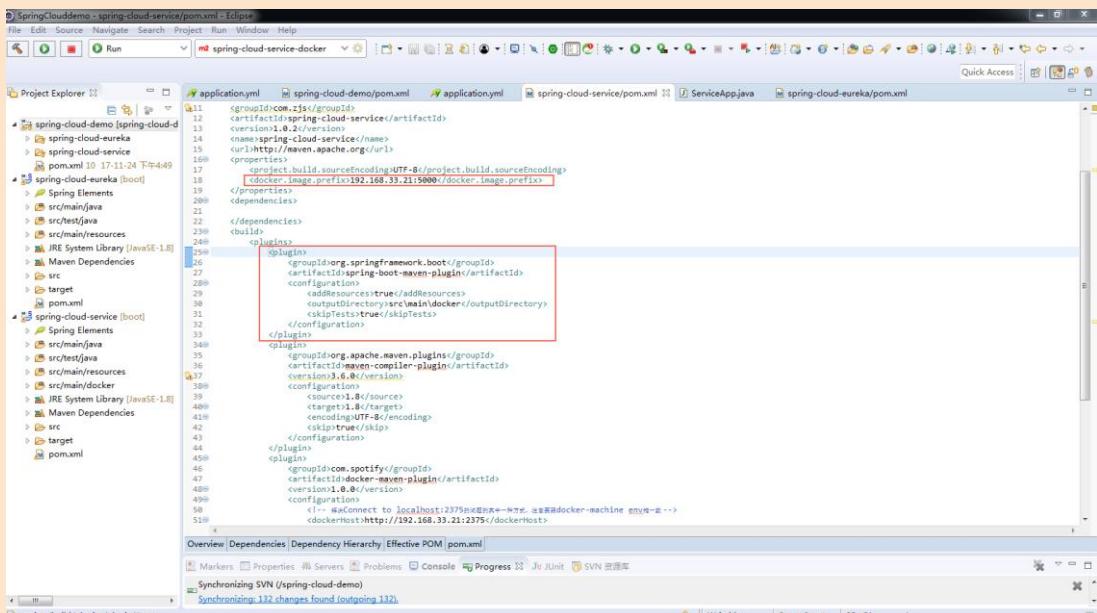
# 第十三章

Jenkins 自动化构建

# 13、jenkins 自动化构建

## 13.1 构建 maven 项目 docker 镜像

### 13.1.1 创建 maven 工程

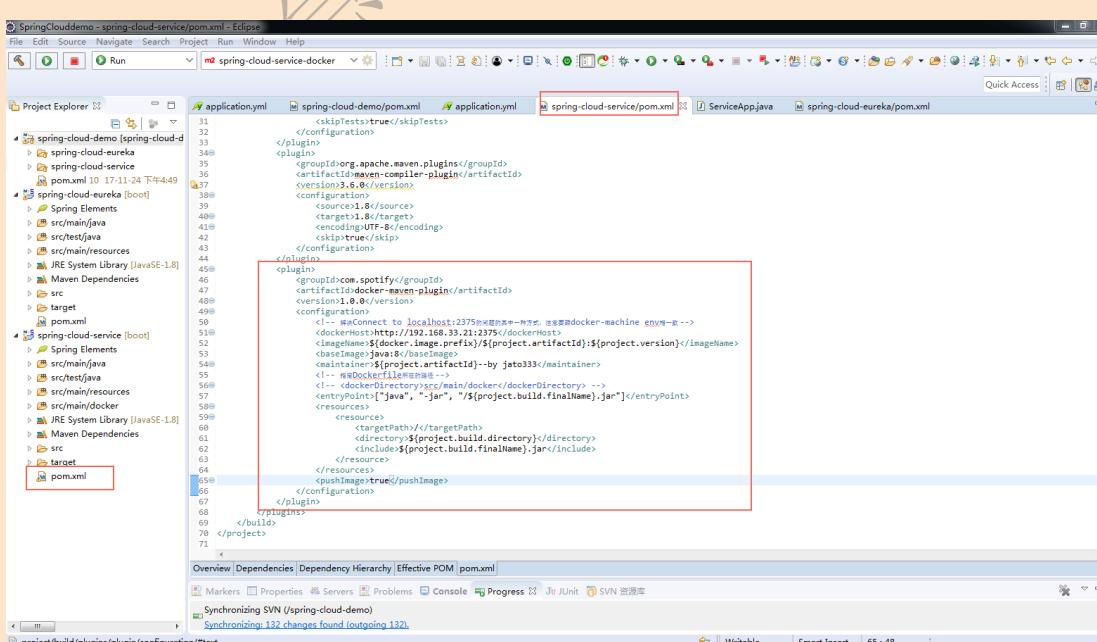


```

<build>
    <sourceEncoding>UTF-8</sourceEncoding>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <addResources>true</addResources>
                <outputDirectory>src/main/docker</outputDirectory>
                <skipTests>true</skipTests>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
                <skip>false</skip>
            </configuration>
        </plugin>
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.0.0</version>
            <configuration>
                <image>${project.build.imagePrefix}192.168.33.21:5000</image>
                <dockerHost>http://192.168.33.21:2375</dockerHost>
                <skipBuild>true</skipBuild>
            </configuration>
        </plugin>
    </plugins>

```

### 13.1.2 添加 docker-maven 插件



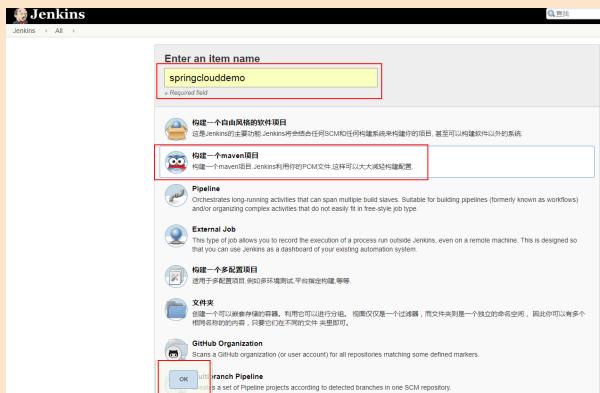
```

<build>
    <sourceEncoding>UTF-8</sourceEncoding>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
                <skip>false</skip>
            </configuration>
        </plugin>
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.0.0</version>
            <configuration>
                <image>${project.build.imagePrefix}192.168.33.21:5000</image>
                <dockerHost>http://192.168.33.21:2375</dockerHost>
                <imageName>${project.artifactId}:${project.version}</imageName>
                <baseImage>java:8-jdk</baseImage>
                <containerName>${project.artifactId}</containerName>
                <entryPoint>java -jar ${project.build.finalName}.jar</entryPoint>
                <resources>
                    <resource>
                        <targetPath>/target</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.jar</include>
                    </resource>
                </resources>
                <pushImage>true</pushImage>
            </configuration>
        </plugin>
    </plugins>

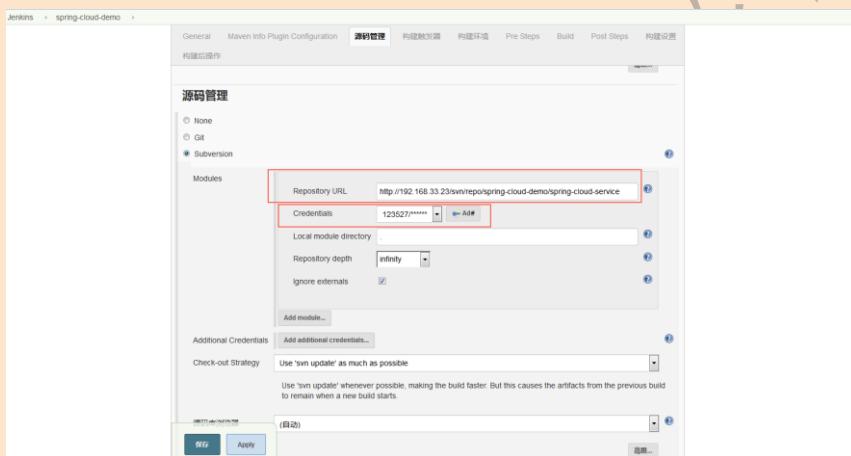
```

## 13.1.3 在 jenkins 中构建一个 maven 项目

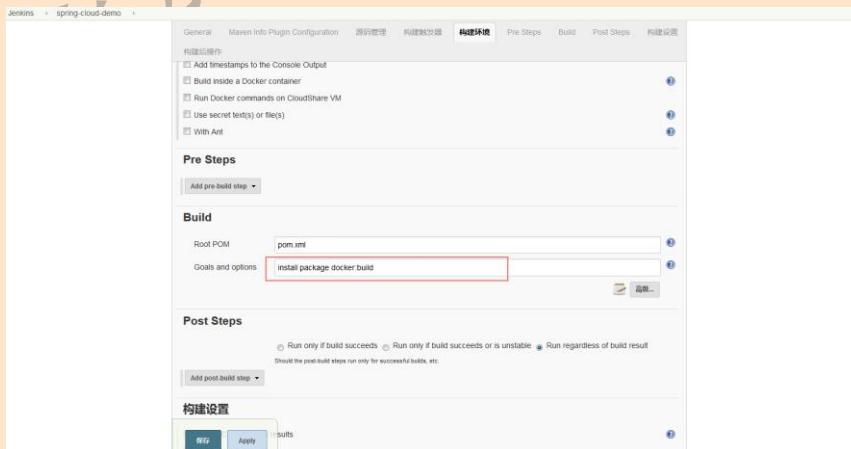
在 Jenkins 首页选择“新建”，输入名字，选择“构建一个 maven 项目”：



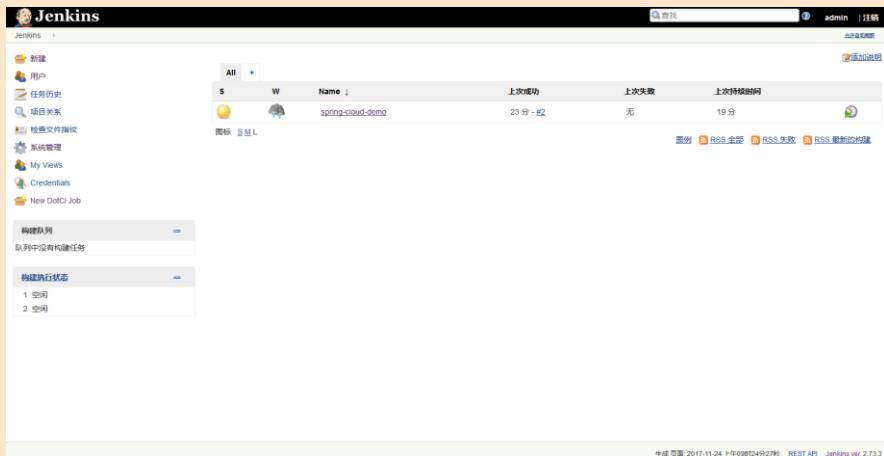
## 13.1.4 配置构建项目



## 13.1.5 配置构建项目要执行的命令

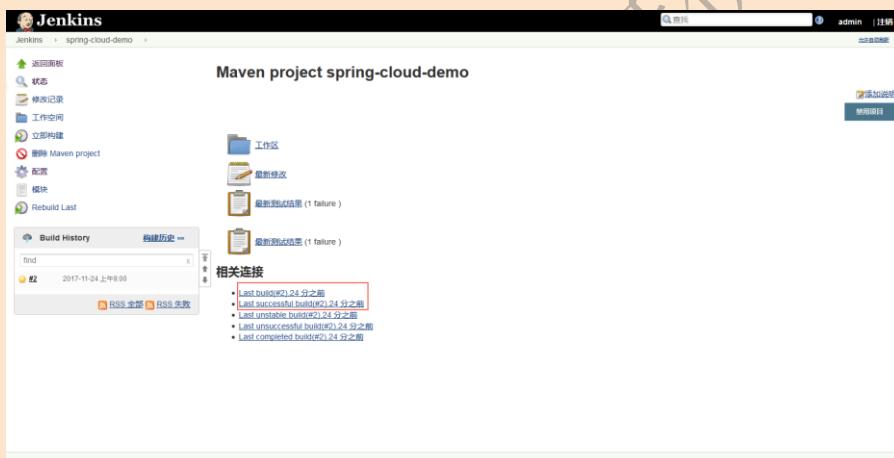


### 13.1.6 完成构建项目的配置

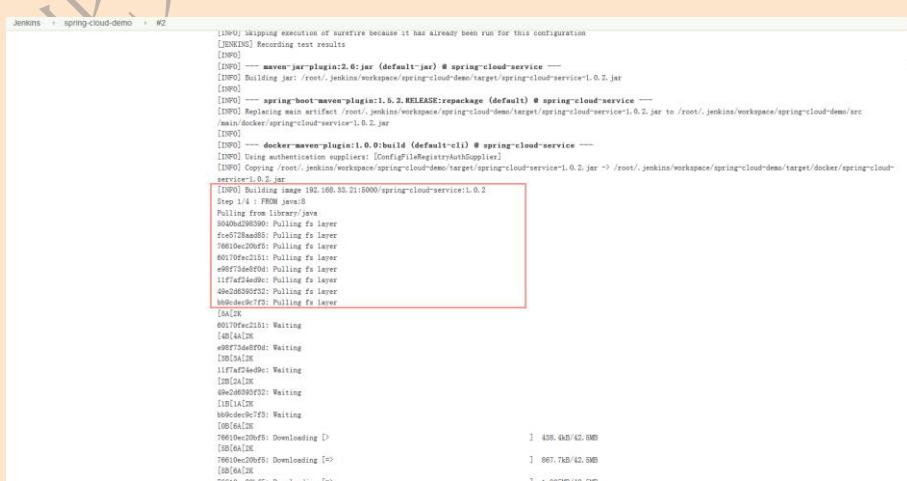


### 13.1.7 立即构建

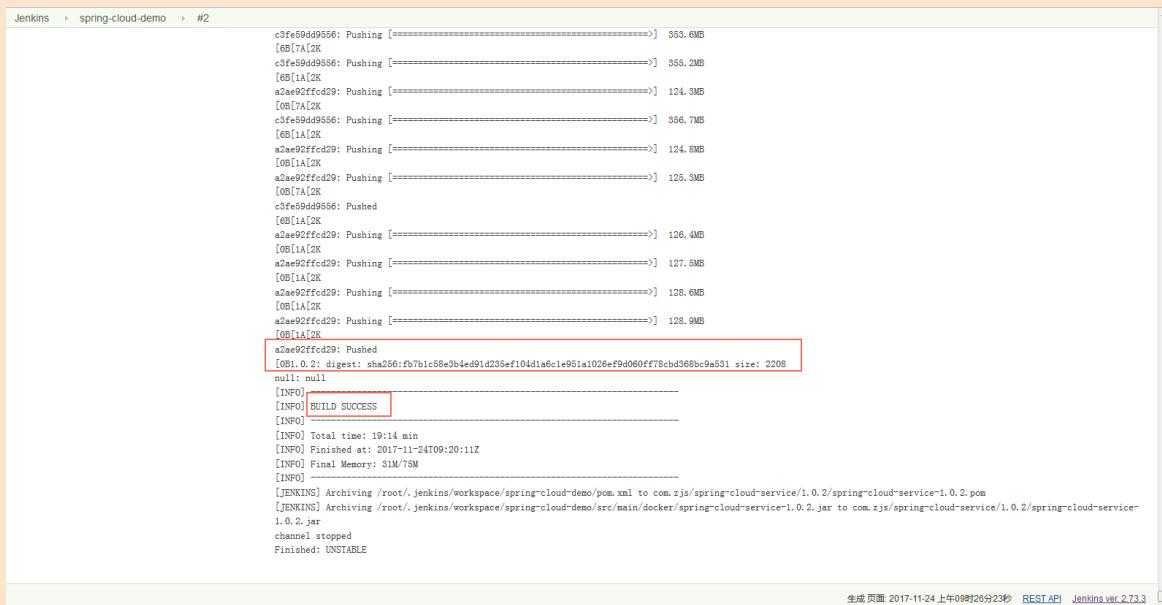
点击“立即构建”，启动构建过程



构建的过程会下载相关的镜像文件



因为在项目的 POM 文件中，有“<pushImage>TRUE</pushImage>”的设置，所以构建成功后，在镜像仓库上可以看见构建成功，并上传的镜像文件



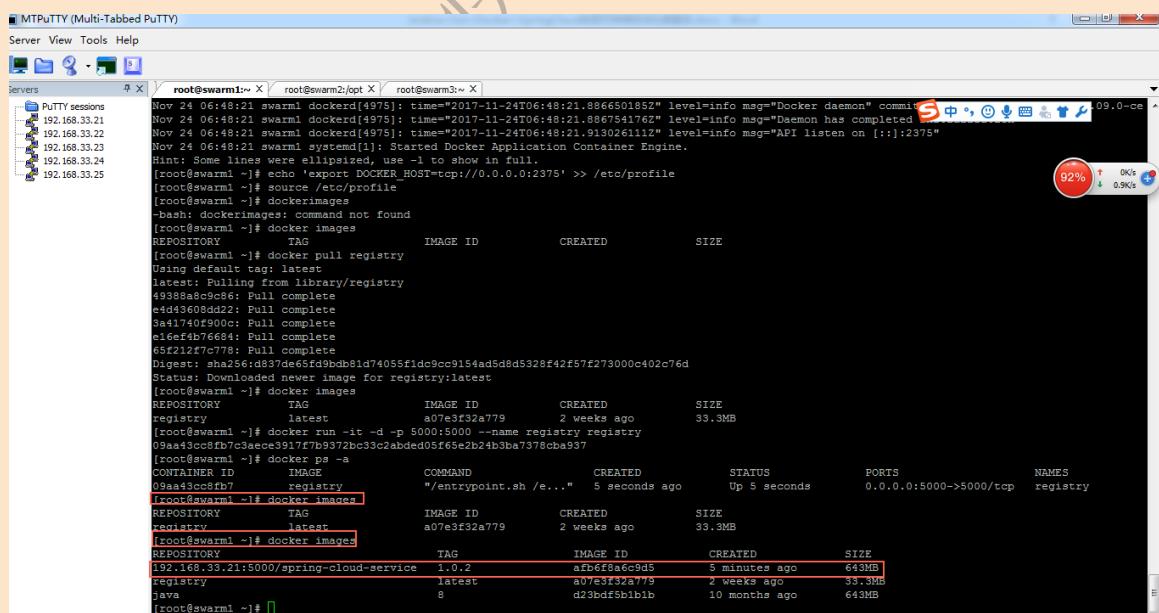
```

Jenkins > spring-cloud-demo > #2
[...]
c3fe59d49556: Pushing [=====] 365.6MB
[0B1A12K]
c3fe59d49556: Pushing [=====] 355.2MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 124.3MB
[0B1A12K]
c3fe59d49556: Pushing [=====] 356.7MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 124.8MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 125.3MB
[0B1A12K]
c3fe59d49556: Pushed
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 126.4MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 127.5MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 128.6MB
[0B1A12K]
a2ae92ffcd29: Pushing [=====] 128.9MB
[0B1A12K]
a2ae92ffcd29: Pushed
[0B1A12K]
null: null
[INFO] [INFO] BUILD SUCCESS [INFO]
[INFO] [INFO] Total time: 19:11 min
[INFO] Finished at: 2017-11-24T09:20:11Z
[INFO] Final Memory: 31M/78M
[INFO] -----
[JENKINS] Archiving /root/.jenkins/workspace/spring-cloud-demo/pom.xml to com.zjs/spring-cloud-service/1.0.2/spring-cloud-service~1.0.2.pom
[JENKINS] Archiving /root/.jenkins/workspace/spring-cloud-demo/src/main/docker/spring-cloud-service~1.0.2.jar to com.zjs/spring-cloud-service~1.0.2.jar
channel stopped
Finished: UNSTABLE

```

生成页面 2017-11-24 上午09时26分23秒 REST API Jenkins ver.2.73.3

## 13.1.8 验证构建成功的镜像文件



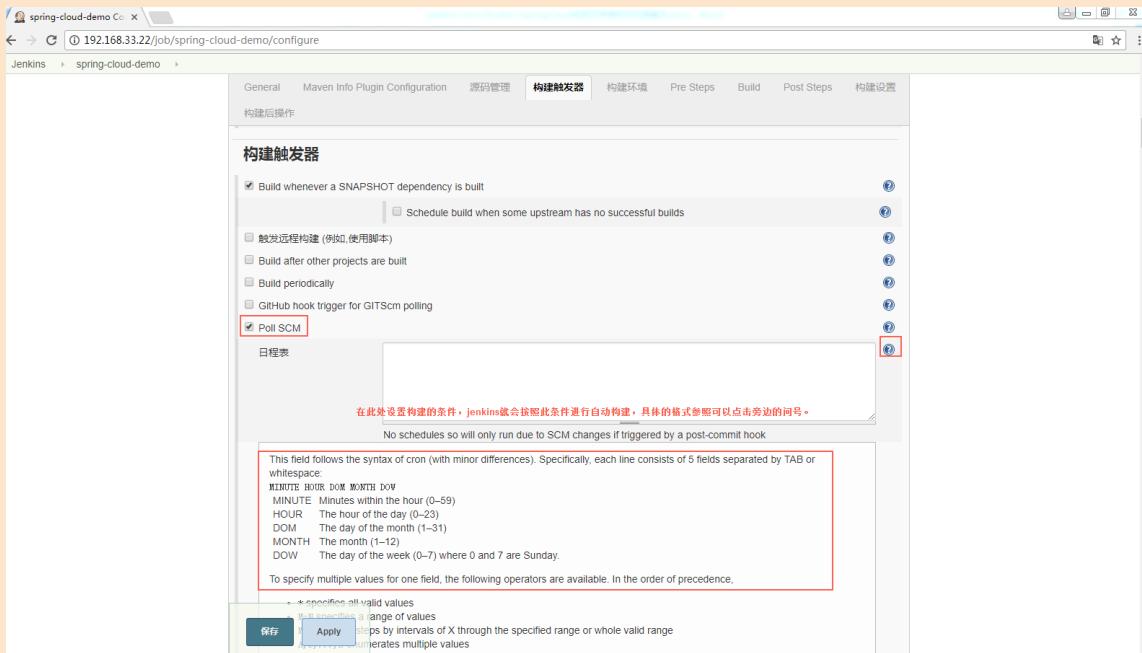
```

root@swarm1:~# root@swarm2:/opt # root@swarm3:~#
[...]
Nov 24 06:49:21 swarm1 dockerd[4975]: time="2017-11-24T06:49:21.886650185Z" level=info msg="Docker daemon" commit=09...
Nov 24 06:49:21 swarm1 dockerd[4975]: time="2017-11-24T06:49:21.886754176Z" level=info msg="Daemon has completed ...
Nov 24 06:49:21 swarm1 dockerd[4975]: time="2017-11-24T06:49:21.913026111Z" level=info msg="API listen on [:]:2375"
Nov 24 06:49:21 swarm1 dockermd[1]: Started Docker Application Container Engine.
Hint: Some lines were ellipsized, use -l to show in full.
[root@swarm1 ~]# echo 'export DOCKER_HOST=tcp://0.0.0.0:2375' >> /etc/profile
[root@swarm1 ~]# source /etc/profile
[root@swarm1 ~]# dockerimages
-bash: dockerimages: command not found
[root@swarm1 ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
[root@swarm1 ~]# docker pull registry
Using default tag: latest
latest: Pulling from library/registry
49388a8c9c86: Pull complete
e4d43609d22: Pull complete
3a41740f900: Pull complete
e16ef4b76684: Pull complete
657212f7c778: Pull complete
Digest: sha256:d837deef5fd9bdb81d74055f1dc9cc9154ad58d5328f42f57f273000c402c76d
Status: Downloaded newer image for registry:latest
[root@swarm1 ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
registry latest a07e3f32a779 2 weeks ago 33.3MB
[root@swarm1 ~]# docker run -it -d -p 5000:5000 --name registry registry
09a414cd2fb7c3aeece3590fb372bc53c2abded05f6e5e2b24b3da7376cb957
[root@swarm1 ~]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
09a414cd2fb7 registry "/entrypoint.sh /..." 5 seconds ago Up 5 seconds 0.0.0.0:5000->5000/tcp registry
[root@swarm1 ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
192.168.33.21:5000/spring-cloud-service 1.0.2 afbf6f8a6c9d5 5 minutes ago 643MB
registry latest a07e3f32a779 2 weeks ago 33.3MB
java 8 d23bdf5b1b1b 10 months ago 663MB
[root@swarm1 ~]#

```

## 13.2 设置自动化构建

上述步骤已经成功构建了 maven 项目，并打包成 docker 镜像文件，上传到私有镜像仓库中。只需小小的设置，就可实现**自动化**构建。



### 构建时间 Poll Scm 的设置

- 每 15 分钟构建一次: H/15 \* \* \* \* 或 \*/5 \* \* \* \*
- 每天 8 点构建一次: 0 8 \* \* \*
- 每天 8 点~17 点, 两小时构建一次: 0 8-17/2 \* \* \*
- 周一到周五, 8 点~17 点, 两小时构建一次: 0 8-17/2 \* \* 1-5
- 每月 1 号、15 号各构建一次, 除 12 月: H H 1,15 1-11 \*

此项设置后，我们只需关注编码即可，剩下的事情完全交由 jenkins 按时间来触发构建过程。

## 13.3jenkins 状态通知

### 13.3.1 与钉钉关联

#### 13.3.1.1 安装钉钉相关插件

在主界面，点击“系统设置”->“插件管理” ->“可选插件” 中搜索“dingding”，选中出现的插件安装即可。



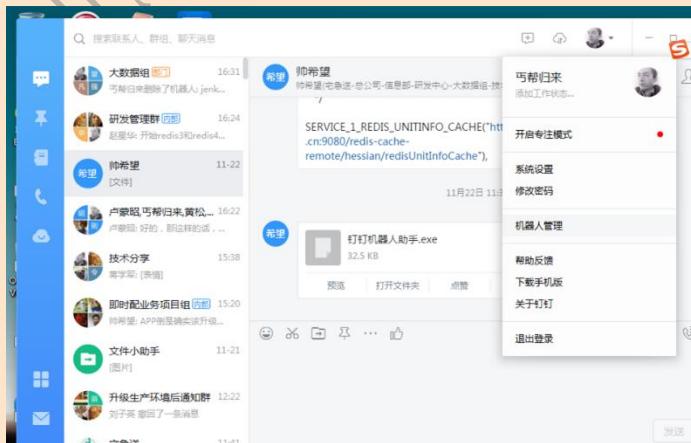
完成之后，返回到主界面



#### 13.3.1.2 获取钉钉 token

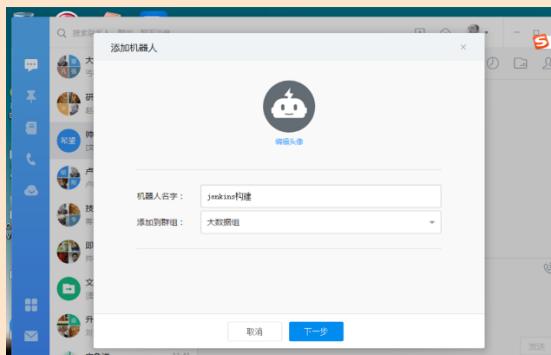
##### 13.3.1.2.1 创建钉钉机器人

在一个在需要通知的钉钉群中，点击头像旁边的三角，选择“机器人管理”



## Jenkins+Svn+Docker+SpringCloud 实现可持续自动化微服务

添加机器人



这个界面出现的值非常有用，复制下来准备下一步使用

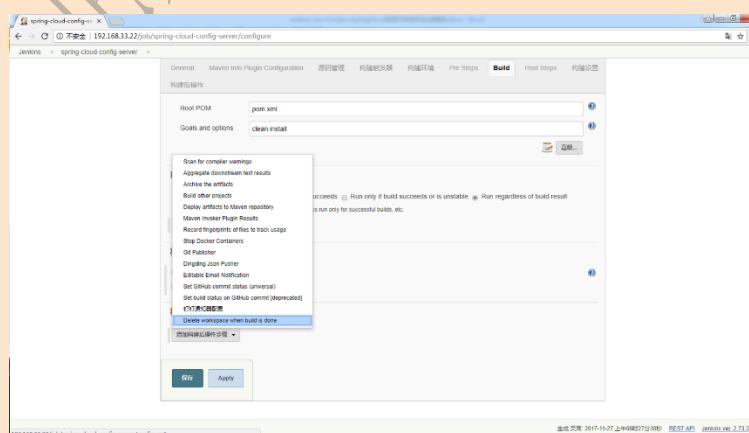


点击完成之后，会在群内出现一条消息



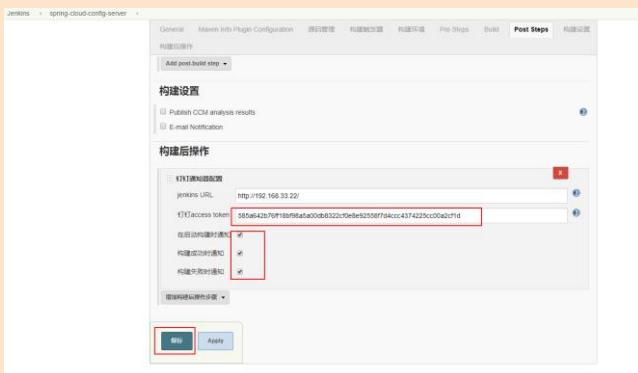
### 13.3.1.2 重新构建配置

在“构建后操作”板块增加“钉钉通知配置器”



### 13.3.1.3 钉钉通知配置

此处只需填写钉钉机器人的 token 值即可



再次构建后，会将构建状态及时通知到钉钉群中



有了这个功能，每次自动化构建的状态会以即时消息的方式及时通知群里的每一个成员，非常实用！

### 13.4 可能遇到的问题

No plugin found for prefix 'docker'，可能是网络问题导致下载插件失败！

构建的过程中可能会出现上述问题，检查 jenkins 服务器的 maven 配置

文件 `settings.xml`，添加设置代理，如：

```
<mirror>
  <id>alimaven</id>
  <mirrorOf>central</mirrorOf>
  <name>aliyun maven</name>
  <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

参考：

<http://blog.csdn.net/shycx/article/details/7726600>

# 第十四章

Portainer 管理微服务容器

## 14、Portainer 管理微服务容器

未完待续.....

北京宅急送快运股份有限公司

北京宅急送快运股份有限公司