

Jenkins持续集成&部署

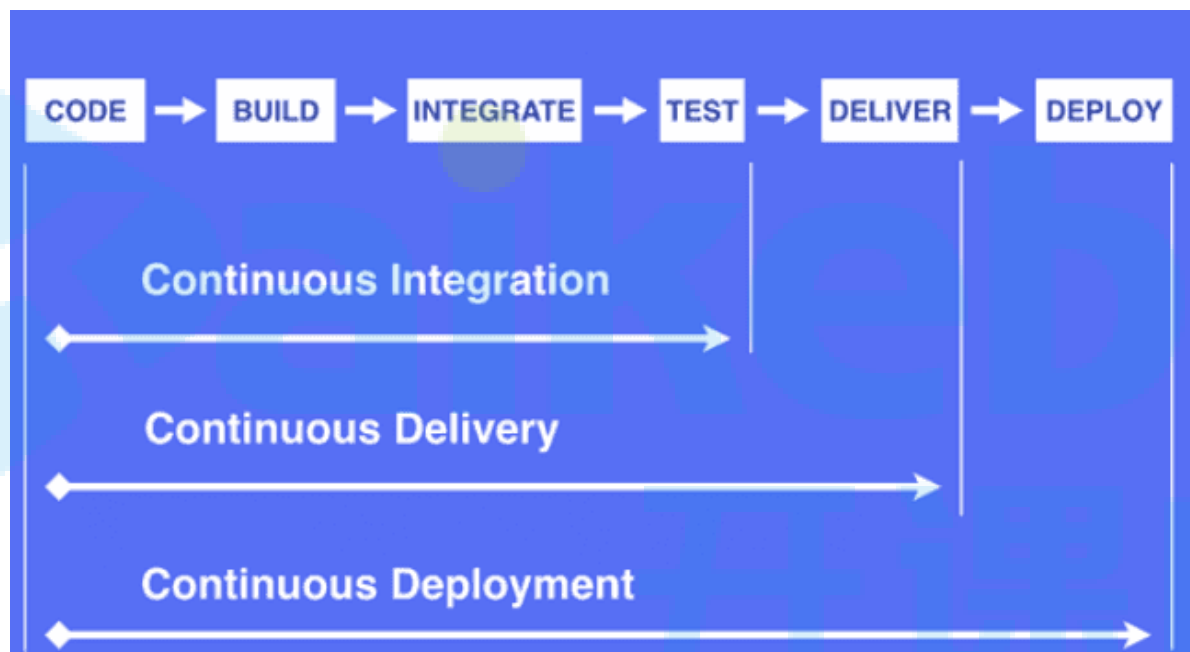
一、什么是持续集成？

持续集成是一个开发的实践，需要开发人员定期集成代码到共享存储库。这个概念是为了消除发现的问题，后来出现在构建生命周期的问题。持续集成要求开发人员有频繁的构建。最常见的做法是，每当一个代码提交时，构建应该被触发。

1、CI/DI简介

互联网软件的开发和发布，已经形成了一套标准流程，假如把开发工作流程分为以下几个阶段：

编码 --> 构建 --> 集成 --> 测试 --> 交付 --> 部署



正如你在上图中看到，[持续集成(Continuous Integration)]、[持续交付(Continuous Delivery)]和[持续部署(Continuous Deployment)]有着不同的软件自动化交付周期。

2、持续集成[CI]

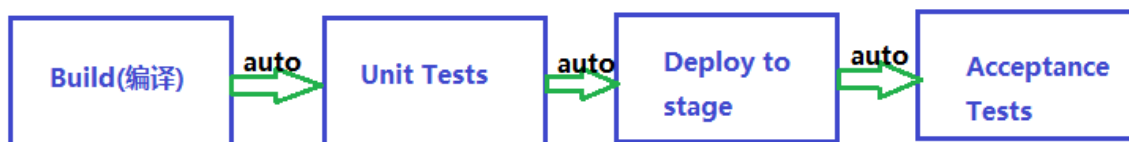
上面整个流程中最重要的组成部分就是持续集成（Continuous integration，简称CI）。

持续集成指的是，频繁地（一天多次）将代码集成到主干。将软件个人研发的部分向软件整体部分交付，频繁进行集成以便更快地发现其中的错误。

它的好处主要有两个：

1. 快速发现错误。每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易；
2. 防止分支大幅偏离主干。如果不是经常集成，主干又在不断更新，会导致以后集成的难度变大，甚至难以集成。

Continuous Integration



持续集成并不能消除Bug，而是让它们非常容易发现和改正。持续集成的目的，就是让产品可以快速迭代，同时还能保持高质量。它的核心措施是，代码集成到主干之前，必须通过自动化测试。只要有一个测试用例失败，就不能集成。

3、持续交付

持续交付（Continuous delivery）指的是，频繁地将软件的新版本，交付给质量团队或者用户，以供评审。如果评审通过，代码就进入生产阶段。

持续交付在持续集成的基础上，将集成后的代码部署到更贴近真实运行环境的「类生产环境」(production-like environments)中。持续交付优先于整个产品生命周期的软件部署，建立在高水平自动化持续集成之上



持续交付可以看作持续集成的下一步。它强调的是，不管怎么更新，软件是随时随地可以交付的。

4、持续部署[DI]

持续部署（continuous deployment）是持续交付的下一步，指的是代码通过评审以后，自动部署到生产环境。

持续部署的目标是，代码在任何时刻都是可部署的，可以进入生产阶段。

持续部署的前提是能自动化完成测试、构建、部署等步骤。

Continuous Deployment



总的来说，持续集成、持续交付、持续部署提供了一个优秀的 DevOps 环境。对于整个开发团队来说，能极大地提升开发效率，好处与挑战并行。无论如何，频繁部署、快速交付以及开发测试流程自动化都将成为未来软件工程的重要组成部分。

二、Jenkins

1、jenkins是什么？

Jenkins是一款开源 CI&CD 软件，用于自动化各种任务，包括构建、测试和部署软件。Jenkins 支持各种运行方式，可通过系统包、Docker 或者通过一个独立的 Java 程序

官网: <https://jenkins.io/> 官方文档: <https://jenkins.io/doc/>

Jenkins特性:

开源的java语言开发持续集成工具，支持CI，CD:

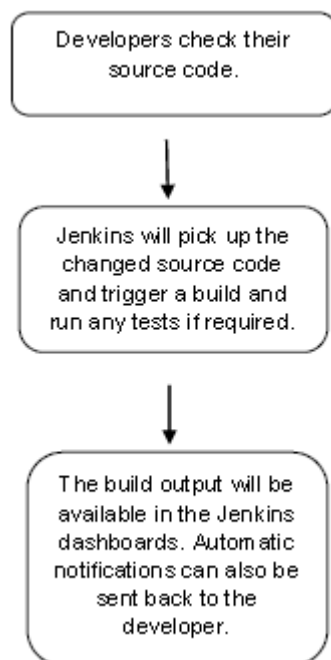
易于安装部署配置: 可通过yum安装, 或下载war包以及通过docker容器等快速实现安装部署, 可方便web界面配置管理;

消息通知及测试报告: 集成RSS/E-mail通过RSS发布构建结果或当构建完成时通过e-mail通知, 生成JUnit/TestNG测试报告;

分布式构建: 支持Jenkins能够让多台计算机一起构建/测试;

文件识别: Jenkins能够跟踪哪次构建生成哪些jar, 哪次构建使用哪个版本的jar等;

丰富的插件支持: 支持扩展插件, 你可以开发适合自己团队使用的工具, 如git, svn, maven, docker等。



伴随着Jenkins, 有时人们还可能看到它与Hudson关联。Hudson是由 Sun Microsystems 开发的一个非常流行的开源, 基于Java 的持续集成工具, 后来被Oracle收购。Sun被Oracle收购之后, 一个从Hudson 源代码的分支由 Jenkins 创建出台

2、Jenkins安装

2.1、准备工作

1) 第一次使用 Jenkins，您需要：

- 机器要求：
 - 256 MB 内存，建议大于 512 MB
 - 10 GB 的硬盘空间（用于 Jenkins 和 Docker 镜像）
- 需要安装以下软件：
 - Java 8 (JRE 或者 JDK 都可以)
 - [Docker](#)（导航到网站顶部的Get Docker链接以访问适合您平台的Docker下载）

2) 下载并运行 Jenkins

1. [下载 Jenkins](#).
2. 打开终端进入到下载目录.
3. 运行命令 `java -jar jenkins.war`.
4. 打开浏览器进入链接 `http://localhost:8080`.
5. 按照说明完成安装.

2.2、Jenkins启动

2.2.1、离线问题

离线

该Jenkins实例似乎已离线。

参考 [离线Jenkins安装文档](#)了解未接入互联网时安装Jenkins的更多信息。

可以通过配置一个代理或跳过插件安装来选择继续。

配置代理

跳过插件安装

发现jenkins处于离线状态（且在后台启动其实已经报错），无法安装相应的插件，因此需要解决此离线问题：

```
javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid
certification path to requested target
```

原因是就是升级站点的链接<https://updates.jenkins.io/update-center.json>默认是https的，如何解决呢？

2.2.2、解决方案

第一步：打开配置页面

<http://localhost:8080/pluginManager/advanced>

进去以后是这样的

Jenkins 1

Jenkins 插件管理

返回到工作台 系统管理 更新中心

可更新 可选插件 已安装 高级

代理设置

服务器

端口

上传

升级站点

URL

提交

改为<http://updates.jenkins.io/update-center.json>
就是把https改为http,其它的都不变

最后别忘了提交啊

第三步：重启Jenkins

OK，重启后再次输入密码进入就是正常的啦，正常的页面如下：

自定义Jenkins

插件通过附加特性来扩展Jenkins以满足不同的需求。

安装推荐的插件

安装Jenkins社区推荐的插件。

选择插件来安装

选择并安装最适合的插件。

Jenkins 2.176.2

2.3、访问jenkins

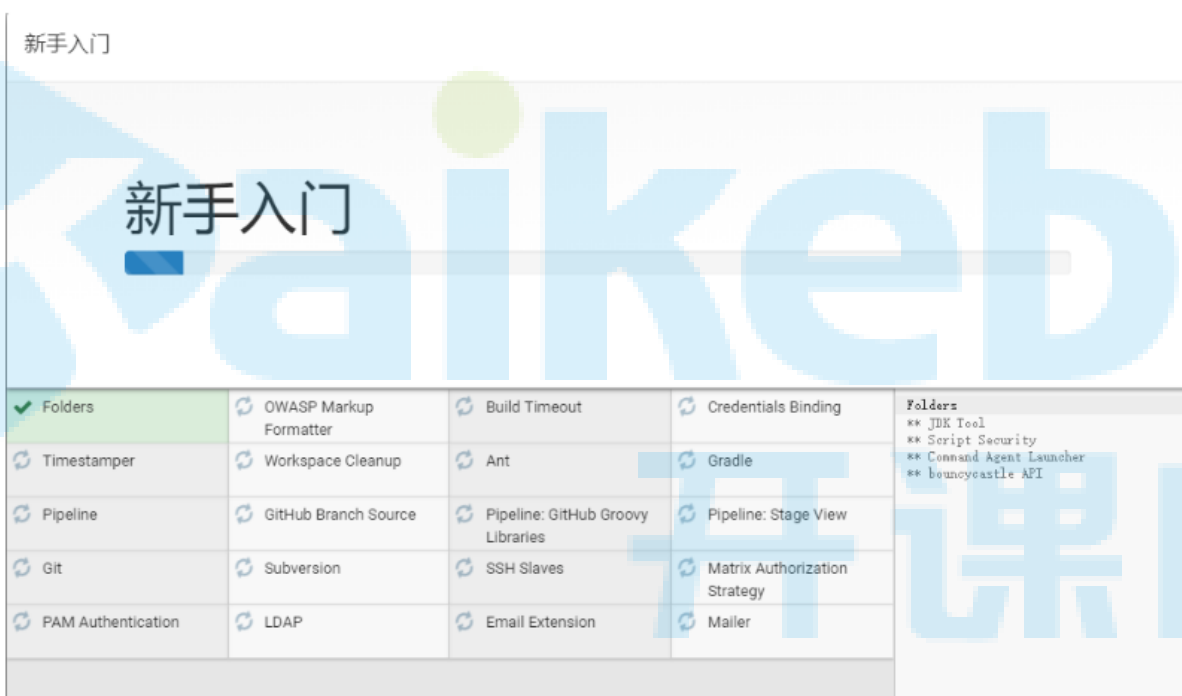
打开浏览器，访问ip: 8080进行安装，ip为linux机器ip



提示需要输入管理员密码，输入 77faa20f2ad544f7bcb6593b1cf1436b，点击 继续，会初始化一小段时间



这里我选择 安装推荐的插件



上面等插件安装完后，就进入到下面界面，提示要创建第一个admin用户，并设置用户名密码

创建第一个管理员用户

用户名:

密码:

确认密码:

全名:

电子邮件地址:

这里我直接创建用户名为admin，密码自定义：

创建第一个管理员用户

用户名:

密码:

确认密码:

全名:

电子邮件地址:

点击 [保存并完成](#) ,然后进行实例配置：

实例配置

Jenkins URL:

Jenkins URL 用于给各种Jenkins资源提供绝对路径链接的根地址。这意味着对于很多Jenkins特色是需要正确设置的，例如：邮件通知、PR状态更新以及提供给构建步骤的BUILD_URL环境变量。

推荐的默认值显示在尚未保存，如果可能的话这是根据当前请求生成的。最佳实践是要设置这个值，用户可能会需要用到，这将会避免在分享或者查看链接时的困惑。

提示配置jenkins URL，这里保持默认即可，继续点击 [保存并完成](#)

Jenkins已就绪！

Jenkins安装已完成。

开始使用Jenkins

提示jenkins已经就绪，现在就可以开始使用jenkins了：



2.4、Jenkins配置

#查看jenkins的配置文件，定义了home、JAVA_CMD、user、port等基础配置，保持默认即可

```
[root@lzx ~]# cat /etc/sysconfig/jenkins
```

```
## Path:      Development/Jenkins
```

```
## Description: Jenkins Automation Server
```

```
## Type:      string
```

```
## Default:   "/var/lib/jenkins"
```

```
## ServiceRestart: jenkins
```

```
#
```

```
# Directory where Jenkins store its configuration and working
```

```
# files (checkouts, build reports, artifacts, ...).
```

```
#
```

```
JENKINS_HOME="/var/lib/jenkins"
```

```
## Type:      string
```

```
## Default:   ""
```

```
## ServiceRestart: jenkins
```

```
#
```

```
# Java executable to run Jenkins
```

```
# When left empty, we'll try to find the suitable Java.
```

```
#
```

```
JENKINS_JAVA_CMD=""
```

```
## Type:      string
```

```
## Default:      "jenkins"
## ServiceRestart: jenkins
#
# Unix user account that runs the Jenkins daemon
# Be careful when you change this, as you need to update
# permissions of $JENKINS_HOME and /var/log/jenkins.
#
JENKINS_USER="jenkins"

## Type:         string
## Default:      "false"
## ServiceRestart: jenkins
#
# Whether to skip potentially long-running chown at the
# $JENKINS_HOME location. Do not enable this, "true", unless
# you know what you're doing. See JENKINS-23273.
#
JENKINS_INSTALL_SKIP_CHOWN="false"

## Type: string
## Default:      "-Djava.awt.headless=true"
## ServiceRestart: jenkins
#
# Options to pass to java when running Jenkins.
#
JENKINS_JAVA_OPTIONS="-Djava.awt.headless=true"

## Type:         integer(0:65535)
## Default:      8080
## ServiceRestart: jenkins
#
# Port Jenkins is listening on.
# Set to -1 to disable
#
JENKINS_PORT="8080"

## Type:         string
## Default:      ""
## ServiceRestart: jenkins
#
# IP address Jenkins listens on for HTTP requests.
# Default is all interfaces (0.0.0.0).
#
JENKINS_LISTEN_ADDRESS=""

## Type:         integer(0:65535)
## Default:      ""
## ServiceRestart: jenkins
#
# HTTPS port Jenkins is listening on.
# Default is disabled.
#
JENKINS_HTTPS_PORT=""

## Type:         string
## Default:      ""
## ServiceRestart: jenkins
#
```

```
# Path to the keystore in JKS format (as created by the JDK 'keytool').
# Default is disabled.
#
JENKINS_HTTPS_KEYSTORE=""

## Type:      string
## Default:   ""
## ServiceRestart: jenkins
#
# Password to access the keystore defined in JENKINS_HTTPS_KEYSTORE.
# Default is disabled.
#
JENKINS_HTTPS_KEYSTORE_PASSWORD=""

## Type:      string
## Default:   ""
## ServiceRestart: jenkins
#
# IP address Jenkins listens on for HTTPS requests.
# Default is disabled.
#
JENKINS_HTTPS_LISTEN_ADDRESS=""

## Type:      integer(1:9)
## Default:   5
## ServiceRestart: jenkins
#
# Debug level for logs -- the higher the value, the more verbose.
# 5 is INFO.
#
JENKINS_DEBUG_LEVEL="5"

## Type:      yesno
## Default:   no
## ServiceRestart: jenkins
#
# Whether to enable access logging or not.
#
JENKINS_ENABLE_ACCESS_LOG="no"

## Type:      integer
## Default:   100
## ServiceRestart: jenkins
#
# Maximum number of HTTP worker threads.
#
JENKINS_HANDLER_MAX="100"

## Type:      integer
## Default:   20
## ServiceRestart: jenkins
#
# Maximum number of idle HTTP worker threads.
#
JENKINS_HANDLER_IDLE="20"

## Type:      string
```

```
## Default:      ""
## ServiceRestart: jenkins
#
# Pass arbitrary arguments to Jenkins.
# Full option list: java -jar jenkins.war --help
#
JENKINS_ARGS=""
```

2.5、主程序目录

```
[root@lzx ~]# ls ~/.jenkins/           //查看程序主目录
config.xml                             nodes
hudson.model.UpdateCenter.xml          plugins
hudson.plugins.git.GitTool.xml         queue.xml.bak
identity.key.enc                      secret.key
jenkins.CLI.xml                       secret.key.not-so-secret
jenkins.install.InstallUtil.lastExecVersion secrets
jenkins.install.UpgradeWizard.state   updates
jenkins.model.JenkinsLocationConfiguration.xml userContent
jobs                                  users
logs                                workflow-labs
nodeMonitors.xml
```

jobs 浏览器上面创建的任务都会存放在这里

logs 存放jenkins相关的日志

nodes 多节点时用到

plugins 插件所在目录

secrets 密码密钥所在目录 //jobs和plugins目录比较重要

2.6、JDK配置



Global Tool Configuration

Maven 配置

默认 settings 提供	使用默认 Maven 设置
默认全局 settings 提供	使用默认 Maven 全局设置

JDK

JDK 安装	新增 JDK
系统下JDK 安装列表	

Git

Git installations	
Git	
Name	Default
Path to Git executable	git
<input type="checkbox"/> 自动安装	

JDK安装配置: [先把JDK安装到Linux服务器, 且配置完善]

JDK

JDK 安装

新增 JDK

JDK

别名

JAVA_HOME

☐ Install automatically

2.7、maven配置

2.7.1、安装maven

官网地址: <http://maven.apache.org/download.cgi>

下载地址: apache-maven-3.3.9-bin.tar.gz

配置环境变量: vim /etc/profile 写到最后, 我用的3.3.3的版本

```
export M2_HOME=/data/apache-maven-3.3.3
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
```

```
unset i
unset -f pathmunge
export JAVA_HOME=/usr/local/jdk1.8.0_65
export NODE_HOME=/opt/node-v9.9.0-linux-x64
export MAVEN_HOME=/usr/local/maven/apache-maven-3.6.1
export PATH=$JAVA_HOME/bin:$PATH:$NODE_HOME/bin:$MAVEN_HOME/bin
"/etc/profile" 80L, 1997C written
[root@jackhu maven]# source /etc/profile
[root@jackhu maven]# mvn -version
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-05T03:00:29+08:00)
Maven home: /usr/local/maven/apache-maven-3.6.1
Java version: 1.8.0_65, vendor: Oracle Corporation, runtime: /usr/local/jdk1.8.0_65/jre
Default locale: zh_CN, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-514.el7.x86_64", arch: "amd64", family: "unix"
```

2.7.2、Jenkins配置maven

Maven

Maven 安装

新增 Maven

Maven

Name

MAVEN_HOME

☐ Install automatically

2.8、插件安装

如果jenkins需要安装什么插件，可以直接安装即可。



1) Maven Integration plugin 安装此插件才能构建maven项目

Maven Integration plugin

This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven. Automatic triggers between projects depending on SNAPSHOTs. automated

3.2

2) Deploy to container Plugin 安装此插件，才能将打好的包部署到tomcat上

Deploy to container Plugin



This plugin allows you to deploy a war to a container after a successful build.

2.9、系统设置



这个没有话看一下上面的插件，没安装maven的插件就没有这个配置：



这个配置费劲九牛二虎之力，在N次报错后遍寻错误无解时候，在系统提供的英文文档里面偶然看到其中一个回答，好长好长看到其中一句，试了试竟然成功了，沃德天！

2.10、git配置

2.10.1、git地址错误



原因分析：这是由于git客户端版本过低造成的！或者系统中没有安装git所造成的

Jenkins本机默认使用"yum install -y git" 安装的git版本比较低，应该自行安装更高版本的git

2.10.2、安装git

安装git

```
[root@jenkins ~]# yum -y install libcurl-devel expat-devel curl-devel gettext-  
devel openssl-devel zlib-devel  
[root@jenkins ~]# yum -y install gcc perl-ExtUtils-MakeMaker  
[root@jenkins ~]# cd /usr/local/src/  
[root@jenkins src]# wget  
https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.1.1.tar.gz  
[root@jenkins src]# tar -zxvf git-2.1.1.tar.gz  
[root@jenkins src]# cd git-2.1.1  
[root@jenkins git-2.1.1]# make prefix=/usr/local/git all  
[root@jenkins git-2.1.1]# make prefix=/usr/local/git install
```

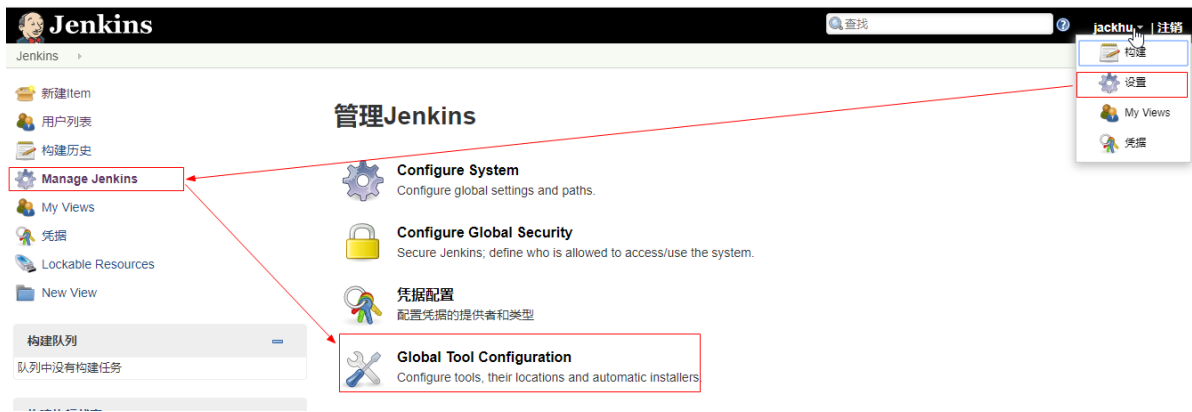
添加git到环境变量

```
`[root@jenkins git-2.1.1]`# echo "export PATH=$PATH:/usr/local/git/bin" >>  
/etc/bashrc`[root@jenkins git-2.1.1]`# source /etc/bashrc`
```

查看更新后的git版本和所在路径

```
`[root@jenkins ~]`# git --version`git version 2.1.1`[root@jenkins ~]`#  
whereis git`git: `/usr/local/git`
```

2.10.3、设置git

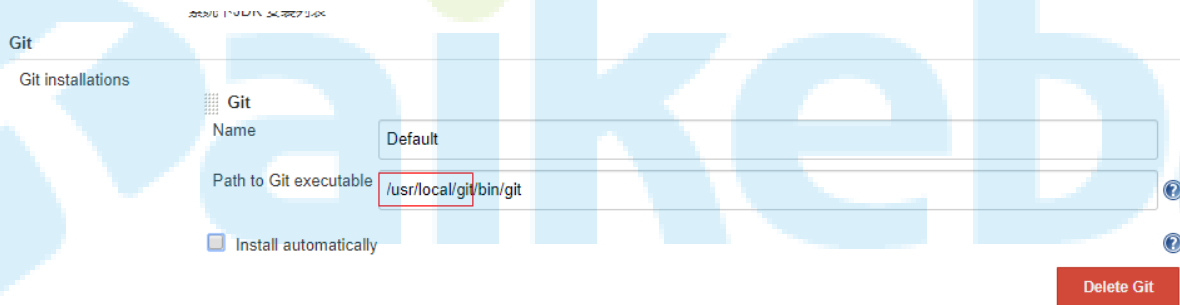


查看git安装地址:

```
git version 1.8.3.1
[root@jackhu git-2.1.1]# whereis git
git: /usr/bin/git /usr/local/git /usr/share/man/man1/git.1.gz
[root@jackhu git-2.1.1]#
```

设置git安装地址:

依次打开"系统管理" -> "系统设置" -> "Git" -> "Path to Git executable", 在此处填入"whereis git"查询出的地址 + "/bin/git" (如上面"whereis git"的地址为"/usr/local/git", 则应该填入"/usr/local/git/bin/git") 并保存



3、docker安装

3.1、安装文档

jenkins.io/zh/doc/book/installing/

Jenkins

- 用 npm 构建 Node.js 和 React 应用
- 用 PyInstaller 构建 Python 应用
- 用 Blue Ocean 创建流水线
- 构建一个多分支的 Pipeline 项目

用户手册 (PDF)

- 用户手册概述
- 安装 Jenkins**
- 使用 Jenkins
- 流水线
- Blue Ocean
- 管理 Jenkins
- 系统管理
- 规模 Jenkins
- 附录
- 术语表

系统要求

最低推荐配置:

- 256MB 可用内存
- 1GB 可用磁盘空间 (作为一个 Docker 容器运行 Jenkins 的话推荐 10GB)

为小团队推荐的硬件配置:

- 1GB+ 可用内存
- 50 GB+ 可用磁盘空间

软件配置:

- Java 8 - 无论是 Java 运行时环境 (JRE) 还是 Java 开发工具包 (JDK) 都可以。

注意: 如果将 Jenkins 作为 Docker 容器运行, 这不是必需的

WAR 文件

苹果系统

Linux

- Debian/Ubuntu

Windows

其他操作系统

- OpenIndiana Hipster
- Solaris, OmniOS, SmartOS, and other siblings

安装后设置向导

- 解锁 Jenkins
- 自定义 Jenkins 插件
- 创建第一个管理员用户

设置映射目录:

macOS和Linux上

1. 打开一个终端窗口。

2. 下载 `jenkinsci/blueocean` 镜像并使用以下 `docker run` 命令将其作为Docker中的容器运行：

```
docker run \
  -u root \
  --rm ❶
  -d ❷
  -p 8080:8080 ❸
  -p 50000:50000 ❹
  -v jenkins-data:/var/jenkins_home ❺
  -v /var/run/docker.sock:/var/run/docker.sock ❻
  jenkinsci/blueocean ❼
```

此目录必须在本地Linux系统进行指定
比如：/root/.jenkins 指定为此目录，否则无法再宿主机获取密码

- ❶ (可选) `jenkinsci/blueocean` 关闭时自动删除Docker容器(下图为实例)。如果您需要退出Jenkins，这可以保持整洁。
 - ❷ (可选) `jenkinsci/blueocean` 在后台运行容器(即“分离”模式)并输出容器ID。如果您不指定此选项，则在终端窗口中输出正在运行的此容器的Docker日志。
 - ❸ 映射(例如“发布”) `jenkinsci/blueocean` 容器的端口8080到主机上的端口8080。第一个数字代表主机上的端口，而最后一个代表容器的端口。因此，如果您为此选项指定 `-p 49000:8080`，您将通过端口49000访问主机上的Jenkins。
- (可选) 将 `jenkinsci/blueocean` 容器的端口50000映射到主机上的端口50000。如果您在其他机器上设置了一个或多个基于INI P的Jenkins代理程序，而这些代理程序又与 `jenkinsci/blueocean` 容器交互(充当“主”Jenkins服务器，或者

安装还是比较速度的：

新手入门

<input checked="" type="checkbox"/> Folders Plugin	<input checked="" type="checkbox"/> OWASP Markup Formatter Plugin	<input checked="" type="checkbox"/> Build Timeout	<input checked="" type="checkbox"/> Credentials Binding Plugin	Folders
<input checked="" type="checkbox"/> Timestamper	<input checked="" type="checkbox"/> Workspace Cleanup	<input type="checkbox"/> Ant	<input type="checkbox"/> Gradle	OWASP Markup Formatter
<input checked="" type="checkbox"/> Pipeline	<input type="checkbox"/> GitHub Branch Source Plugin	<input checked="" type="checkbox"/> Pipeline: GitHub Groovy Libraries	<input type="checkbox"/> Pipeline: Stage View	Build Timeout
<input type="checkbox"/> Git plugin	<input type="checkbox"/> Subversion	<input type="checkbox"/> SSH Slaves	<input checked="" type="checkbox"/> Matrix Authorization Strategy Plugin	Credentials Binding
<input type="checkbox"/> PAM Authentication	<input type="checkbox"/> LDAP	<input type="checkbox"/> Email Extension	<input type="checkbox"/> Mailer Plugin	Timestamper
<input type="checkbox"/> Localization: Chinese (Simplified)				** Resource Disposer
				Workspace Cleanup

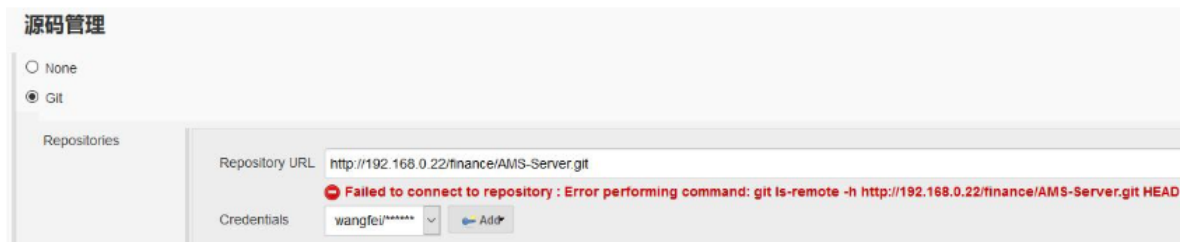
3.2、安装命令

#使用docker安装时必须映射maven,jdk, jenkins在配置时需要

```
docker run \
  -u root \
  --rm \
  -d \
  -p 8080:8080 \
  -p 50000:50000 \
  -v /root/.jenkins:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  # git目录映射非必须选项，且docker容器化git也无法映射
  #-v /usr/local/git:/usr/local/git \
  -v /usr/local/jdk1.8.0_65:/usr/local/jdk1.8 \
  -v /usr/local/maven/apache-maven-3.6.1:/usr/local/maven \
```

3.3、错误提示

1) 无法使用宿主机的git环境问题



问题描述: docker安装Jenkins, 无法使用宿主机的git环境问题。

解决 方案: docker安装Jenkins, 其实不需要使用宿主机git环境, 因为在安全jenkins的时候已经安装了git插件, 所以不需要配置宿主机git.

2) 无法使用宿主机Java环境

错误描述: Cannot run program "/usr/local/jdk/bin/java" (in directory "/var/jenkins_home/workspace/Ccloud预生产"): error=2, No such file or directory

```
ERROR: Failed to parse POMs
java.io.IOException: Cannot run program "/usr/local/jdk1.8/bin/java" (in directory "/var/jenkins_home/workspace/one-ee"): error=2, No
such file or directory
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:1048)
    at hudson.Proc$LocalProc.<init>(Proc.java:250)
    at hudson.Proc$LocalProc.<init>(Proc.java:219)
    at hudson.Launcher$LocalLauncher.launch(Launcher.java:937)
```

原因: /usr/local/jdk/bin/java找不到或者不存在

解决办法: 删掉jenkins全局配置里的JDK配置。docker jenkins 是一个专用来做部署的容器, 自带Java, 不需要额外配置JDK

3) jenkins时间与北京时间不一致 (早8个小时) 的解决办法

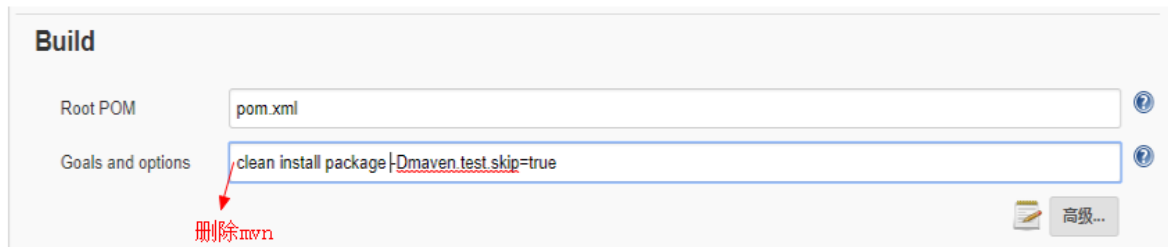
docker启动jenkins时加上 `-e JAVA_OPTS=-Duser.timezone=Asia/Shanghai`
`docker run ... -e JAVA_OPTS=-Duser.timezone=Asia/Shanghai`

4) pushImage

Unknown lifecycle phase "mvn". You must specify a valid lifecycle phase or a goal in the format <plugin-prefix>:<goal> or <plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal>. Available lifecycle phases are: validate, initialize, generate-sources, process-sources, generate-resources, process-resources, compile, process-classes, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-test-classes, test, prepare-package, package, pre-integration-test, integration-test, post-integration-test, verify, install, deploy, pre-clean, clean, post-clean, pre-site, site, post-site, site-deploy. -> [Help 1]

以上是Jenkins报的错，我的报错原因在于项目构建配置的Build项添加了mvn。Jenkins是已经加上了自动mvn命令符的(和Eclipse里面的run as等一样)，去掉mvn命令符，问题解决

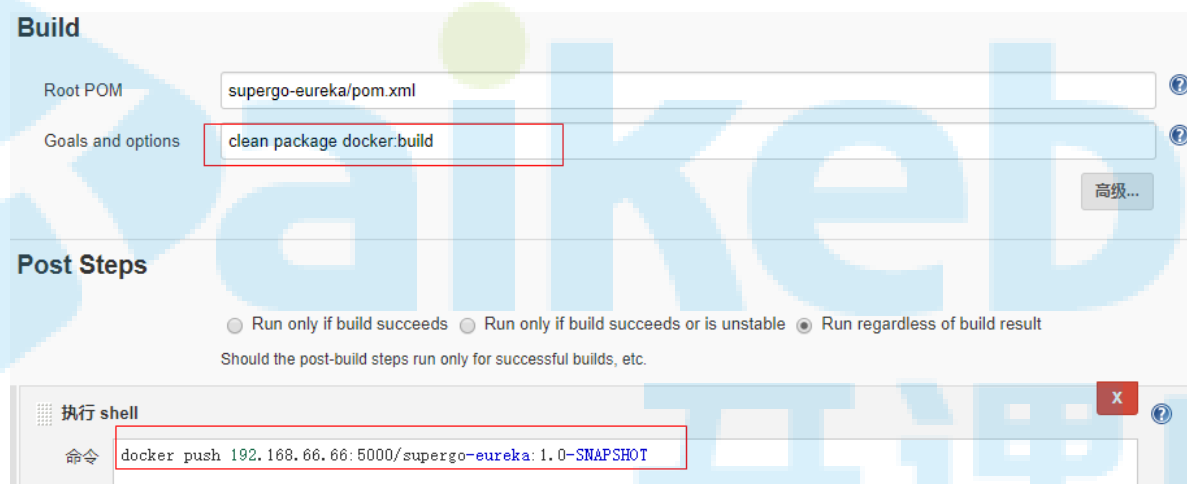
修改后的Build:



The screenshot shows the Jenkins 'Build' configuration page. The 'Root POM' field contains 'pom.xml'. The 'Goals and options' field contains 'clean install package -Dmaven.test.skip=true'. A red arrow points to the '-Dmaven.test.skip=true' part, with the text '删除mvn' (Delete mvn) written below it. There is a '高级...' (Advanced...) button on the right.

注意：Jenkins已经安装了maven插件，但是必须配置maven插件，但是不需要写mvn

5) Unknown lifecycle phase "-DpushImage"



The screenshot shows the Jenkins 'Build' configuration page. The 'Root POM' field contains 'supergo-eureka/pom.xml'. The 'Goals and options' field contains 'clean package docker:build'. Below the 'Build' section is the 'Post Steps' section with three radio buttons: 'Run only if build succeeds', 'Run only if build succeeds or is unstable', and 'Run regardless of build result'. The 'Run regardless of build result' option is selected. Below the radio buttons is the text 'Should the post-build steps run only for successful builds, etc.'. At the bottom is the '执行 shell' (Execute shell) section with a text box containing the command 'docker push 192.168.66.66:5000/supergo-eureka:1.0-SNAPSHOT'.

三、持续构建

1、构建maven项目

点击新建，出现下图，名字随便起，选择构建一个maven项目

输入一个任务名称

kkb-test

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

构建一个maven项目
构建一个maven项目。Jenkins利用你的POM文件,这样可以大大减轻构建配置。

流水线
精心地组织一个可以长期运行在多个节点上的任务。适用于构建流水线(更加正式地应当称为工作流),增加或者组织难以采用自由风格的任务类型。

构建一个多配置项目
适用于多配置项目,例如多环境测试,平台指定构建,等等。

GitHub 组织
扫描一个 GitHub 组织(或者个人账户)的所有仓库来匹配已定义的标记。

确定

2、构建配置

General 源码管理 构建触发器 构建环境 Pre Steps Build Post Steps 构建设置 构建后操作

描述

test

[Plain text] 预览

☐ Discard old builds

☒ GitHub 项目

项目 URL

https://gitee.com/ithubin/jackhu_one.git

源码地址:

Git

Repositories

Repository URL

https://gitee.com/ithubin/jackhu_one.git

Credentials

jackhu/***** 添加

高级...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

构建触发器:

构建触发器

- ☒ Build whenever a SNAPSHOT dependency is built
- ☐ Schedule build when some upstream has no successful builds
- ☐ 触发远程构建 (例如,使用脚本)
- ☐ Build after other projects are built
- ☐ Build periodically

3、构建成功

```
[INFO] Reactor Summary for supergo-parent 1.0-SNAPSHOT:
[INFO]
[INFO] supergo-parent ..... SUCCESS [ 2.125 s]
[INFO] supergo-common ..... SUCCESS [ 8.197 s]
[INFO] supergo-pojo ..... SUCCESS [ 3.255 s]
[INFO] supergo-mapper ..... SUCCESS [ 1.536 s]
[INFO] supergo-manager ..... SUCCESS [ 0.074 s]
[INFO] supergo-manager-feign ..... SUCCESS [ 2.311 s]
[INFO] supergo-manager-web ..... SUCCESS [ 7.926 s]
[INFO] supergo-eureka ..... SUCCESS [ 3.055 s]
[INFO] supergo-monitor ..... SUCCESS [ 1.589 s]
[INFO] supergo-auth ..... SUCCESS [ 2.257 s]
[INFO] supergo-zuul ..... SUCCESS [10.296 s]
[INFO] supergo-base-service ..... SUCCESS [ 3.377 s]
[INFO] supergo-manager-service ..... SUCCESS [38.110 s]
[INFO] supergo-search ..... SUCCESS [ 0.065 s]
[INFO] supergo-search-feign ..... SUCCESS [ 3.333 s]
[INFO] supergo-search-web ..... SUCCESS [ 3.909 s]
[INFO] supergo-search-service ..... SUCCESS [01:22 min]
[INFO] supergo-cart ..... SUCCESS [ 0.062 s]
[INFO] supergo-cart-feign ..... SUCCESS [ 2.113 s]
[INFO] supergo-cart-web ..... SUCCESS [ 4.347 s]
```

四、持续部署

1、执行shell脚本

#磁盘空间不足，导致启动失败，必须加‘#!/bin/bash’，否则会启动不起来

```
#!/bin/bash
```

#export BUILD_ID=dontKillMe这一句很重要，这样指定了，项目启动之后才不会被Jenkins杀掉。

```
export BUILD_ID=dontKillMe
```

#指定最后编译好的jar存放的位置

```
www_path=/var/codespace/test/
```

#Jenkins中编译好的jar位置

```
jar_path=/root/.jenkins/workspace/test/supergo-eureka/target
```

#Jenkins中编译好的jar名称

```
jar_name=supergo-eureka-1.0-SNAPSHOT.jar
```

#获取运行编译好的进程ID，便于我们在重新部署项目的时候先杀掉以前的进程

```
pid=$(cat /var/codespace/test/test-web.pid)
```

#进入指定的编译好的jar的位置

```
cd ${jar_path}
```

#将编译好的jar复制到最后指定的位置

```
cp ${jar_path}/${jar_name} ${www_path}
```

#进入最后指定存放jar的位置

```
cd ${www_path}

#杀掉以前可能启动的项目进程
kill -9 ${pid}

#启动jar, 指定SpringBoot的profiles为test,后台启动
java -jar -Dspring.profiles.active=test ${jar_name} &

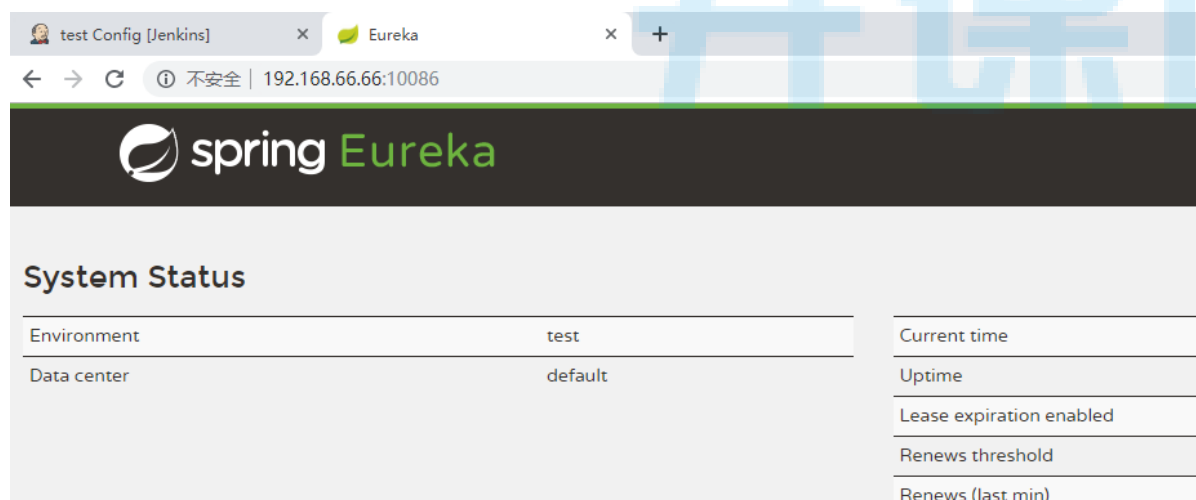
#将进程ID存入到shaw-web.pid文件中
echo $! > /var/codespace/test/test-web.pid
```

以上脚本执行完毕，就会自动启动相应的项目。配置位置如下：



注意：必须加上 `#!/bin/bash`，否则不能启动起来

2、执行完毕



执行完毕后，我们发现项目可以访问了

五、Jenkins&Docker

1、docker私有仓库

(1) 拉取私有仓库镜像

```
docker pull registry
```

(2) 启动私有仓库容器

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

(3) 打开浏览器

输入地址: http://192.168.66.66:5000/v2/_catalog

看到 `{"repositories":[]}` 表示私有仓库搭建成功并且内容为空

(4) 修改宿主机的docker配置, 让其可以远程访问

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

其中ExecStart=后添加配置:

```
/usr/bin/dockerd -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375 --  
insecure-registry 192.168.66.66:5000
```

修改后如下:

```
#修改daemon.json  
#vi /etc/docker/daemon.json  
#注意此种方式: 也可以, 但是某些版本无法识别, 具体情况, 如果无法识别, 使用第2种  
{"insecure-registries":["192.168.66.66:5000"]}  
  
#较高版本的docker容器配置如下所示:  
#命令: vi /lib/systemd/system/docker.service  
#配置方式, 此配置相当关键, 配置完后可上传镜像了。  
ExecStart=/usr/bin/dockerd -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375  
--insecure-registry 192.168.66.66:5000
```

此步用于让 docker信任私有仓库地址。配置完毕结果如下所示:

```
[root@jackhu ~]# vi /lib/systemd/system/docker.service  
[Unit]  
Description=Docker Application Container Engine  
Documentation=https://docs.docker.com  
BindsTo=containerd.service  
After=network-online.target firewalld.service containerd.service  
Wants=network-online.target  
Requires=docker.socket  
  
[Service]  
Type=notify  
# the default is not to use systemd for cgroups because the delegate issues still  
# exists and systemd currently does not support the cgroup feature set required  
# for containers run by docker  
#ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock  
ExecStart=/usr/bin/dockerd -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375 --insecure-registry 192.168.66.66:5000  
ExecReload=/bin/kill -s HUP $MAINPID  
TimeoutSec=0
```

(5) 重启docker 服务

```
#刷新docker.service配置
systemctl daemon-reload
#重新启动
systemctl restart docker
```

2、上传镜像

(1) 标记此镜像为私有仓库的镜像

```
docker tag jdk1.8 192.168.66.66:5000/jdk1.8
```

2) 上传标记的镜像

```
docker push 192.168.66.66:5000/jdk1.8
```

3) 删除镜像

简单的删除方法:

```
#删除registry中镜像, 可以发现删除成功
docker exec registry rm -rf
/var/lib/registry/docker/registry/v2/repositories/supergo-eureka
#清除blob文件
docker exec registry bin/registry garbage-collect
/etc/docker/registry/config.yml
```

复杂的删除方法 (不一定删除成功) 【只作为参考】

查询Dgist:

```
#通过 /v2/<镜像名称>/manifests/<tag>的方式获取镜像的digest
curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET
http://192.168.66.66:5000/v2/supergo-eureka/manifests/1.0-SNAPSHOT 2>&1 | grep
Docker-Content-Digest | awk '{print ($3)}'
```

执行以上命令输出SHA256加密的Digest:

```
java          latest          025b0f5b1010          2 years ago          645MB
[root@jackhu test]# curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET
eureka/manifests/0.0.1 2>&1 | grep Docker-Content-Digest | awk '{print ($3)}'
[root@jackhu test]# curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET
eureka/manifests/1.0-SNAPSHOT 2>&1 | grep Docker-Content-Digest | awk '{print ($3)}'
sha256:549f7fc9167fc42ebbbd0aefbb674656c929e079b8c63b4546af4da80f02ef26
[root@jackhu test]#
```

删除镜像:


```
#根据sha256结果删除镜像
curl -v --silent -H "Accept:
application/vnd.docker.distribution.manifest.v2+json" -X DELETE
http://192.168.66.66:5000/v2/supergo-
eureka/manifests/sha256:549f7fc9167fc42ebbbd0aefbb674656c929e079b8c63b4546af4da8
0f02ef26
```

删除镜像方法说明:

```
1)、通过 /v2/<镜像名称>/manifests/<tag>的方式获取镜像的digest
注意获取的请求头中需要加Accept: application/vnd.docker.distribution.manifest.v2+json
否则获取不到正确的digest (虽然也能够获取到一个digest, 但是之后的删除操作会失败)
没加那个header值导致失败的返回值是:
404{"errors":[{"code":"MANIFEST_UNKNOWN","message":"manifest unknown"}]
2)、删除镜像
DELETE /v2/<name>/manifests/<SHA256>
```

删除错误说明:

```
root@jackhu test]# curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X DELETE http://
eureka/manifests/sha256:549f7fc9167fc42ebbbd0aefbb674656c929e079b8c63b4546af4da80f02ef26
* About to connect() to 192.168.66.66 port 5000 (#0)
* Trying 192.168.66.66...
* Connected to 192.168.66.66 (192.168.66.66) port 5000 (#0)
* DELETE /v2/supergo-eureka/manifests/sha256:549f7fc9167fc42ebbbd0aefbb674656c929e079b8c63b4546af4da80f02ef26 HTTP/1.1
* User-Agent: curl/7.29.0
* Host: 192.168.66.66:5000
* Accept: application/vnd.docker.distribution.manifest.v2+json
< HTTP/1.1 405 Method Not Allowed
< Content-Type: application/json; charset=utf-8
< Docker-Distribution-API-Version: registry/2.0
< X-Content-Type-Options: nosniff
< Date: Tue, 26 Nov 2019 09:49:03 GMT
< Content-Length: 78
{"errors":[{"code":"UNSUPPORTED","message":"The operation is unsupported."}]}
Connection #0 to host 192.168.66.66 left intact
root@jackhu test]# docker exec -it registry /bin/bash
OCI runtime exec failed: exec failed: container_linux.go:345: starting container process caused "exec: \"/bin/bash\": st
directory": unknown
root@jackhu test]# docker exec -it registry
docker exec requires at least 2 arguments
```

这种情况是私有仓库不支持删除操作, 需要在配置文件config.yml中增加delete:enabled:true字段

```
#登录容器
docker exec -it registry sh
#配置文件
vi /etc/docker/registry/config.yml
```

配置结果如下所示:

```
version: 0.1
log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
http:
  addr: :5000
  headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
delete:
  enabled: true
```

3、自动部署

3.1、部署方法

微服务部署有两种方法：

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

(2) 通过Maven插件自动部署。

对于数量众多的微服务，手动部署无疑是非常麻烦的做法，并且容易出错。

所以我们这里学习如何自动部署，这也是企业实际开发中经常使用的方法。

3.2、docker部署

Maven插件自动部署步骤：

(1) 工程pom.xml 增加配置

```
# 注意编码问题
<build>
  <finalName>eureka</finalName>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>com.spotify</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <version>0.4.13</version>
      <configuration>

        <imageName>192.168.66.66:5000/${project.artifactId}:${project.version}</imageName>

        <baseImage>java</baseImage>
        <entryPoint>["java", "-
jar", "-/${project.build.finalName}.jar"]</entryPoint>
        <resources>
          <resource>

            <targetPath></targetPath>
```

```

        <directory>${project.build.directory}
    </directory>

    <include>${project.build.finalName}.jar</include>

    </resource>
</resources>
<dockerHost>http://192.168.66.66:2375</dockerHost>
</configuration>
</plugin>
</plugins>
</build>

```

(2) 工程的src/main目录下创建docker目录，目录下创建Dockerfile文件，内容如下：

```

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

```

解释下这个配置文件：

- VOLUME 指定了临时文件目录为 /tmp。其效果是在主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的 /tmp。此步骤是可选的，如果涉及到文件系统的应用就很有必要了。/tmp 目录用来持久化到 Docker 数据文件夹，因为 SpringBoot 使用的内嵌 Tomcat 容器默认使用 /tmp 作为工作目录
- 项目的 jar 文件作为 “app.jar” 添加到容器的
- ENTRYPOINT 执行项目 app.jar。为了缩短 Tomcat 启动时间，添加一个系统属性指向 “/dev/urandom” 作为 Entropy Source

(3) 在windows的命令提示符下，进入工程所在的目录，输入以下命令，进行打包和上传镜像

```
mvn clean package docker:build -DpushImage #此命令放入jenkins中，将会自动打包进入容器
```

使用Jenkins上传镜像需要做如下配置：

Build

Root POM

supergo-eureka/pom.xml

?

Goals and options

clean package docker:build -DpushImage

?

高级...

执行后，会有如下输出，代码正在上传

```
c3fe59dd9556: Pushing [=====>] 353.6 MB
c3fe59dd9556: Pushing [=====>] 354.1 MB
c3fe59dd9556: Pushing [=====>] 354.7 MB
c3fe59dd9556: Pushing [=====>] 355.2 MB
c3fe59dd9556: Pushing [=====>] 355.7 MB
c3fe59dd9556: Pushing [=====>] 356.3 MB
c3fe59dd9556: Pushing [=====>] 356.7 MB
c3fe59dd9556: Pushed
```

浏览器访问 <http://192.168.184.135:5000/v2/catalog> , 输出

```
{"repositories":["项目名"]}
```

(4) 进入宿主机, 查看镜像

```
docker images
```



