



Jenkins

构建持续集成环境

Jenkins – Continuous Integration





CONTENT

01. Why CI

为什么要持续集成

02. Jenkins CI 环境搭建

使用Jenkins搭建持续集成环境



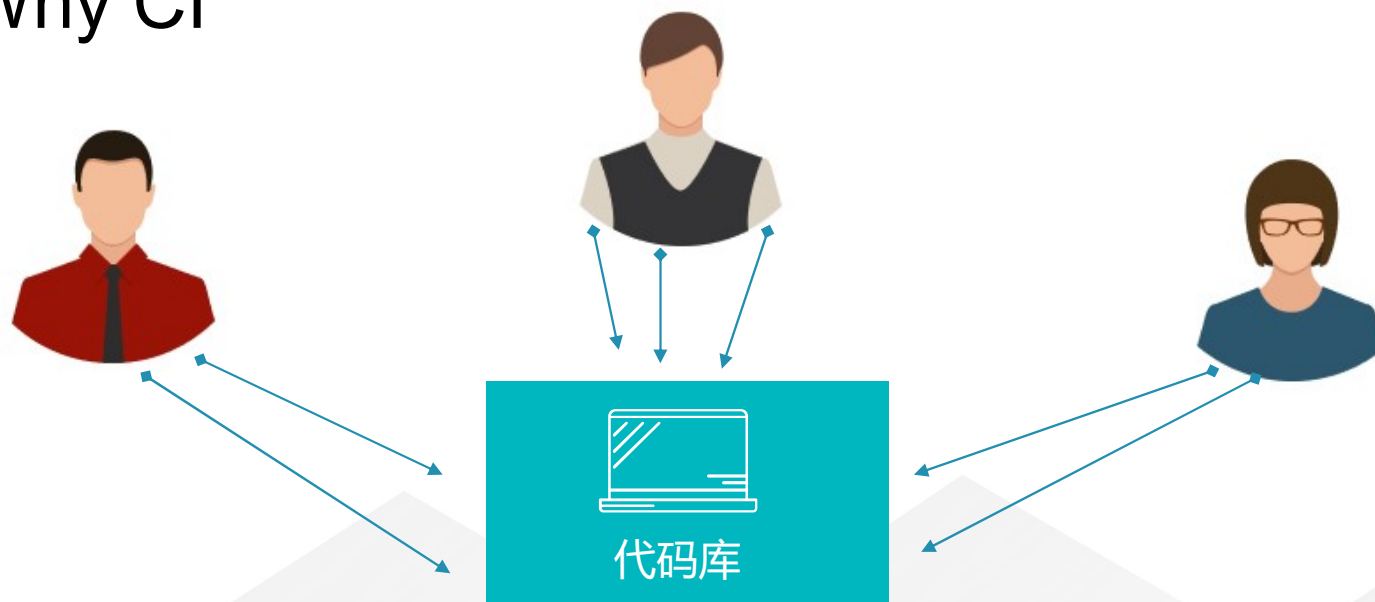
01

Why CI

为什么要持续集成



Why CI



BUG来自哪?

BUG!!



Why CI

“持续集成 (Continuous Integration)”

持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽快地发现集成错误。

每一次集成

- 向版本控制库提交代码
- 自动化的构建过程
—— 编译 分发 部署 测试
- 修改代码
- 构建成功

01



02



核心价值

- 减少风险，减少重复过程
- 任何时间、任何地点生成可部署的软件
- 增强项目的可见性
- 建立团队对开发产品的信心

02

Jenkins CI 环境搭建

使用Jenkins搭建持续集成环境





什么是Jenkins



Jenkins

- Jenkins 是一个开源项目，提供了一种易于使用的持续集成系统，使开发者从繁杂的集成中解脱出来，专注于更为重要的业务逻辑实现上。同时 Jenkins 能实施监控集成中存在的错误，提供详细的日志文件和提醒功能，还能用图表的形式形象地展示项目构建的趋势和稳定性。



新建一个任务

 **Jenkins**

Jenkins ▶

 新建任务

 用户

 构建历史

 系统管理

 我的视图

 Credentials

 新建视图

构建队列

队列中没有构建任务

构建执行状态

1 空闲


2 空闲





输入一个任务名称


mvn_test


» 必填项


 **构建一个自由风格的软件项目**
这是Jenkins的主要功能。Jenkins将会结合任何SCM和任何构建系统来构建你的项目，甚至可以构建软件以外的系统。

 **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 Organization**
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

 **Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.

 **文件夹**
创建一个可以嵌套存储的容器。利用它可以进行分组。视图仅仅是一个过滤器，而文件夹则是一个独立的命名空间，因此你可以有多个相同名称的内容，只要它们在不同的文件夹里即可。

确定

如果你根据一个已经存在的任务创建，可以使用这个选项



配置

General 源码管理 构建触发器 构建环境 构建 构建后操作

描述

simple maven project

[纯文本] 预览

☒ GitHub project

Project url

高级...

☐ Throttle builds

☐ 丢弃旧的构建

☐ 参数化构建过程

☐ 关闭构建

☐ 在必要的时候并发构建

高级...

General 配置中主要是写一下描述，我在这里写了示例项目的Github地址



配置源码管理

General **源码管理** 构建触发器 构建环境 构建 构建后操作

源码管理

☐ 无
☒ Git

Repositories

Repository URL

Credentials [Add](#)

[高级...](#)

[Add Repository](#)

Branches to build

Branch Specifier (blank for 'any')

[Add Branch](#)

源码库浏览器

Additional Behaviours [Add](#)

☐ Subversion

使用Git进行源码管理



配置触发器

构建触发器

- ☐ 触发远程构建 (例如,使用脚本)
- ☐ GitHub hook trigger for GITScm polling
- ☐ 其他工程构建后触发
- ☐ 定时构建

☒ 轮询 SCM

日程表

H/5 * * * *

上次运行的时间 2018年5月20日 星期日 下午04时58分40秒 CST; 下次运行的时间 2018年5月20日 星期日 下午05时03分40秒 CST

忽略钩子 post-commit ☐

定时构建

【周期】进行项目构建 (它不关心源码是否发生变化)

H 2 * * * (每天2:00 必须build一次源码)

轮询 SCM

【定时】检查源码变更, 如果有更新就checkout【最新代码】下来, 然后执行构建动作。

H/5 * * * * (每5分钟检查一次源码变化)



配置构建命令

构建

执行 Windows 批处理命令

命令

```
mvn install  
mvn compile  
mvn test
```

参阅 [可用环境变量列表](#)

高级...

增加构建步骤 ▾

这个取决于构建的环境
如果是Linux环境，就选择“执行Shell”

在这里输入需要Jenkins为我们自动执行的命令
一般是compile和test，需要注意的是构建环境也需要显式的写出命令install



构建

 **Jenkins**

Jenkins ▶ mvn test ▶

 返回面板

 状态

 修改记录

 工作空间

 **立即构建**

 删除工程

 配置

 Git 轮询日志

 GitHub

 重命名

 **Build History** **构建历史**

find x

 **#1** 2018-5-20 下午4:32

 [RSS 全部](#)  [RSS 失败](#)

TESTS

Running TestHello

Hello Maven

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.083 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ jenkins ---

[INFO] Building jar: D:\Jenkins\workspace\mvn test\target\jenkins-1.0-SNAPSHOT.jar

[INFO]

[INFO] --- maven-install-plugin:2.4:install (default-install) @ jenkins ---

[INFO] Installing D:\Jenkins\workspace\mvn test\target\jenkins-1.0-SNAPSHOT.jar to D:\Maven\repository\1yc\jenkins\1.0-SNAPSHOT\jenkins-1.0-SNAPSHOT.jar

[INFO] Installing D:\Jenkins\workspace\mvn test\pom.xml to D:\Maven\repository\1yc\jenkins\1.0-SNAPSHOT\jenkins-1.0-SNAPSHOT.pom

[INFO]

[INFO] BUILD SUCCESS

[INFO]

[INFO] Total time: 5.552 s

[INFO] Finished at: 2018-05-20T16:32:16+08:00

[INFO] Final Memory: 15M/115M

[INFO]

Finished: SUCCESS

[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform dependent!

[INFO] Compiling 1 source file to D:\Jenkins\workspace\mvn test\target\test-classes

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ jenkins ---

[INFO] Surefire report directory: D:\Jenkins\workspace\mvn test\target\surefire-reports

成功构建项目



Heroku

构建持续部署环境

Heroku – Continuous Deployment



CONTENT

03. Why CD

为什么要持续部署

04. Heroku CD 环境搭建

使用Heroku搭建持续部署环境



03

Why CD

为什么要持续部署



Why CD



持续部署 (Continuous Deployment)

持续部署是通过自动化的构建、测试和部署循环来快速交付高质量的产品。某种程度上代表了一个开发团队工程化的程度，毕竟快速运转的互联网公司人力成本会高于机器，投资机器优化开发流程化相对也提高了人的效率，让生产效率最大化。

每一次部署

- 向repo提交代码
- 编译、测试、上线、交付
- 部署成功

01



02



核心价值

- 快速发布
- 缩短编码->测试->上线->交付的迭代周期

04

Heroku CD 环境搭建

使用Heroku搭建持续部署环境





准备工作

- 注册heroku
- 下载heroku-cli并安装
- 确认本地已安装node与git, 可通过命令

```
node --version  
npm --version  
git --version
```

查询本地是否已经安装完毕



登录并创建

- 登录使用命令

heroku login

```
PS D:\> heroku login
! heroku-cli: update available from 6.15.5-1f03166 to 6.99.0-ec9edad
Enter your Heroku credentials:
Email: dingd2015@sjtu.edu.cn
Password: *****
Logged in as dingd2015@sjtu.edu.cn
```

如图即登录成功

- 在本地仓库中使用命令

heroku create

```
PS D:\node-js-getting-started> heroku create
! heroku-cli: update available from 6.15.5-1f03166 to 6.99.0-ec9edad
Creating app... done,
https://boiling-shore-82463.herokuapp.com/ | https://git.heroku.com/boiling-shore-82463.git
```

如图即创建了一个heroku repo



上传及部署

- 使用命令

`git push heroku master`

```
PS D:\node-js-getting-started> git push heroku master
Counting objects: 493, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (369/369), done.
Writing objects: 100% (493/493), 230.71 KiB | 0 bytes/s, done.
Total 493 (delta 89), reused 493 (delta 89)
```

上传至repo并进行部署

- 部署完成后，使用命令

`heroku open`

查看部署完成的项目



修改及上传

- 修改代码后，使用命令
`git add .`
将所有修改添加至暂存区

- 使用命令
`git commit`
提交暂存区的修改记录

```
PS D:\node-js-getting-started> git add .
PS D:\node-js-getting-started> git commit -m "Add cool face API"
[master d5f2985] Add cool face API
1 file changed, 1 insertion(+)
```



修改及上传

- commit后，依旧使用命令

`git push`

上传代码并部署

```
PS D:\node-js-getting-started> git push heroku master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 289 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
```

- 上传后，再次使用命令

`git open`

即可打开修改过后的部署的项目



Reference

- [Jenkins工具]之一：持续集成和jenkins入门介绍
[<https://www.2cto.com/kf/201609/544550.html>]
- Jenkins+Node.js持续集成
[<https://www.jianshu.com/p/64b498304d07>]
- Jenkins+Github持续集成
[<https://www.jianshu.com/p/b2ed4d23a3a9>]
- Jenkins基础入门-7-创建一个Project的基本过程
[<https://blog.csdn.net/u011541946/article/details/78014179>]



Reference

- **docker与CI/CD**
[<https://blog.csdn.net/eugenelee2096/article/details/73332615>]
- **CI/CD持续集成/持续部署 敏捷开发**
[https://blog.csdn.net/qq_32261399/article/details/76651376]
- **Heroku官方文档**
[<https://devcenter.heroku.com/articles/getting-started-with-nodejs#introduction>]



THANKS

感谢您的观看

