

持续集成



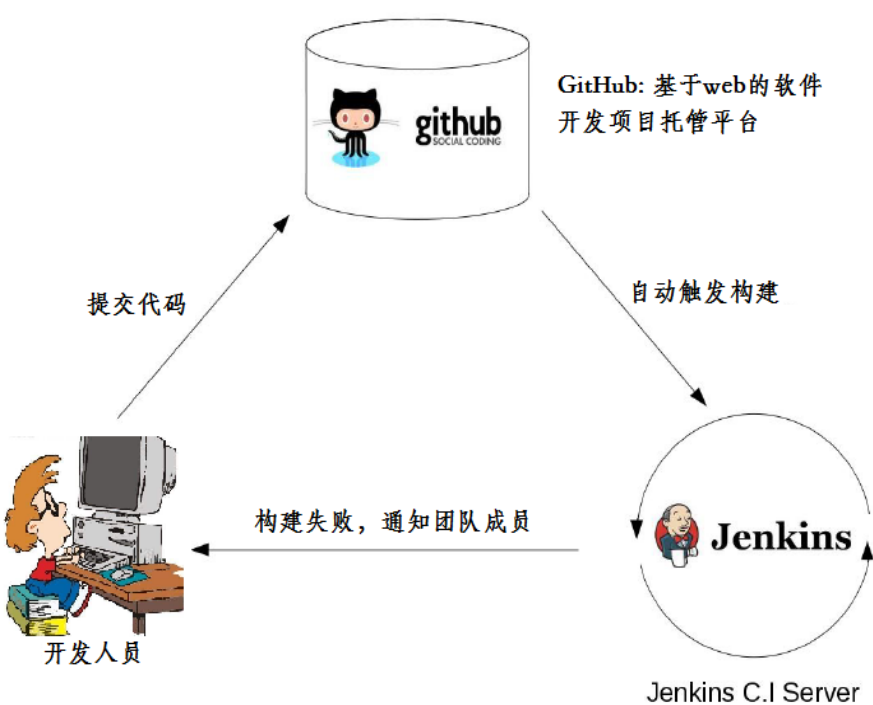
Jenkins



Jenkins

持续集成

- 什么是持续集成?
- 我们为什么需要它?
- 采用持续集成的不同阶段



什么是持续集成?

1. 开发人员定期向共享存储库提交代码。
2. 监视版本控制系统，当检测到提交时，将自动触发构建。
3. 如果构建不是绿色的，将立即通知开发人员

为什么我们需要持续集成？

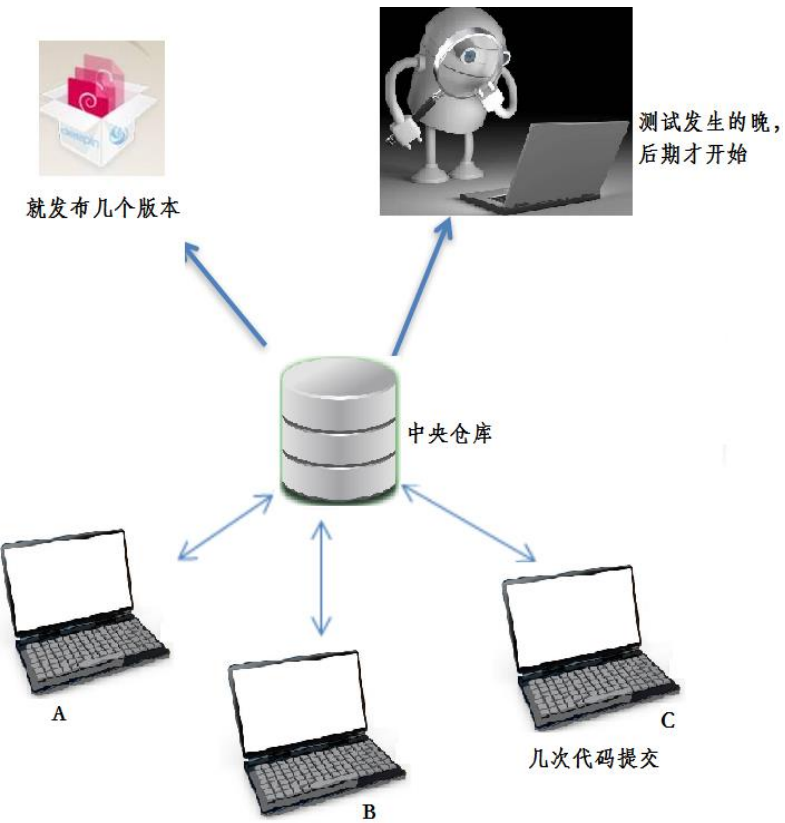
- 尽可能早地在开发生命周期中检测问题或 bug。
- 由于整个代码库是不断集成、构建和测试的，所以潜在的 bug 和错误在生命周期的早期被捕获，从而带来更好的软件质量。



Jenkins

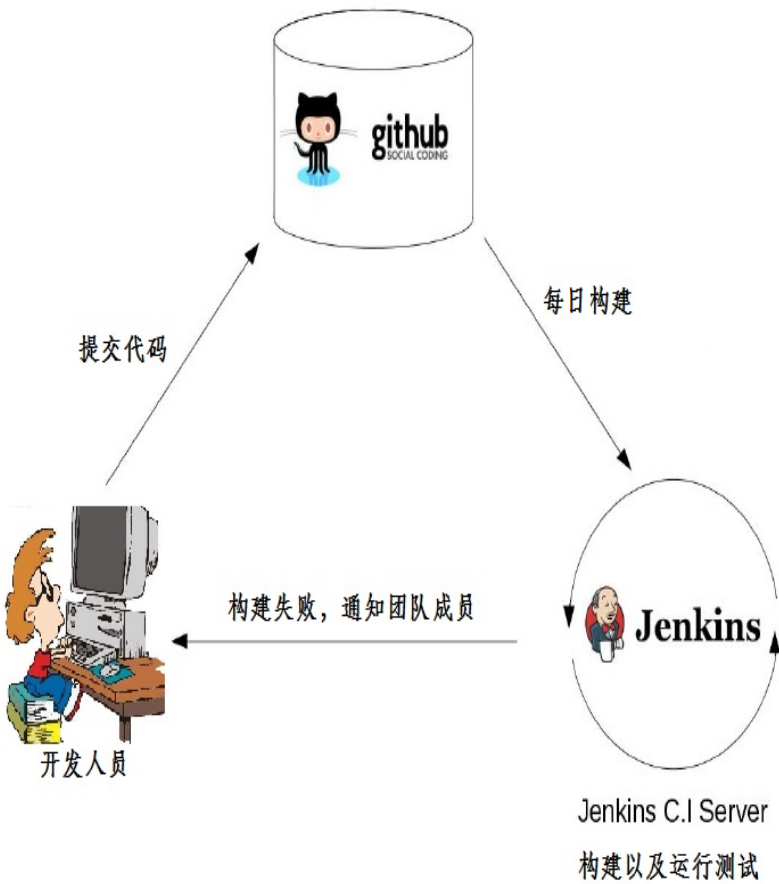
采用持续集成的不同阶段

阶段 1:



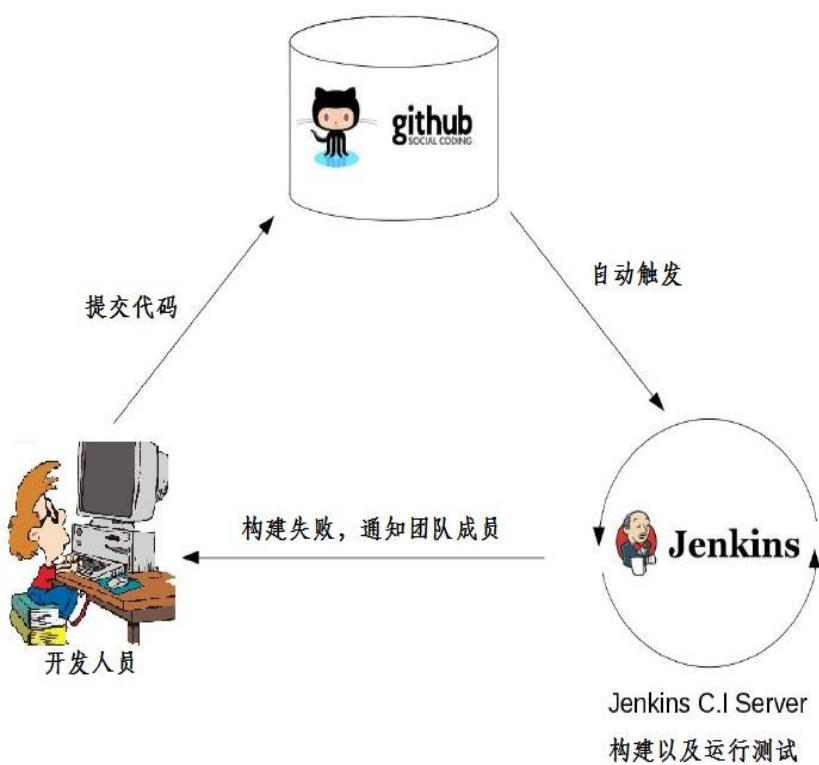
- 没有构建服务器。
- 开发人员不会定期有规律的提交代码。
- 代码改动被手动的集成和测试。
- 就发布几个版本。

阶段 2:



- 自动构建是定期安排的。
- 用构建脚本编译应用程序并运行一组自动化测试。
- 开发人员这个时候定期有规律的提交他们的变更。
- 构建服务器将在构建失败时提醒团队成员。

阶段 3:



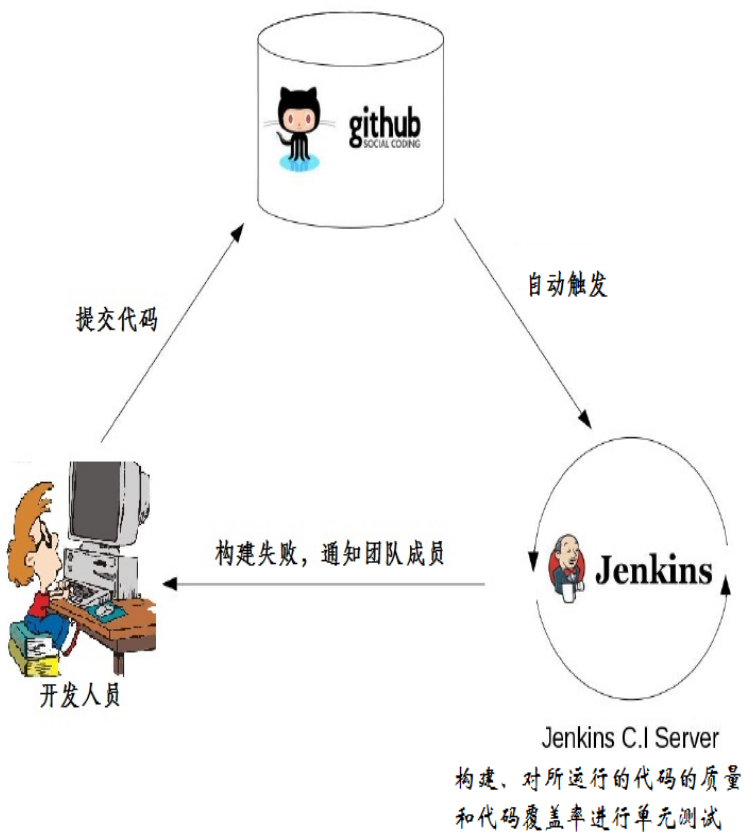
- 每当新代码提交到中央存储库时，就会触发构建。
- 失败的构建通常被视为一个高优先级的问题，并且要很快修复。

阶段 4:

- 自动化的代码质量和代码覆盖率度量,在这个时候与单元测试一起运行,以持续地评估代码质量。

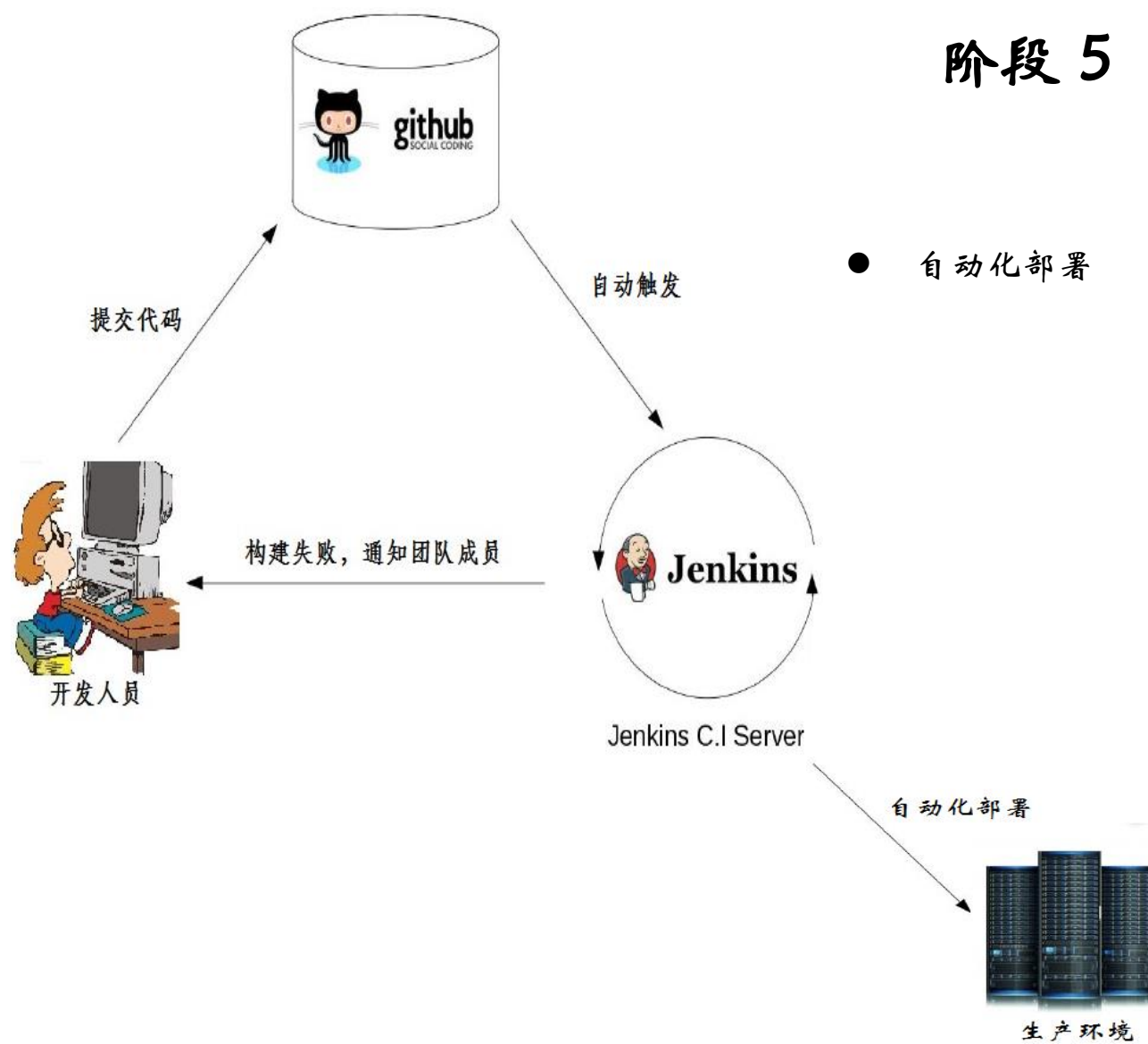
代码覆盖率在增加吗?

我们的构建失败越来越少了么?



阶段 5

- 自动化部署

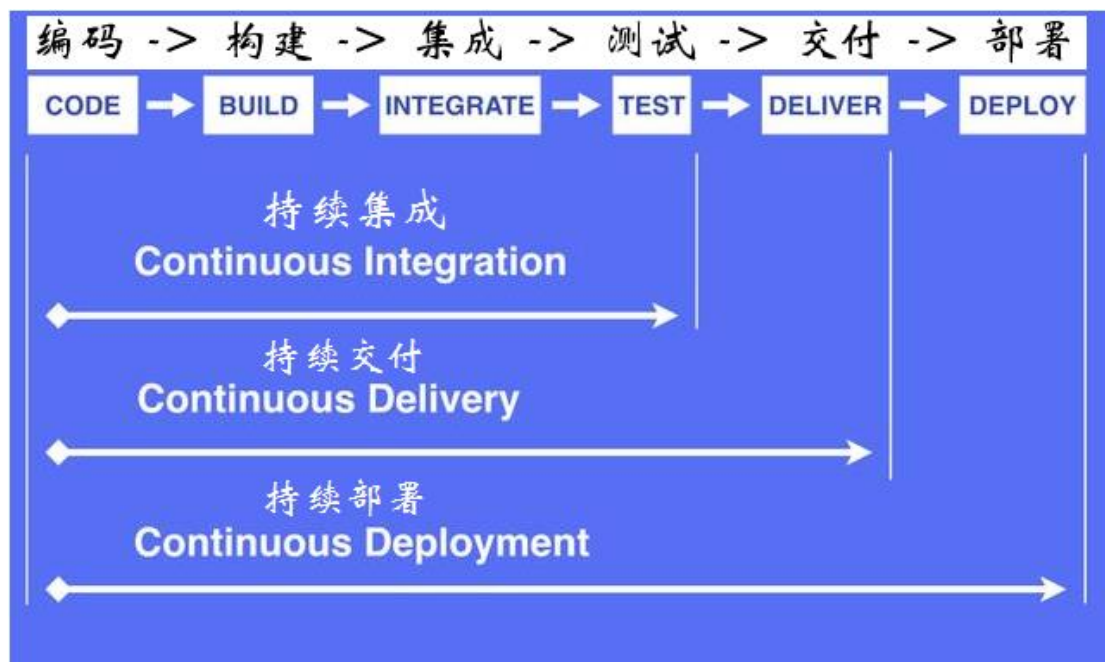




持续集成

持续交付

持续部署



● 持续集成

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。

● 持续交付

持续交付在持续集成的基础上，将集成后的代码交付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。

● 持续部署

持续部署是指当交付的代码通过评审之后，自动部署到生产环境中。持续部署是持续交付的最高阶段。这意味着，所有通过了一系列的自动化测试的改动都将自动部署到生产环境。它也可以被称为“Continuous Release”。

如何实现持续集成？



私有部署



托管型

持续集成也是一种心态

- 对于所有团队成员来说，修复损坏的构建应该被视为一个高度优先级的问题。
- 部署过程应该是自动化的，不需要手动步骤。
- 所有的团队成员都应该专注于为高质量的测试作出贡献



Jenkins 简介以及 Jenkins 的历史

什么是 Jenkins

- Jenkins 是一个持续集成和构建服务器。
- 用于手动、定期或自动构建软件开发项目。
- 它是一个用 Java 编写的开源持续集成工具。
- Jenkins 适合各种规模的团队，和使用各种语言的项目。

为什么 Jenkins 受欢迎

- 易于使用的

- 具有很好的可扩展性

- ✓ 支持不同的版本控制系统
- ✓ 代码质量度量
- ✓ 构建通知
- ✓ 用户界面定制

📁 新建任务

👤 用户

📜 构建历史

⚙️ 系统管理

👁️ 我的视图

📁 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

📝 添加说明

所有	+					
S	W	名称 ↓	上次成功	上次失败	上次持续时间	
🌐	☀️	first-jenkins-job	2 days 16 小时 - #5	无	0.66 秒	🔄
🌐	☀️	maven-project	没有	无	无	🔄

图标: [S](#) [M](#) [L](#)

📖 图例

📡 RSS 全部




















📡 RSS 失败

📡 RSS 最新的构建

Plugins by topic

Source code management

Jenkins has native support for Subversion and CVS as well as the following plugins:

 AccuRev Plugin	— This plugin allows you to use AccuRev as a SCM.
 Anchore Container Image Scanner Plugin	— Allows users to add a build step to run the Anchore container image scanner.
 archive-files-scm-plugin	— ArchiveFilesSCM - This plugin for Jenkins checkouts archive files and extracts to Jenkins job workspace
 AWS CodePipeline Plugin	— AWS CodePipeline is a continuous delivery service for fast and reliable application updates.
 Bazaar Plugin	— This plugin integrates Bazaar version control system to Jenkins. The plugin requires the Bazaar binary (bzr) to be installed on the target machine.
 Bitbucket Branch Source Plugin	— Multibranch projects and Team/Project folders from Bitbucket Cloud and Server . Please note that this plugin requires a server running BitBucket 4.0 or later; Stash 3.x and earlier are not supported.
 BitKeeper Plugin	— Add BitKeeper support to Jenkins
 BlameSubversion	— This plug-in provides utilities for getting svn info from upstream job to downstream job
 ClearCase Plugin	— Integrates Jenkins with ClearCase .
 ClearCase UCM Baseline Plugin	— Allows using ClearCase UCM baselines as the input of builds: When using this SCM, users will be asked at build-time to select the baseline on which the job has to work.
 ClearCase UCM Plugin	— A Pragmatic integration to ClearCase UCM, simplifying continuous integration with Jenkins.
 Clone Workspace SCM Plugin	— This plugin makes it possible to archive the workspace from builds of one project and reuse them as the SCM source for another project.
 CMVC Plugin	— This plugin integrates CMVC to Hudson.
 Compuware Source Code Download for Endeavor, PDS, and ISPW Plugin	— The Compuware Source Code Download for Endeavor, PDS, and ISPW plugin allows Jenkins users to download Endeavor, PDS, or ISPW members from the mainframe to the PC.
 Config Rotator Plugin	
 CVS Plugin	— This bundled plugin integrates Jenkins with CVS version control system.
 Darcs Plugin	— This plugin integrates Darcs version control system to Jenkins. The plugin requires the Darcs binary (darcs) to be installed on the target machine.
 Dimensions Plugin	— This plugin integrates the Serena Dimensions CM SCM with Jenkins.
 File System SCM	— Use File System as SCM.

Jenkins 的历史



Hudson



Jenkins

- Hudson 是由 Kohsuke Kawaguchi 于 2004 年在 Sun 创立的一个业余项目。
- 在 2005 年首次发布。
- Kohsuke 于 2008 年初全职工作在 Hudson 项目上。
- 2010 年成为领先的持续集成解决方案，市场占有率超过 70%。
- 2011 年更名为 Jenkins。



在本地机器安装 Java

Java 版本号要求

- Jenkins 需要至少安装 Java 7.0 及以上版本
- 推荐安装 Java 8.0
- 注:因为一些 Jenkins 插件不支持的问题, 目前 Java 9 不适用于本课程

JAVE_HOME 环境变量

- JAVA_HOME 环境变量指向 Java 开发工具包的安装路径。
- 其他使用 Java 的应用程序也需要 JAVE_HOME。
- 如果您使用的是 Mac，可以按照这节课来配置 JAVA_HOME。
- 如果你使用的是 Windows 或 Linux，你可以跳过这节课，按照下一节课的指导学习。



Jenkins

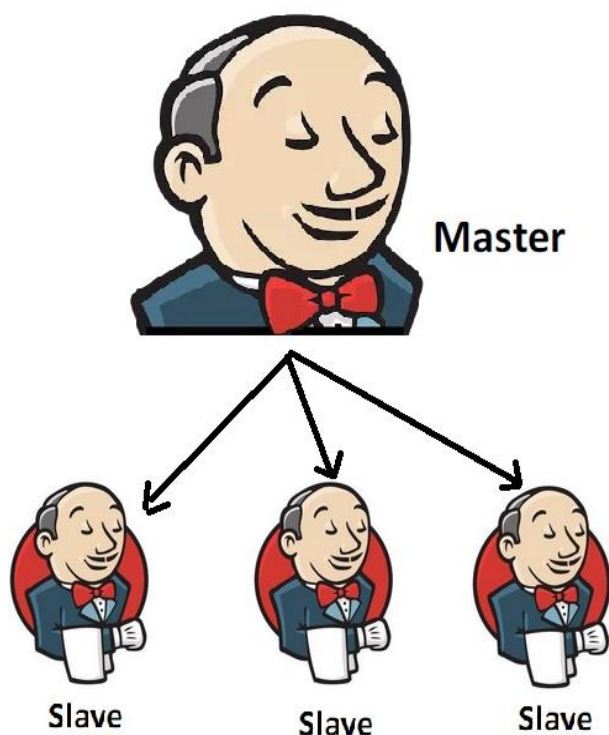
在本地机器上安装 Jenkins



Jenkins

- Jenkins 的 Master 和 Slave 架构
- 一些重要的 Jenkins 术语

Jenkins 的 Master 和 Slave 架构



Master:

- 计划构建 Jobs。
- 把构建分配给子节点运行，来执行实际的 job。
- 监控 slave 并记录构建结果。
- 还可以直接执行构建 job。

Slave:

- 执行由 master 分配的构建任务

Job / Project

- 这两个术语可以互换使用。它们都是指由 Jenkins 控制/监控的可运行的任务。

Slave / Node

- Slaves 是配置用来为 Master 计算机构建项目的计算机。
- Jenkins 在 slaves 上运行一个单独的程序，叫做“slave agent”。
- 当 slaves 注册到 master 上之后，master 开始向 slaves 分配负载。
- node 指的是 Jenkins grid 里的的所有机器，包括 slaves 和 master。

Executor

- Executor 是一个独立的构建流，并行运行在一个节点上。
- 一个节点可以有一个或多个 Executor。

Build

- 一个 Build 是其中一个 Project/Job 的结果

Plugin

- 插件，就像其他系统上的插件一样，是一个扩展核心 Jenkins 服务核心功能的软件。



Jenkins

Jenkins UI 概述



创建第一个 Jenkins Job



运行第一个 Jenkins Job



安裝 GIT 和 GitHub 插件



Jenkins

安裝和配置 Maven

Maven 是做什么的？

- Maven 描述软件是如何构建的。
- Maven 描述项目的依赖。

Java 构建工具



maven



将这个新文件夹的 bin 目录添加到 Path 环境变量中，以便操作系统知道在哪里可以找到 maven 可执行文件。



为我们基于 maven 的项目配置 Jenkins



创建我们的第一个基于 maven 的 Jenkins 项目



运行我们的第一个基于 maven 的 Jenkins 项目

Maven pom.xml 文件

- 描述正在构建的软件项目，包括：
 - ✓ 其他外部模块的依赖。
 - ✓ 目录结构。
 - ✓ 所需的插件。
 - ✓ 为执行某些任务指定预定义的目标，例如编译和打包。

Maven 构建生命周期中的不同阶段

验证 (**validate**)：验证项目是正确的，所有必要的信息都是可用的。

编译 (**compile**)：编译项目的源代码。

测试 (**test**)：使用合适的单元测试框架测试已编译的源代码。

打包 (**package**)：将编译后的代码打包成可分发的格式。

检查 (**verify**)：运行任何检查，以验证包是否有效和满足质量标准。

安装 (**install**)：将包安装到本地存储库中，以便在本地其他项目中作为依赖项使用。

部署 (**deploy**)：将最终包复制到远程存储库，以便与其他开发人员和项目共享。

Maven 构建阶段

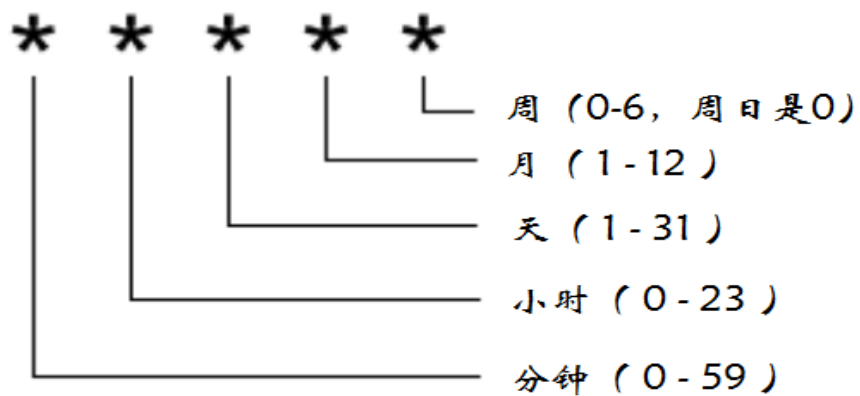
- 这些生命周期阶段按顺序执行，以完成默认生命周期。
- 我们要指定 maven 的 package 命令，这个命令会执行他前面的每个生命周期阶段，包括在执行 package 命令之前的 validate、compile、test 这些命令。
- 我们只需要调用要执行的最后一个构建阶段。



配置 Jenkins 来定期的检查源代码更新

Cron 语法

在 Cron 中，每行由 5 个字段组成，用制表符或空格隔开。



Cron 语法

为一个字段指定多个值:

- * 所有有效的值
- X-Y 一个范围的值
- X,Y,Z 列举了多个值

Cron 语法

举例：

00***

每天的午夜

02-4***

每天早上 2 点，3 点和 4 点

为 Github 帐户设置 SSH 密钥

- SSH 密钥是一种无需密码就能识别受信任计算机的方法
- 生成一个 SSH 密钥对，并将私有 SSH 密钥保存在本地计算机中，将公共密钥添加到 Github 帐户中
- 然后你可以直接将更改推到 Github 存储库，而无需输入密码

如何检查本地计算机中是否有 SSH 公钥文件

SSH 公钥文件通常位于 `~/.ssh` 目录下，以 `.pub` 扩展名结束



Jenkins 的其他构建触发器

什么时候使用计划构建(定时构建选项)?

- 长时间的运行构建 jobs，快速的反馈不那么重要。
- 可能需要运行几个小时的一些集中的加载和性能测试



Jenkins 代码质量度量报告

About

Checkstyle

[Release Notes](#)
[Consulting](#)
[Sponsoring](#)

Documentation

- ▼ Configuration
 - Property Types
 - Filters
 - File Filters
- ▼ Running
 - Ant Task
 - Command Line
- ▼ Checks
 - Annotations
 - Block Checks
 - Class Design
 - Coding
 - Headers
 - Imports
 - Javadoc Comments
 - Metrics
 - Miscellaneous
 - Modifiers
 - Naming Conventions
 - Regexp
 - Size Violations
 - Whitespace
- ▼ Style Configurations
 - Google's Style
 - Sun's Style
- Developers
 - ▼ Extending Checkstyle
 - Writing Checks
 - Writing Javadoc Checks
 - Writing Filters
 - Writing File Filters
 - Writing Listeners
 - Contributing
 - ▼ Beginning Development
 - Eclipse IDE
 - NetBeans IDE
 - IntelliJ IDE
- Javadoc
- Project Documentation
 - ▼ Project Information
 - CI Management
 - Dependencies
 - Dependency Information
 - Distribution

Overview

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration files are supplied supporting the [Sun Code Conventions](#), [Google Java Style](#).

A good example of a report that can be produced using Checkstyle and [Maven](#) can be [seen here](#).

Checkstyle是一种代码静态分析工具，帮助程序员编写符合编码标准的java代码，例如：

- 避免多个空行。
- 删除未使用的变量，
- 执行正确的代码缩进
-

Previous Version Documentation

Current site contains documentation only for latest, released version. We only support this latest version. We highly recommend read latest documentation first.

However, you might need documentation for previous releases because you are still using an older version and latest version describe some features that does not work for you.

You can find documentation for most old versions using a URL format like <http://checkstyle.sourceforge.net/version/X.X> where "X.X" is the version number

PMD Plugin

Created by Ulli Hafner, last modified by Daniel Beck less than a minute ago

[View PMD Plug-in on the plugin site for more information.](#)

ⓘ Older versions of this plugin may not be safe to use. Please review the following:

- XML External Entity (XXE) processing vulnerability

This plugin generates the trend report for PMD, an open source static analysis p

⚠ Installation Requirements

This plug-in requires the utility plug-in "analysis-core" (called "Static Analysis Ut this plug-in is also installed.

Description



PMD: [719 warnings](#) in 3 PMD files.

💡 This plug-in is supported by the Static Analysis Collector plug-in that collects differ Additionally, health reporting and build stability is also based on the aggregated result

The PMD plug-in scans for pmd.xml files in the build workspace and reports the numt analysis plug-ins that are documented on a [separate WIKI page](#).

The following features are provided by this plug-in:

- Configuration of the files to scan after a build.
- Build summary showing the new and fixed warnings of a build
- Several trend reports showing the number of warnings per build
- Overview of the found warnings per module, package, category, or type
 - Parsing of Maven pom.xml or Ant build.xml configuration files to obtain ti
 - Parsing of Java files to obtain the package or name space name

🏠 PMD Source Code Analyzer

Documentation Bugs About News Downloads Support



An extensible cross-language static code analyzer.

 GITHUB

Latest Version: 6.6.0 (29th July 2018)

[Release Notes](#) | [Download](#) | [Documentation](#)

FindBugs Plugin

Created by Ulli Hafner, last modified by Dani

[View FindBugs Plug-in on the plu](#)

📄 Older versions of this plugin ma

- XML External Entity (XXE

This plugin generates the trend report

⚠️ Installation Requirements
This plug-in requires the utility p
latest version of this plug-in is al

Description



FindBugs: [1 warning](#) in 14 Fin

💡 This plug-in is supported by the St:
trend graphs. Additionally, health repo

The FindBugs plug-in scans for findbu
suite of static code analysis plug-ins tr

The following features are provided by

- Configuration of the files to scan
- Build summary showing the new
- Several trend reports showing t
- Overview of the found warnings
 - Parsing of Maven pom.xml
 - Parsing of Java or C# file



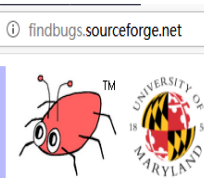
Docs and Info

- FindBugs 2.0
- Demo and data
- Users and supporters
- FindBugs blog
- Fact sheet
- Manual
- Manual(ja/日本語)
- FAQ
- Bug descriptions
- Bug descriptions(ja/日本語)
- Bug descriptions(fr)
- Mailing lists
- Documents and Publications
- Links

Downloads

FindBugs Swag

Development



FindBugs™ - Find Bugs in Java Programs

This is the web page for FindBugs, a program which uses static analysis to look for bugs in Java code. It is free software, distributed under the terms of the [Lesser GNU Public License](#). The name FindBugs™ and the [FindBugs logo](#) are trademarked by [The University of Maryland](#). FindBugs has been downloaded more than a million times.

The current version of FindBugs is 3.0.1.

FindBugs requires JRE (or JDK) 1.7.0 or later to run. However, it can analyze programs compiled for any version of Java, from 1.0 to 1.8.

The current version of FindBugs is 3.0.1, released on 13:05:33 EST, 06 March, 2015. [We are very interested in getting feedback on how to improve FindBugs](#). File bug reports on [our sourceforge bug tracker](#)

[Changes](#) | [Talks](#) | [Papers](#) | [Sponsors](#) | [Support](#)

FindBugs 3.0.1 Release

- A number of changes described in the [changes document](#), including new bug patterns:
 - [BSHIFT_WRONG_ADD_PRIORITY](#)
 - [CO_COMPARETO_INCORRECT_FLOATING](#)
 - [DC_PARTIALLY_CONSTRUCTED](#)
 - [DM_BOXED_PRIMITIVE_FOR_COMPARE](#)



Jenkins

Jenkins 支持其他构建系统
(Ant、Gradle 和 shell 脚本)

Apache Ant

- 使用广泛并且非常著名的 Java 构建脚本语言。
- 灵活、可扩展、相对低级的脚本语言。
- Ant 构建脚本由许多目标组成，每个目标在构建过程中执行特定的任务。



Gradle

- Gradle 是一个相对较新的 Java 虚拟机开源构建工具。
- Gradle 的构建脚本是由基于 Groovy 的领域特定语言编写的。
- Groovy 脚本的简洁性可以让你用很少的代码编写非常有表现力的构建脚本。

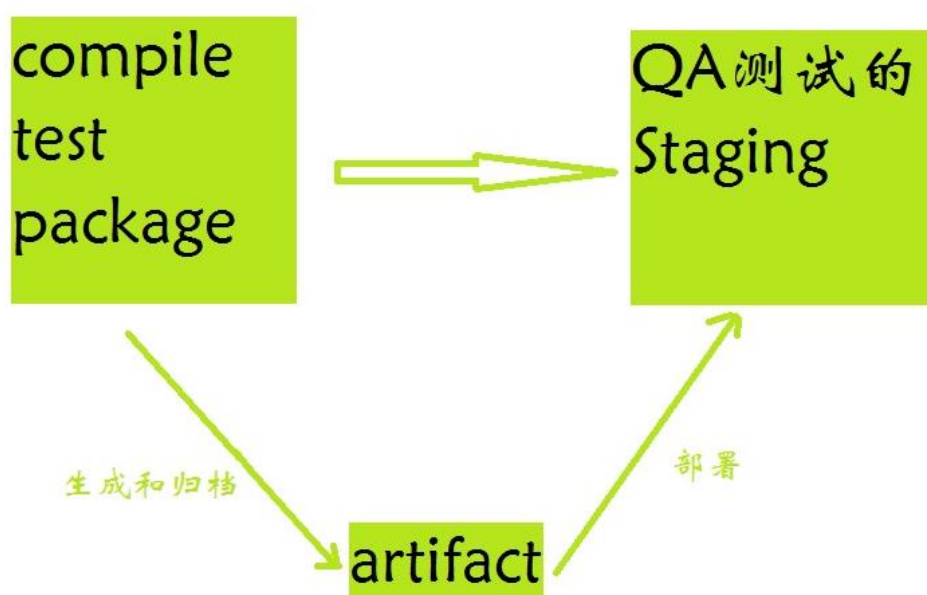




Jenkins

归档生成的 artifacts

持续集成 workflow





Jenkins

Staging 环境安装和配置 Tomcat

Tomcat

Tomcat 是一个开源的 web 服务器，它提供了一个“纯 Java” HTTP web 服务器环境，可以在上面运行 Java 代码。



修改 Tomcat 服务器端口

- Jenkins 运行在端口 8080 上。
- Tomcat 的默认端口也是 8080。



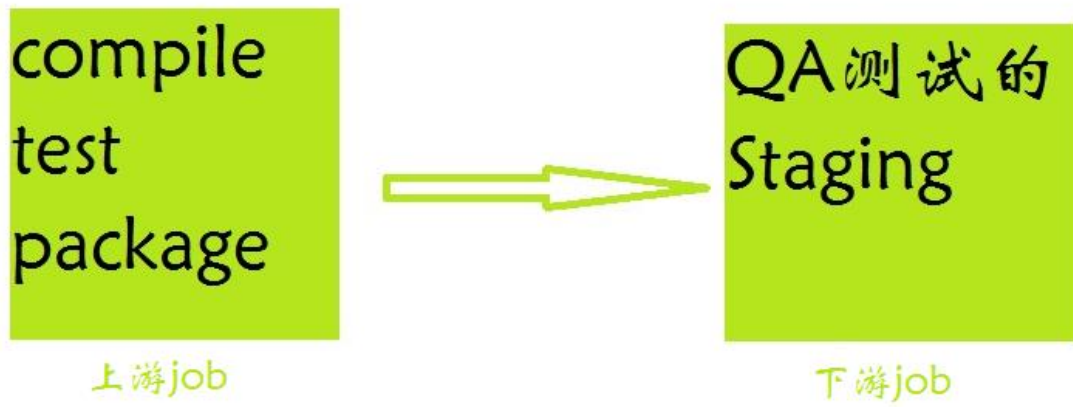
Jenkins

- 安装 “copy artifact” 和 “deploy to container” 插件
- 将应用程序部署到 staging 环境



Jenkins 构建管道

当前所用构建管道



当 build jobs 的数量越来越多时...

All	MyProject Build Pipeline	MyProject Delivery Pipeline	MyProject Jobs	Sample	+
S	W	Name ↓	Last St		
		MyProject » 1 - Developer Jobs » Basic Build and Package	21 sec		
		MyProject » 1 - Developer Jobs » Deploy to Android Func Test Env	52 sec		
		MyProject » 1 - Developer Jobs » Deploy to iOS Func Test Env	41 sec		
		MyProject » 1 - Developer Jobs » Static Code Quality Analysis	1 min 2		
		MyProject » 1 - Developer Jobs » Trigger Deploy to Func Test Envs	1.4 sec		
		MyProject » 2 - QA Jobs » Deploy to Perf Test Env	32 sec		
		MyProject » 2 - QA Jobs » Deploy to Reqr Test Env	31 sec		
		MyProject » 2 - QA Jobs » Func Tests	25 sec		
		MyProject » 2 - QA Jobs » Perf Tests	11 sec		
		MyProject » 2 - QA Jobs » Reqr Tests	6 hr 18		

Build Pipeline 插件

← → ↻ 安全 | https://wiki.jenkins.io/display/JENKINS/Build+Pipeline+Plugin

Spaces People Create ...

Jenkins ☆



Pages

SPACE SHORTCUTS

- Product requirements
- How-to articles
- Retrospectives

CHILD PAGES

- Plugins
 - Build Pipeline Plugin
 - Build Pipeline Plugin - ...
 - Build Pipeline Plugin - ...
- + Create child page

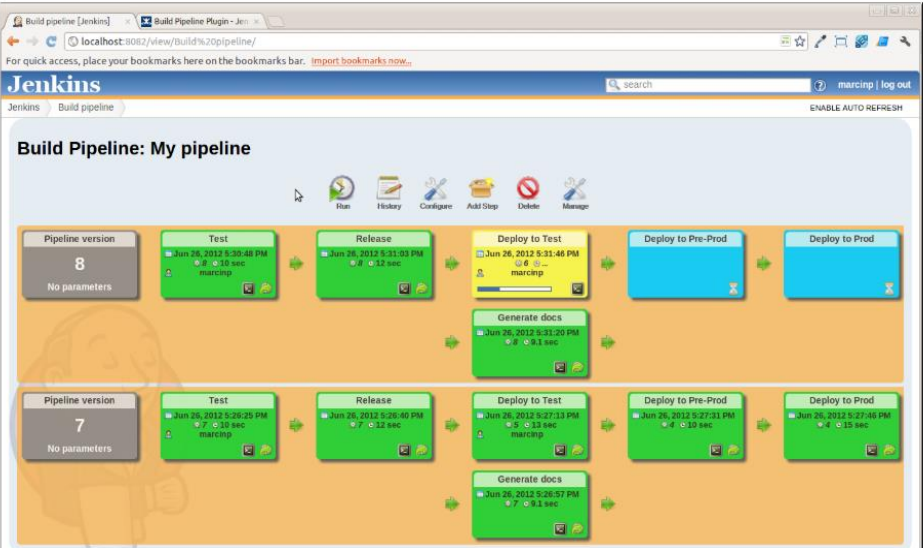
Pages / Home / Plugins  

Build Pipeline Plugin

Created by Geoff Bullen, last modified by Dan Alvizu on Dec 12, 2017

Summary

This plugin provides a *Build Pipeline View* of upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.

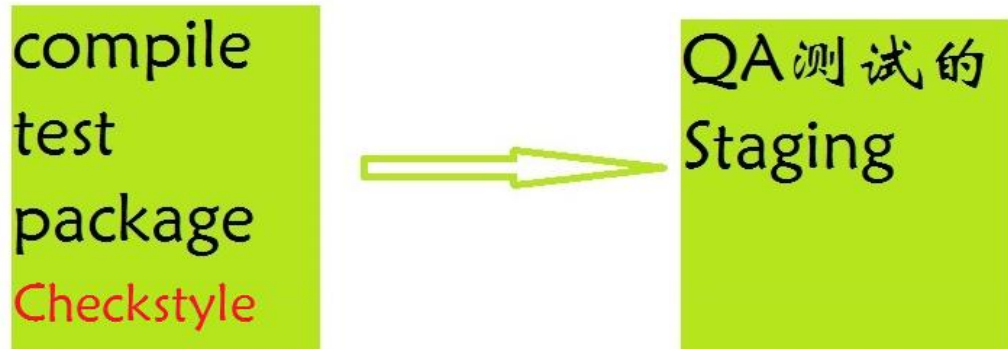


The screenshot shows the Jenkins web interface for the 'Build Pipeline Plugin'. The main view is titled 'Build Pipeline: My pipeline'. It displays two versions of the pipeline, version 8 and version 7. Each version is represented by a series of colored boxes connected by arrows, indicating the flow of the build process. Version 8 includes steps: Pipeline version (8), Test (Jun 26, 2012 5:30:48 PM, 1:30 sec, marcip), Release (Jun 26, 2012 5:31:03 PM, 1:30 sec, marcip), Deploy to Test (Jun 26, 2012 5:31:46 PM, 1:4 sec, marcip), Generate docs (Jun 26, 2012 5:31:30 PM, 1:3 sec, marcip), Deploy to Pre-Prod, and Deploy to Prod. Version 7 includes steps: Pipeline version (7), Test (Jun 26, 2012 5:28:25 PM, 1:30 sec, marcip), Release (Jun 26, 2012 5:28:40 PM, 1:30 sec, marcip), Deploy to Test (Jun 26, 2012 5:27:13 PM, 1:5 sec, marcip), Generate docs (Jun 26, 2012 5:26:57 PM, 1:7 sec, marcip), Deploy to Pre-Prod, and Deploy to Prod. The interface also includes a search bar, a 'log out' button, and a 'ENABLE AUTO REFRESH' toggle.

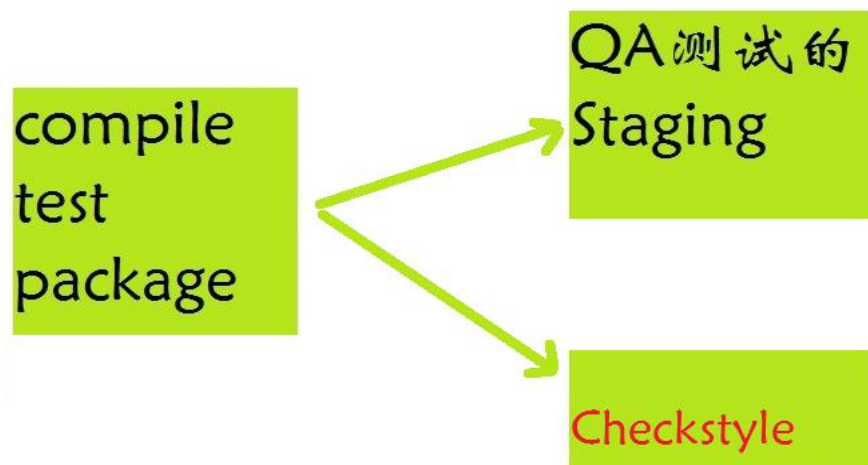


并行进行 Jenkins 构建

线性管道



并行管道



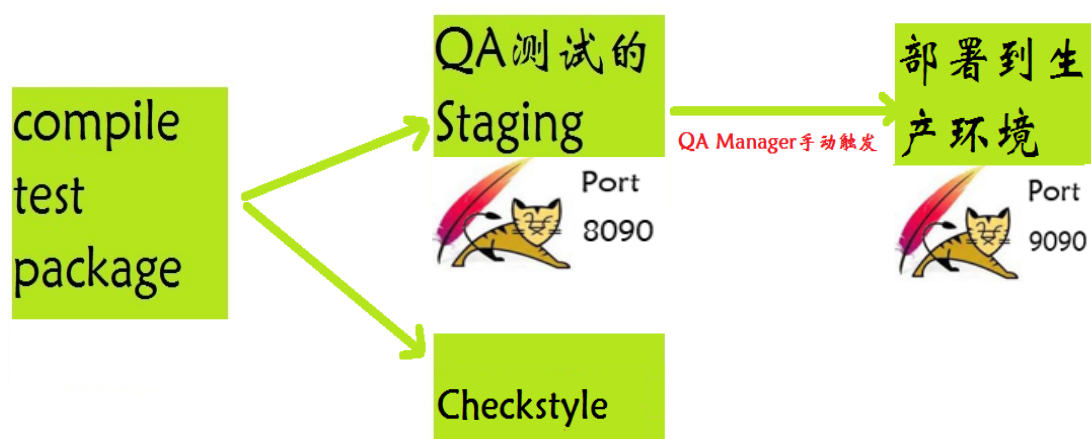


Jenkins

持续交付

将我们的应用程序部署到生产环境中

完整的持续交付 Pipeline





Jenkins

Pipeline as code

基于代码的 Pipeline 的好处

- 版本控制
- 最佳实践
- 执行 jobs 不容易出错
- 基于逻辑的执行步骤

```
1
2 pipeline {
3     agent any
4     stages {
5         stage ('Initialize') {
6             steps {
7                 sh '''
8                     echo "PATH = ${PATH}"
9                     echo "M2_HOME = ${M2_HOME}"
10                '''
11            }
12        }
13        stage ('Build') {
14            steps {
15                echo 'Hello World!'
16            }
17        }
18    }
19 }
```



Jenkins

Pipeline as code

自动化我们现有的 Jenkins Pipeline



Pipeline as code

让我们的自动化更进一步

附加的自动化

- 设置 Git 仓库轮询
- 部署到 tomcat 服务器
- 我们将设置几个并行运行的任务
- 我们将简要介绍如何在 Amazon Web 服务中的 EC2 上设置 tomcat

Pipeline As Code 代码介绍

```
1 pipeline {
2   /*(A Declarative Pipeline)*/
3   agent any
4
5   stages {
6     stage('Build') {
7       steps {
8
9       }
10    }
11    stage('Test') {
12      /*agent section could go here as well*/
13      steps {
14
15      }
16    }
17  }
18 }
```



Visual Studio Code 安裝和使用



创建 pipeline job



Jenkins

Windows 机器安装 Cmdr

Why Cmdr?

- 既可以用 Windows 下的所有命令，有可以用 Linux 命令
- 不需要对现有的 Jenkinsfile 进行修改
- Windows 上可以用 Git 可以用 SSH

在 AWS 准备 Staging 和 Production 环境的步骤

- 创建 AWS 账号，为 Tomcat 服务器配置 AWS 安全组并创建密钥对。
- 在 AWS 上创建 EC2 实例来模拟 Staging 环境。
- 在 AWS 上创建 EC2 实例来模拟 Production 环境。
- 在创建的 AWS 实例上安装并运行 Tomcat。

在 AWS 的 instance 上安装拟 production 环境的 tomcat

- Staging 和 Production 这两个 tomcats 可以使用相同的 8080 端口，因为它们运行在不同的主机上。
- 不需要配置 tomcat 用户和角色，因为我们不再使用 deploy to container plugin 了。



Jenkins

Pipeline as code

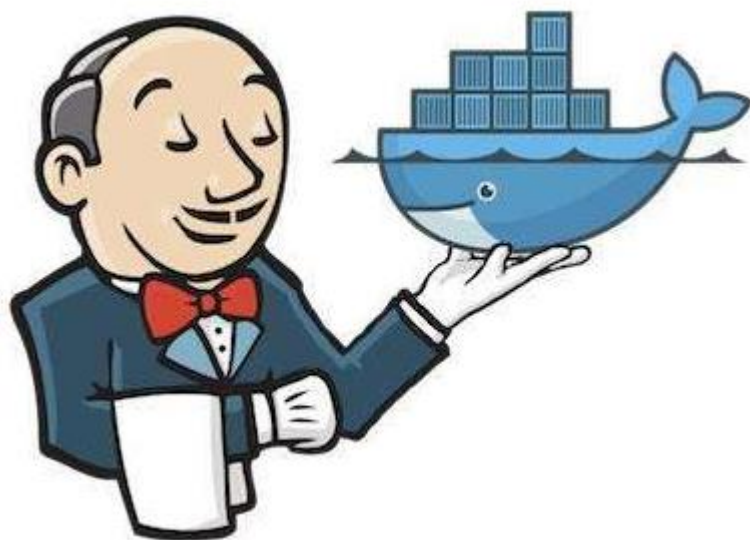
完全自动化我们现有的 Jenkins Pipeline

让我们的自动化更进一步

本地和远程服务器

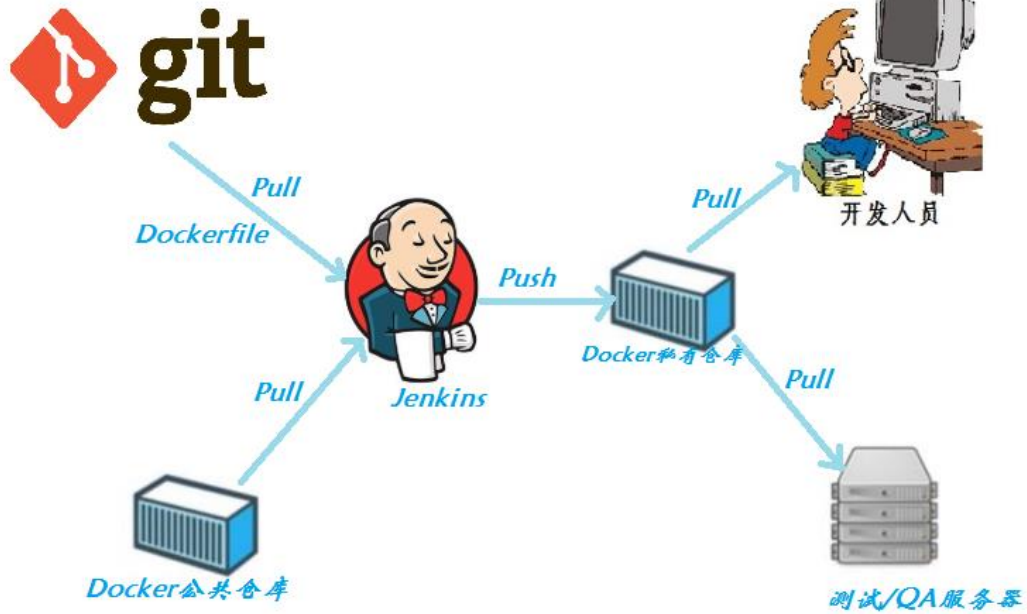
- scp 到远程 tomcat
- cp 到本地 tomcat
- 本地服务器把 ip 地址换成 localhost

Docker 与 Jenkins 集成



Docker 与 Jenkins 集成

Docker构建管道



Docker 与 Jenkins 集成

- 我们将在 Mac 上安装 Docker
- 如果你的电脑符合要求建议下载 Mac-native 的版本，如果不符合要求，用 Docker Toolbox
- 验证安装结果

Docker Hub

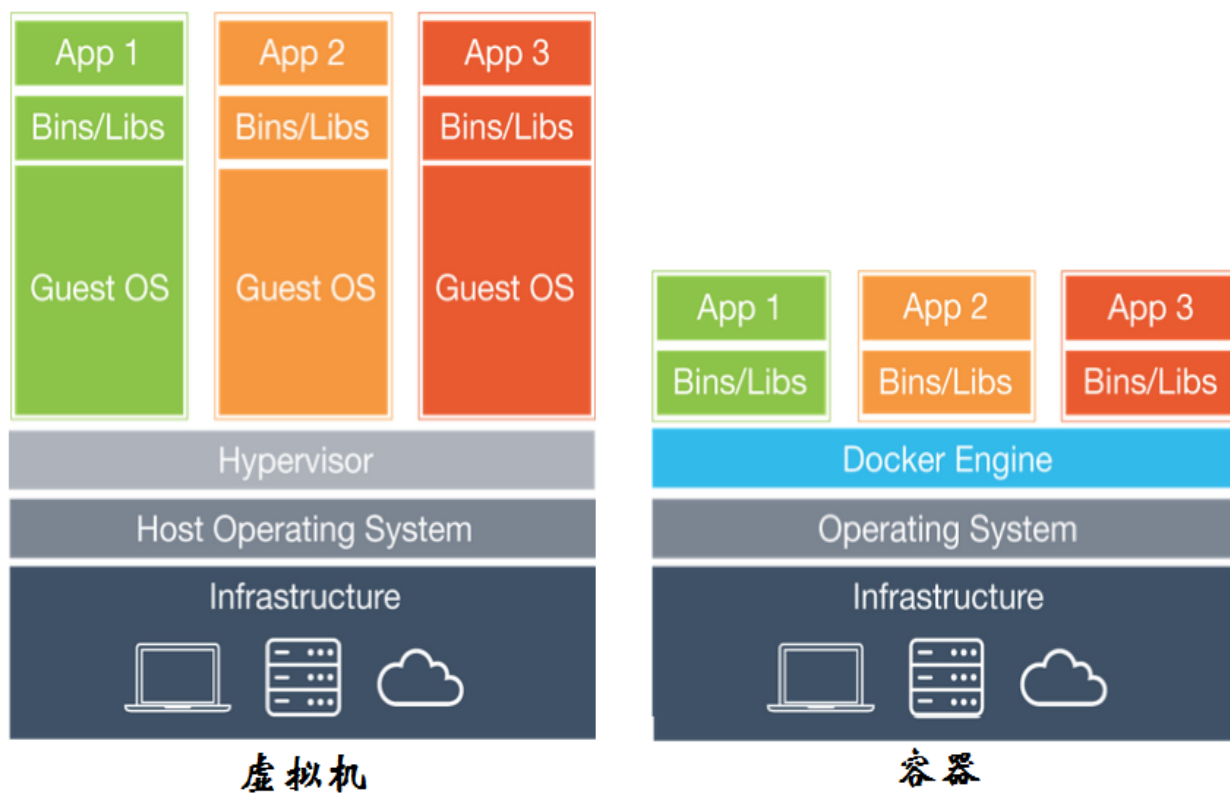


Docker 镜像仓库 (Docker Hub 公共 images 仓库) 介绍

什么是 container?

- Container 是轻量级版本的虚拟机
- 一些底层资源是共享的
- 您可以在主机上安装比 VM 更多的容器

对比 VMs 和 Containers



Docker 与 Jenkins 集成

- 我们将在 Linux 上安装 Docker
- 开启服务用： `service docker start`
- 验证安装结果

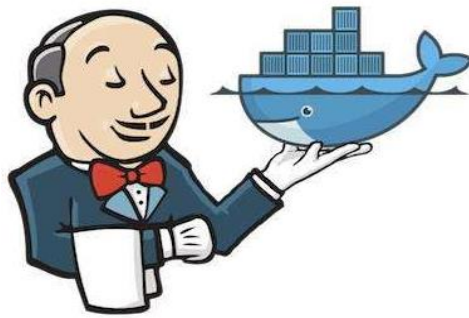
CentOS 安装 docker 说明：

<https://docs.docker.com/install/linux/docker-ce/centos>

Docker 与 Jenkins 集成

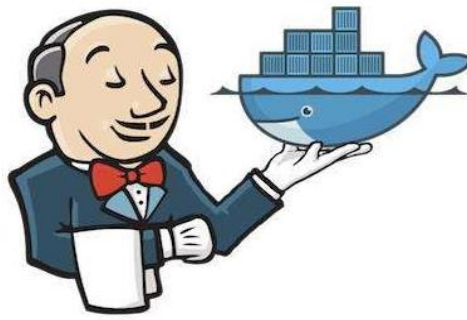
- 运行容器
- 学习管理镜像和容器
- 学习映射端口

Docker 与 Jenkins 集成



配置项目，为 Docker 镜像构建做准备

Docker 与 Jenkins 集成



Dockerfile 概述

Docker 与 Jenkins 集成

Dockerfile 样例：

```
FROM ubuntu:14.04

# Install.
RUN \
apt-get -y upgrade && \
apt-get install -y build-essential && \
ADD root/.bashrc /root/.bashrc
# Define working directory.
WORKDIR /root

# Define default command.
CMD ["bash"]
```

Docker 与 Jenkins 集成

Dockerfile 语法：

FROM image-name:tag

使用指定的镜像作为起点，创建自己的镜像

ADD 主机中的文件 容器里的路径

COPY 主机中的文件 容器里的路径

使用 ADD 和 COPY 把容器装载上所需的文件

Docker 与 Jenkins 集成

Dockerfile 语法：

RUN <some command(s)>

用 RUN 在容器里执行命令

CMD ["command" "parameters"]

使用 CMD 作为入口点命令来启动应用程序，比如

"catalina.sh run"

Docker 与 Jenkins 集成

Dockerfile 语法：

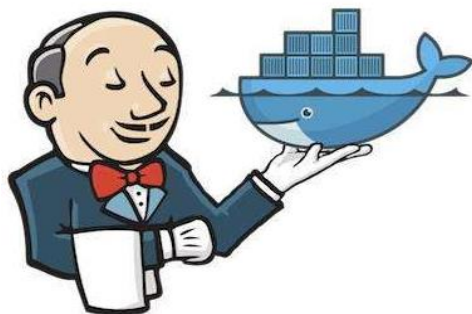
VOLUME 容器里的文件夹

用 VOLUME 来映射容器里的文件夹到主机里的文件夹

WORKDIR 容器里的文件夹

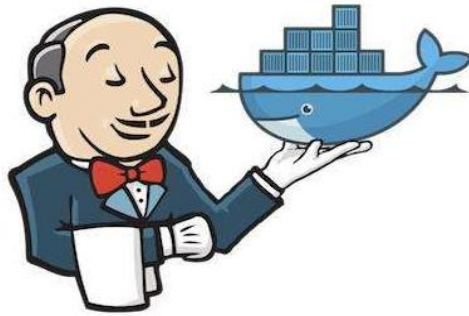
在执行 ADD,COPY,RUN 命令时，使用 WORKDIR 在容器内的文件夹之间切换用

Docker 与 Jenkins 集成



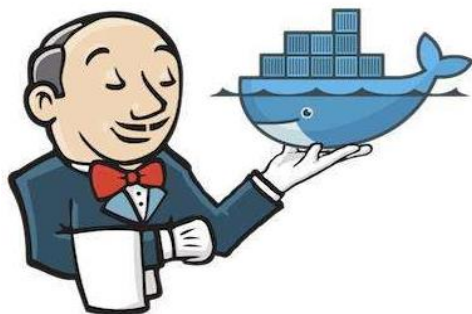
让我们来构建一个

Docker 与 Jenkins 集成



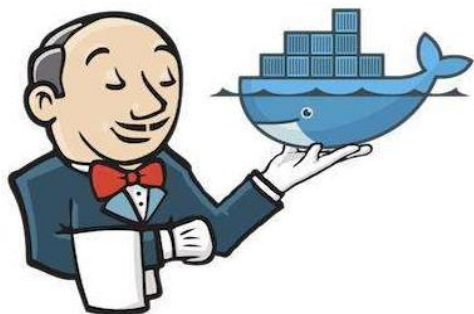
创建 Dockerfile 并且添加到 project

Docker 与 Jenkins 集成



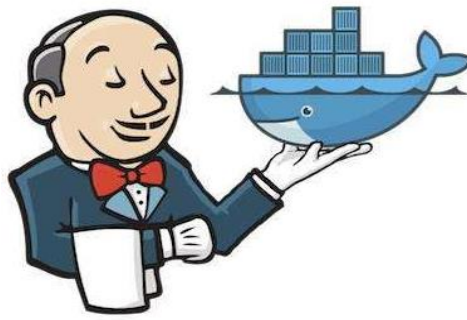
构建和标记 Docker 镜像

Docker 与 Jenkins 集成

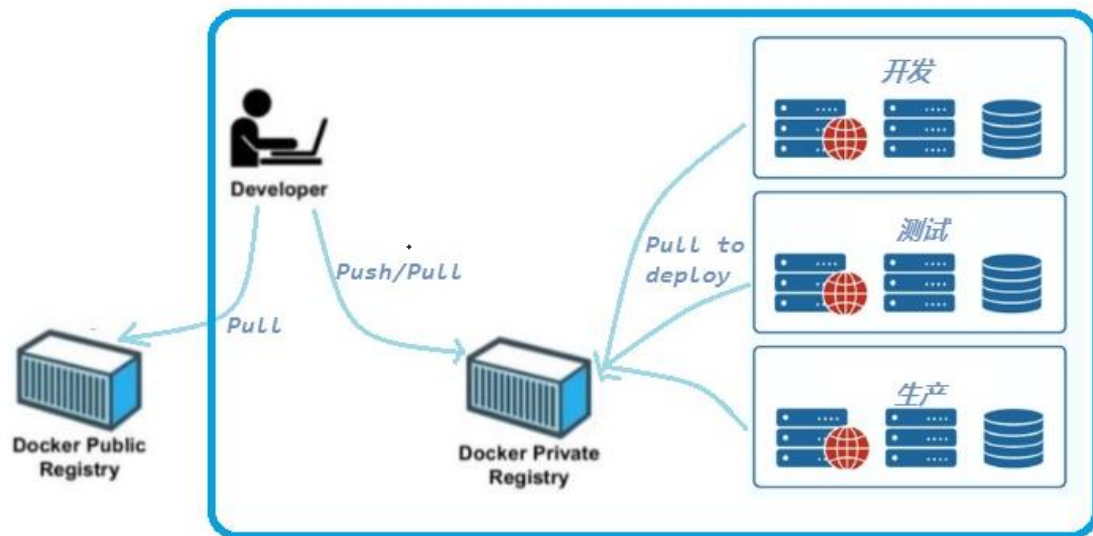


在 Docker 中运行生成的二进制文件

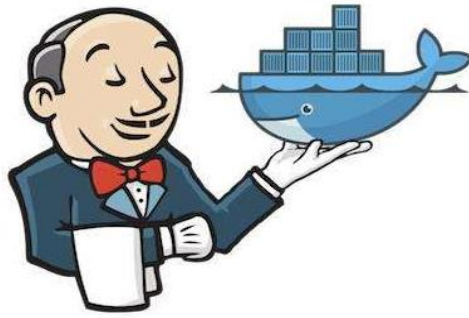
Docker 与 Jenkins 集成



回顾

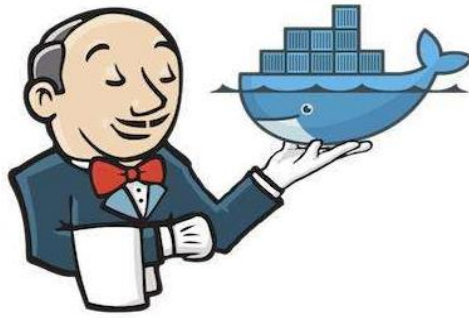


Docker 与 Jenkins 集成



tomcatwebapp:\$(env.BUILD_ID)

Docker 与 Jenkins 集成



继续学习



分布式 Jenkins 构建介绍



AWS EC2 是业界广泛使用的云计算平台和虚拟机提供商。

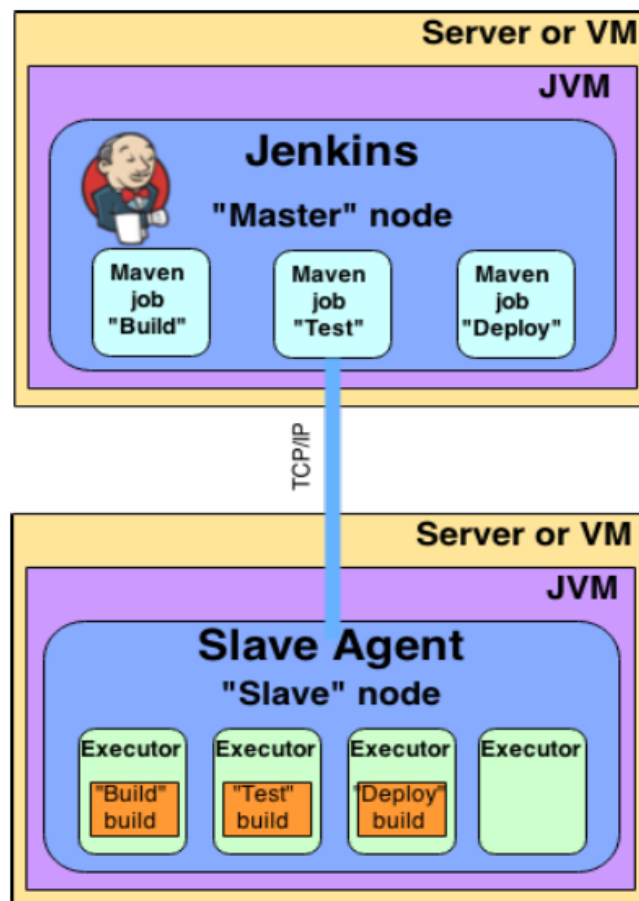
它类似于 Digital Ocean 和 Microsoft Azure。



Jenkins

在云中安装 Jenkins Master

Jenkins Slave Agent



启动 slave agent 的不同方法

- master 可以通过 SSH 启动 slave agent。
- 使用 Java Web Start 手动启动 slave agent。
- 将 slave agent 安装为窗口服务。
- 从 slave 机器上的命令行直接启动 slave agent



在云中安装 Jenkins slave 并形成 Jenkins 集群

启动 Slave agent

- 主节点将通过 SSH 在 slave 机器上启动 slave agent。
- 从 master 节点到 slave 节点设置自动 SSH 免密登录。
- master 节点将用一个名为 Jenkins 的特定用户来启动 slave agent。

Executor

- Jenkins executor 是允许构建在节点上运行的基本构建块之一。
- 可以将 executor 看作单个“进程 ID”，或 Jenkins 在你的机器上运行构建时执行的基本资源单元。
- 这个 executor 的数量基本上指定了 Jenkins 可能在这个代理上执行的并发构建的最大数量。
- executor 数量的一个合适的值是机器上的 CPU 内核数量。
- 设置更高的值会导致每个构建花费更长的时间，但可能会增加总体吞吐量。



Jenkins

在 Jenkins 集群上并发构建

Jenkins 节点加标签

给节点加标签

- 在实际场景中，我们可能希望为某些类型的作业保留一个节点。
- 例如，如果你有运行性能测试的 job，你可能希望它们只在一个特定配置的机器上运行，同时不允许其他 job 使用该机器。
- 为此，你可以通过给测试 job 一个与机器匹配的标签表达式来限制它们可能运行的位置。