互联网中小型企业的持续集成实施路线

互联网研发的世界里唯快不破、迭代速度往往很快。在快速的发展迭代中，如何让项目产品平稳的落地，就需要有完善可靠的持续集成 CICD 和 DevOps 方案。

本场 Chat 首先会带领大家入门互联网持续集成的一些实践和落地，帮助大家了解中小型互联网企业持续集成应该如何落地。本场 Chat 您将学到如下内容：

- 了解持续集成。
- 持续集成的工具、框架、平台。
- 中小型企业如何设计架构 CI 平台。
- 持续集成主要是分享互联网中小型企业持续集成的实践路线 、架构和一些场景的落地。

了解持续集成

集成是指软件个人研发的部分向软件整体部分交付，以便尽早发现个人开发部分的问题；部署是代码尽快向可运行的开发/测试环节交付，以便尽早测试；交付是指研发尽快向客户交付，以便尽早发现生产环境中存在的问题。如果说等到所有东西都完成了才向下个环节交付，导致所有的问题只能在最后才爆发出来，解决成本巨大甚至无法解决。

而所谓的持续，就是说每完成一个完整的部分，就向下个环节交付，发现问题可以马上调整。是的，问题不会放大到其他部分和后面的环节。这种做法的核心思想在于，既然事实上难以做到事先完全了解完整的、正确的需求，那么就干脆一小块一小块的做，并且加快交付的速度和频率，使得交付物尽早在下个环节得到验证。早发现问题早返工。举个例子，你家装修厨房，其中一项是铺地砖，边角地砖要切割大小。如果一次全切割完再铺上去，发现尺寸有误的话浪费和返工时间就大了，不如切一块铺一块。这就是持续集成。装修厨房有很多部分，每个部分都有检测手段，如地砖铺完了要测试漏水与否，线路铺完了要通电测试电路通顺，水管装好了也要测试冷水热水。如果全部装完了再测，出现问题可能会互相影响，比如电路不行可能要把地砖给挖开……。那么每完成一部分就测试，这是持续部署。全部装修完了，你去验收，发现地砖颜色不合意，水池太小，灶台位置不对，返工吗？所以不如没完成一部分，你就去用一下试用验收，这就是持续交付。

持续集成，Continuous integration ，简称 CI。

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

持续集成正是针对这一类问题的一种软件开发实践。它倡导团队开发成员必须经常集成他们的工作，甚至每天都可能发生多次集成。而每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件。

以我经过的项目（假设为 A 项目）为例进行描述。

首先，解释下集成。我们所有项目的代码都是托管在 SVN 服务器上。每个项目都要有若干个单元测试，并有一个所谓集成测试。所谓集成测试就是把所有的单元测试跑一遍以及其它一些能自动完成的测试。只有在本地电脑上通过了集成测试的代码才能上传到 SVN 服务器上，保证上传的代码没有问题。所以，集成指集成测试。

再说持续。不言而喻，就是指长期的对项目代码进行集成测试。既然是长期，那肯定是自动执行的，否则，人工执行则没有保证，而且耗人力。对此，我们有一台服务器，它会定期的从 SVN 中检出代码，并编译，然后跑集成测试。每次集成测试结果都会记录在案。完成这方面工作的就是下面要介绍的 Jenkins 软件。当然，它的功能远不止这些。在我们的项目中，执行这个工作的周期是1天。也就是，服务器每1天都会准时地对 SVN 服务器上的最新代码自动进行一次集成测试。

持续集成的特点有以下几点：

- 它是一个自动化的周期性的集成测试过程，从检出代码、编译构建、运行测试、结果记录、测试统计等都是自动完成的，无需人工干预；
- 需要有专门的集成服务器来执行集成构建；
- 需要有代码托管工具支持。

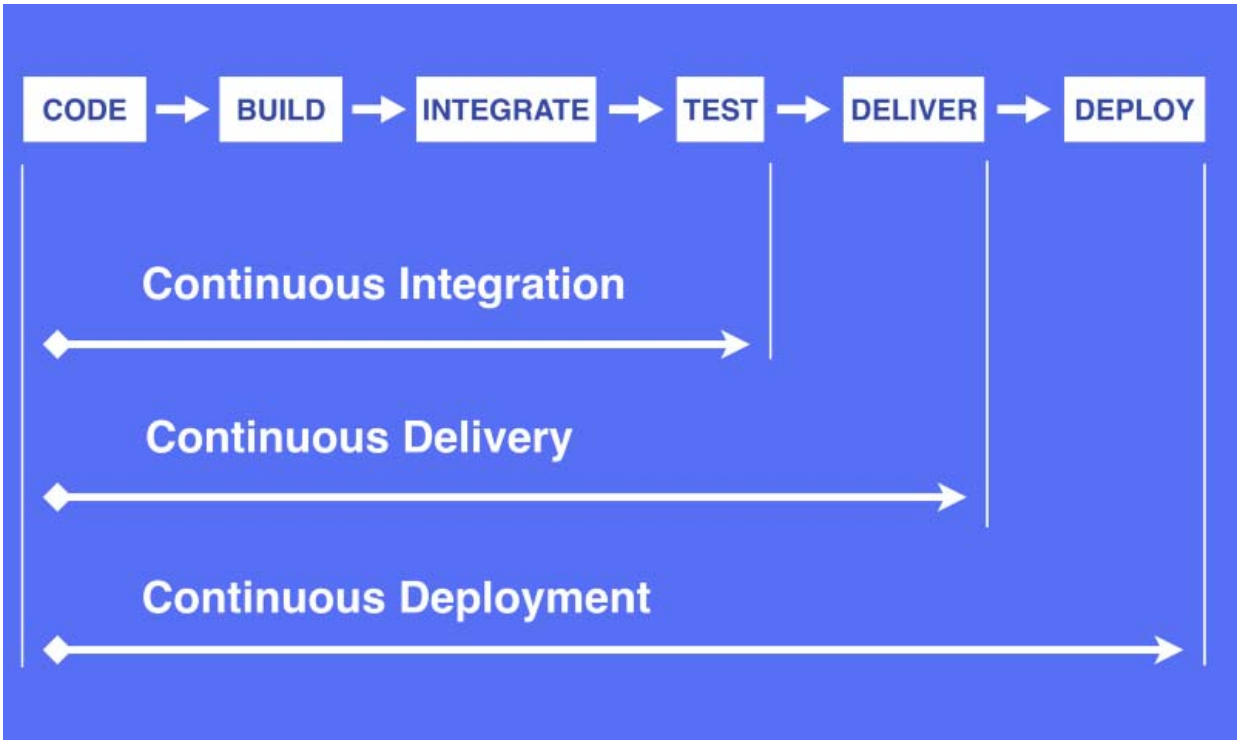
持续集成的作用有几下几项：

- 保证团队开发人员提交代码的质量，减轻了软件发布时的压力；
- 持续集成中的任何一个环节都是自动完成的，无需太多的人工干预，有利于减少重复过程以节省时间、费用和工作量。

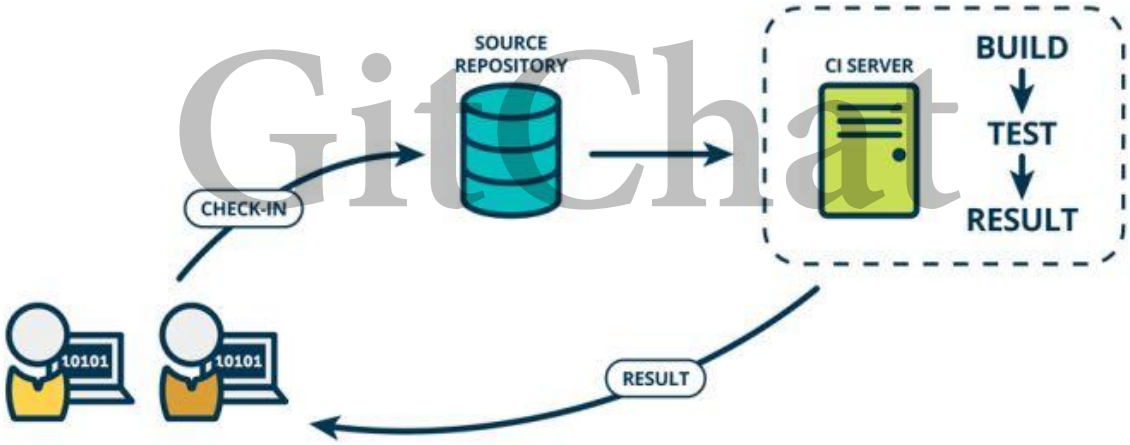
持续集成和部署有以下不同的阶段和流程：

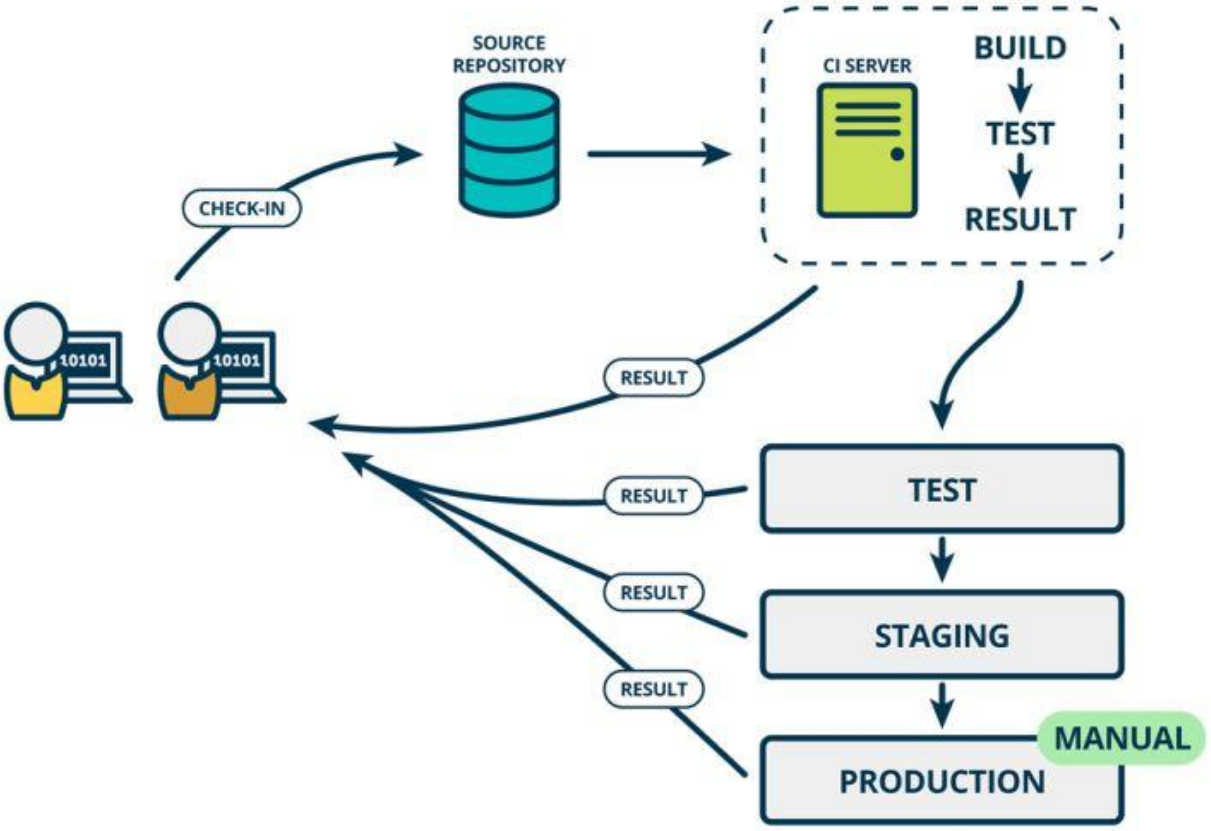
- 持续（Continuous）：不断的获取反馈，响应反馈。
- 集成（Integration）：编译、测试、打包。
- 部署（Deployment）：应用组件或基本设施的代码或配置变更在产品环境生效称为“部署”。
- 发布（Release）：具有业务影响的功能变化对最终用户可见称为“发布”。
- 交付（Delivery）：可以理解为从 Deployment 到 Release 之间的阶段，更多的强调的是一种能力。开发有能力频繁的部署，业务有能力随时发布。

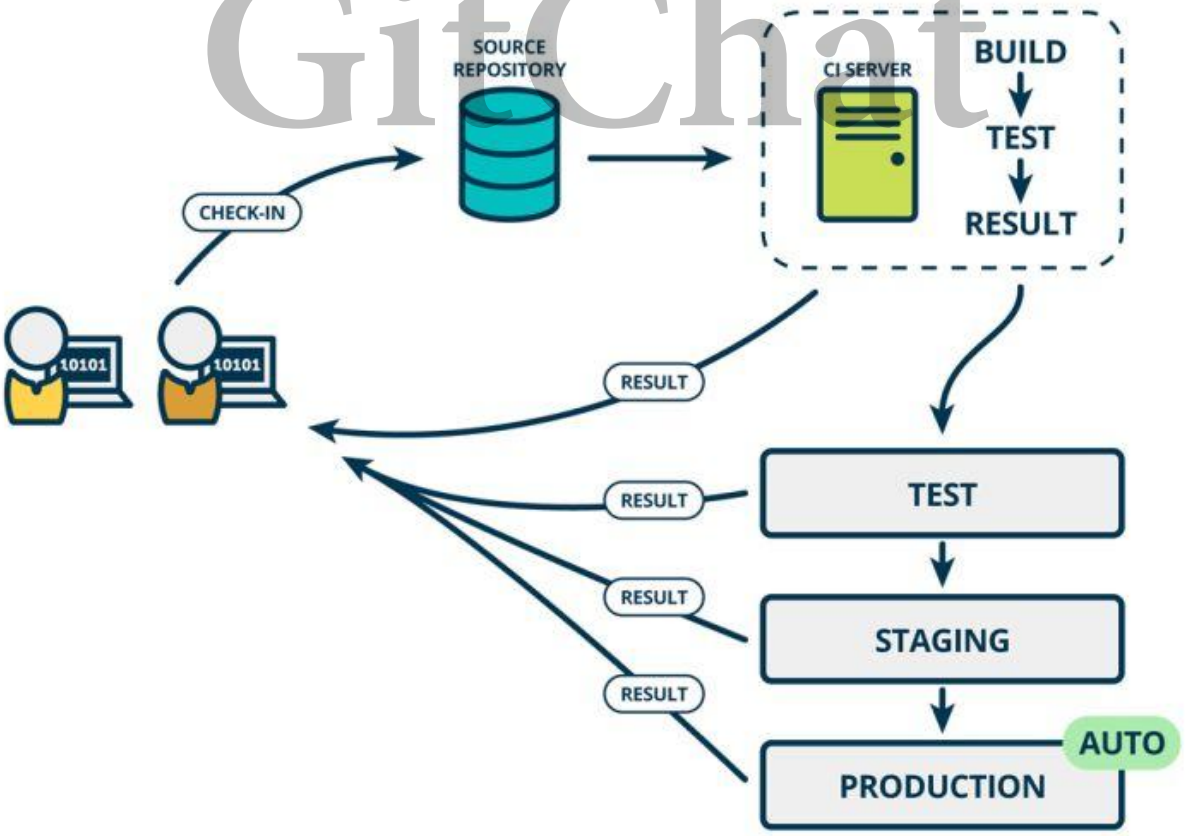
持续集成整体构件的架构流程，如下图所示。



下面三张图分别展示了持续集成构件的流程：







持续集成的工具、框架、平台

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

持续集成正是针对这一类问题的一种软件开发实践。它倡导团队开发成员必须经常集成他们的工作，甚至每天都可能发生多次集成。而每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件。常用的持续集成工具平台有 Travis CI、Jenkins、GoCD、 Teamcity、Bamboo 等，其中最出名和市场占有率最高的是 Jenkins。

Jenkins ，原名 Hudson，2011年改为现在的名字，它是一个开源的实现持续集成的软件工具。点击这里，访问官方网站。

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。

Jenkins 的特点有：

- 易安装：仅仅一个 java -jar jenkins.war，从官网下载该文件后，直接运行，无需额外的安装，更无需安装数据库；
- 易配置：提供友好的 GUI 配置界面；
- 变更支持：Jenkins 能从代码仓库（Subversion/CVS）中获取并产生代码更新列表并输出到编译输出信息中；
- 支持永久链接：用户是通过 Web 来访问 Jenkins 的，而这些 Web 页面的链接地址都是永久链接地址，因此，你可以在各种文档中直接使用该链接；
- 集成 E-Mail/RSS/IM：当完成一次集成时，可通过这些工具实时告诉你集成结果（据我所知，构建一次集成需要花费一定时间，有了这个功能，你就可以在等待结果过程中，干别的事情）；
- JUnit/TestNG 测试报告：也就是以图表等形式提供详细的测试报表功能；
- 支持分布式构建：Jenkins 可以把集成构建等工作分发到多台计算机中完成；
- 文件指纹信息：Jenkins 会保存哪次集成构建产生了哪些 jars 文件，哪一次集成构建使用了哪个版本的 jars 文件等构建记录；
- 支持第三方插件：使得 Jenkins 变得越来越强大。

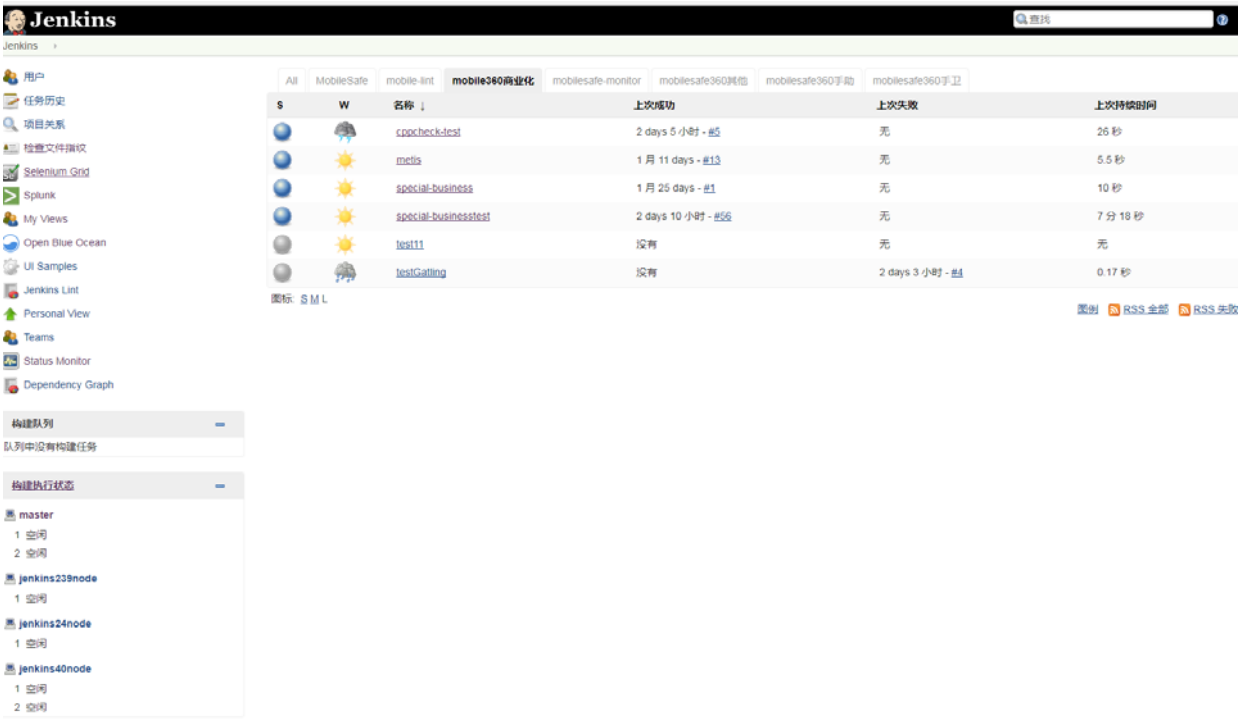
其它比较著名的持续集成工具有：CruiseControl、TeamCity、Continuum 等。

接着我们说说嵌入式软件集成。据我了解，在嵌入式软件开发中，很少人有用到持续集成工具。个人觉得最主要的原因是嵌入试软件与硬件联系比较紧密，很多时候难以满足持续集成的条件——构建自动化测试。

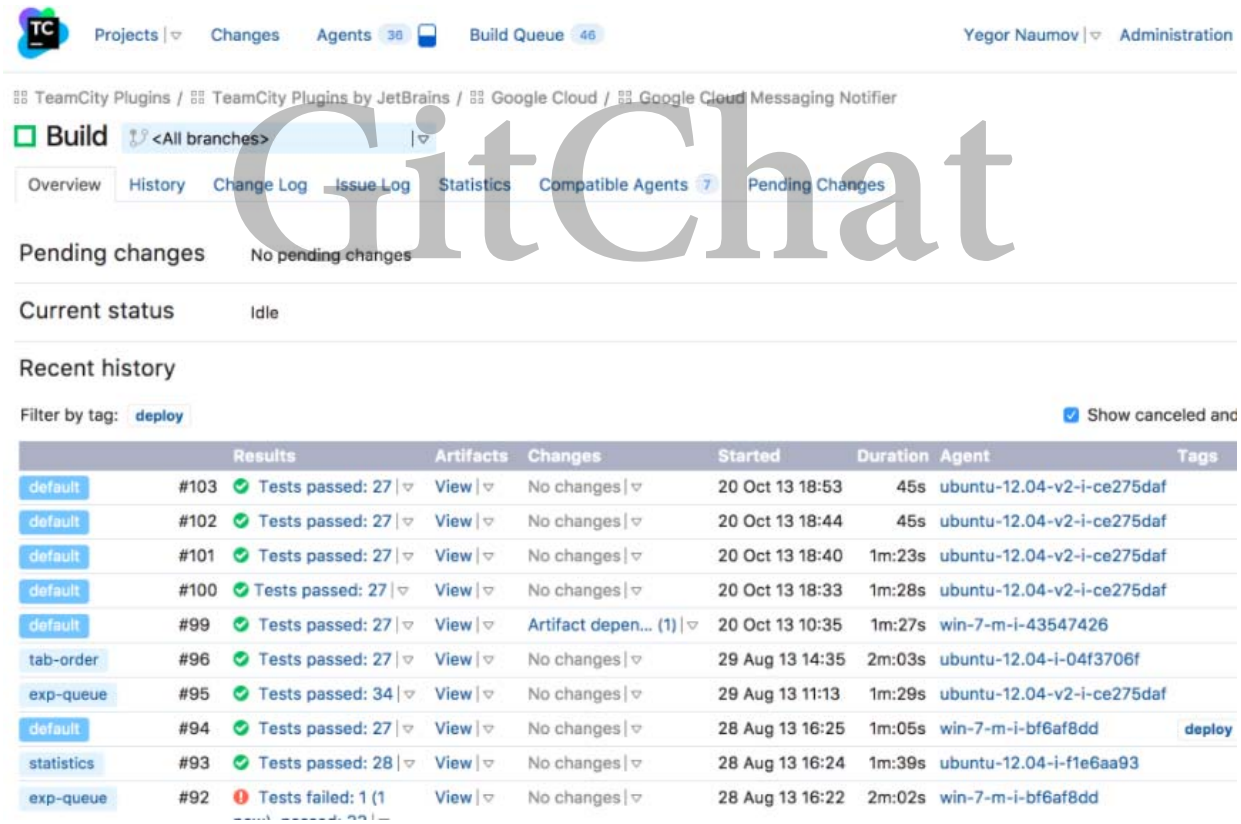
但我们还是可以有所作为。在设计应用软件时，把逻辑业务与硬件相关功能区分开来，对逻辑业务部分编写单元测试，然后做集成测试。

当然，对于小型的嵌入式应用软件就没必要做这个集成工作了。

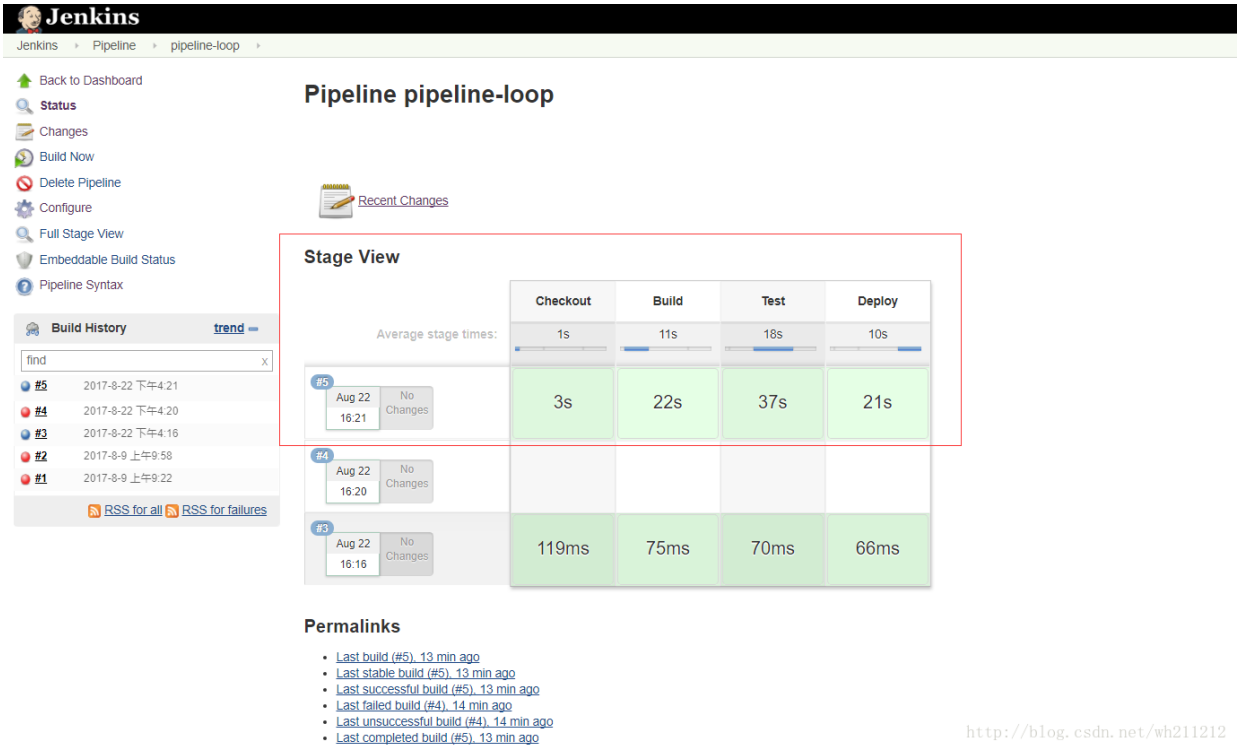
下图为持续集成最常用的开源平台 Jenkins 主界面平台：



下图为持续集成最常用的开源平台Tteamcity 主界面平台：



下图为持续集成最常用的开源平台 Jenkins Pipeline 主界面平台：

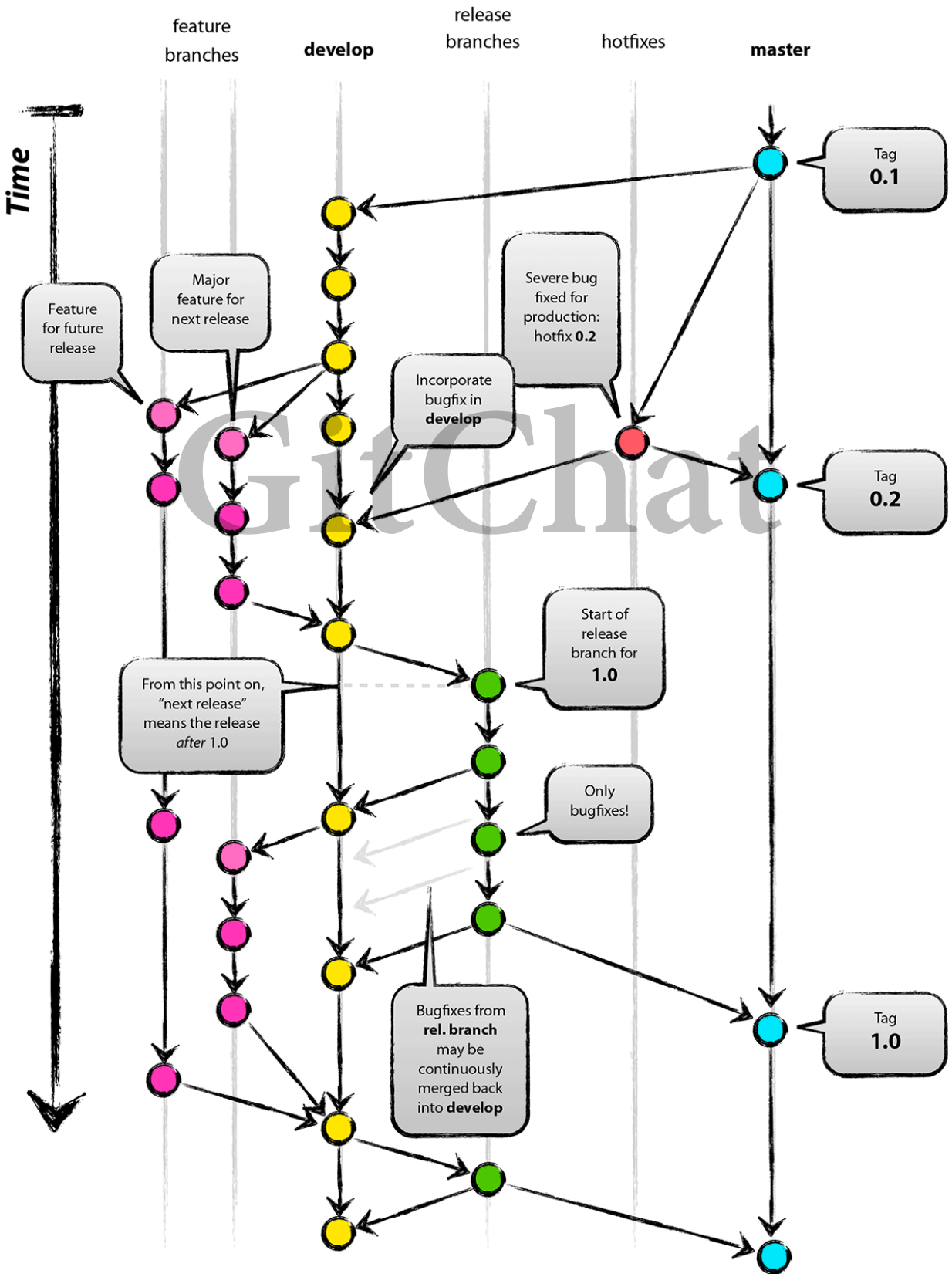


持续集成 Jenkins Pipeline 例子代码如下：

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                echo 'Checkout'
                checkout([$class: 'GitSCM', branches: [[name:
'*/master']], doGenerateSubmoduleConfigurations: false,
extensions: [], submoduleCfg: [], userRemoteConfigs:
[[credentialsId: '500378f5-a6e4-4255-984e-61537fe0e455', url:
'git@gitlab.aniu.so:aniu-yunwei/game-of-life.git']]])
            }
        }
        stage('Build') {
            steps {
                echo 'Building'
                sh 'mvn clean install' # 可以用自己的 mvn clean
deploy + 参数替代
            }
        }
        stage('Test') {
            steps {
                echo 'Testing'
                sh 'mvn clean verify sonar:sonar' # 此处可以使用mvn
test替代，笔者这步是检测代码的质量同步到自己的代码质量检测平台。
            }
        }
        stage('Deploy') {
            steps {
```

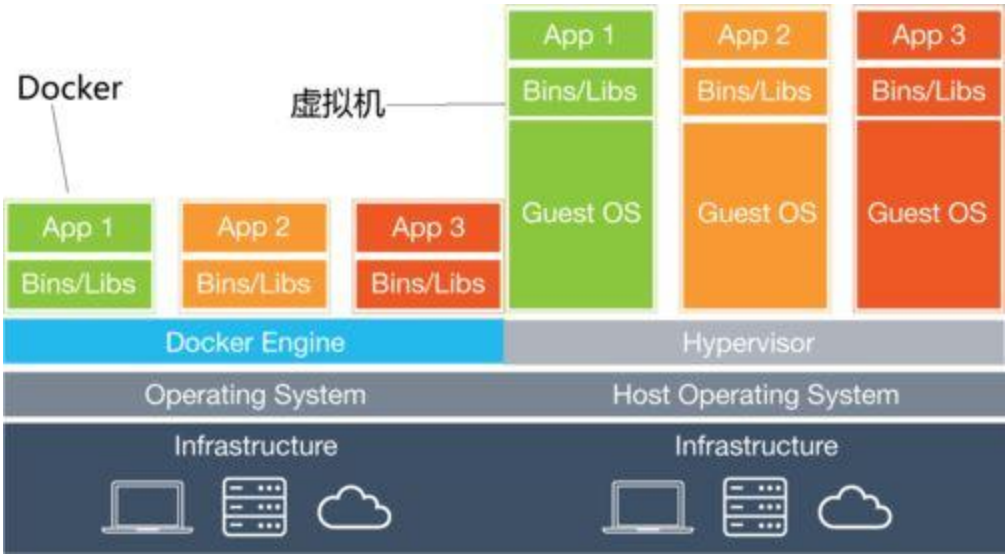
```
                    echo 'Deploying'
                    sh 'mvn clean deploy'   # 此处调用脚本或者ansible、saltstak，部署到远程
                }
            }
        }
    }
```

持续集成代码仓库管理 Git and Git FLOW的工作流程，如下图所示：

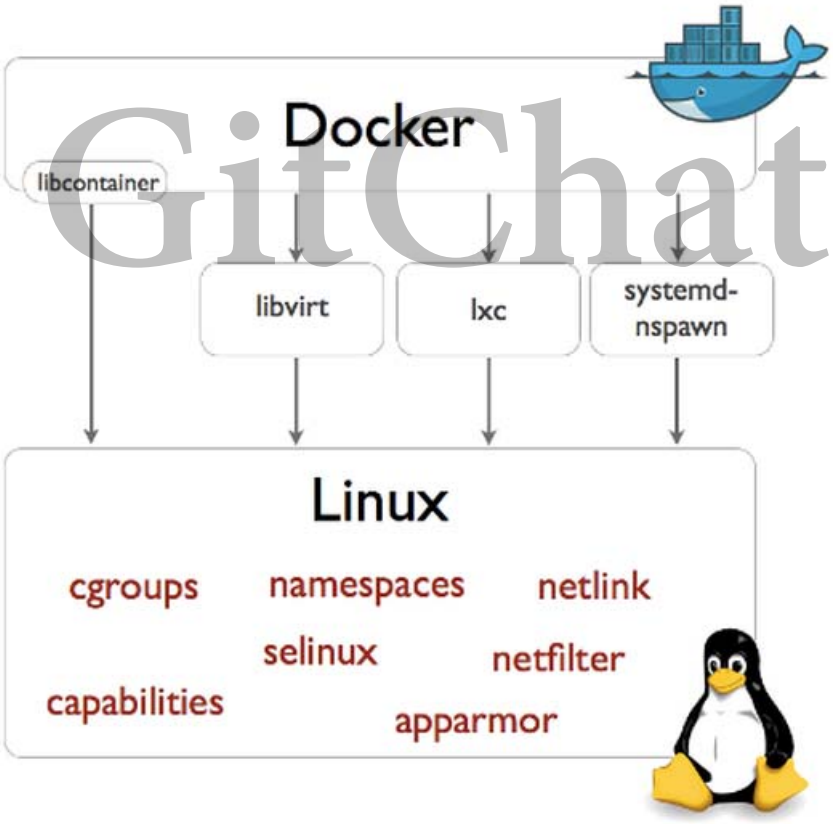


下面我们看看持续集成中 Docker and Kubernetes平台的应用。

下图为 Docker 的架构设计图。

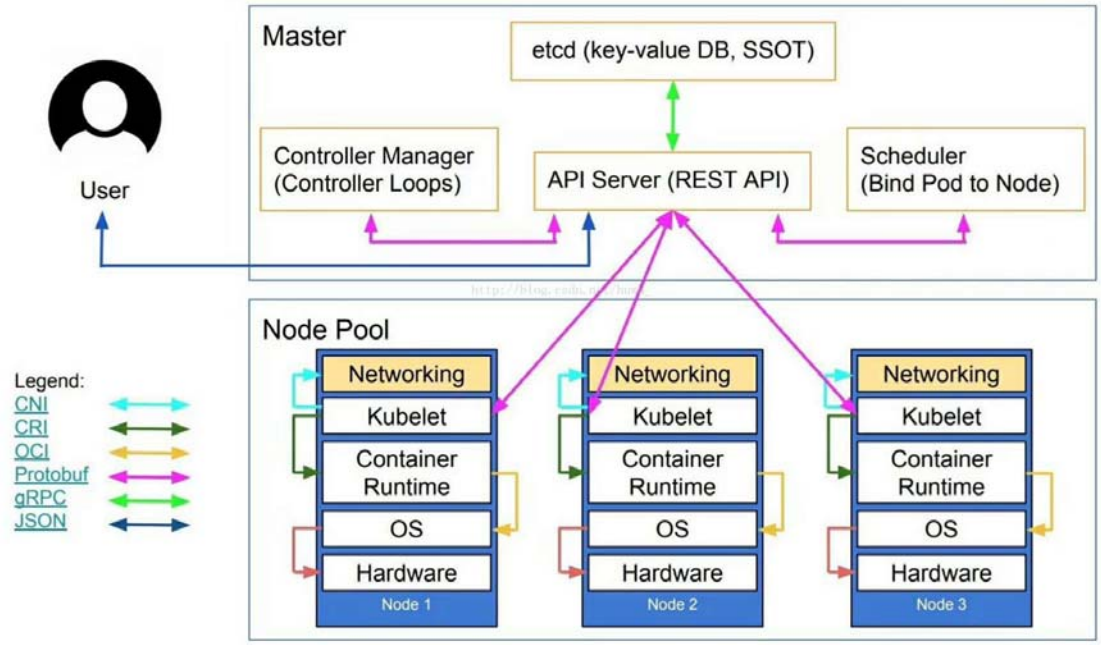


Docker的 Linux 原理架构，如下图所示。

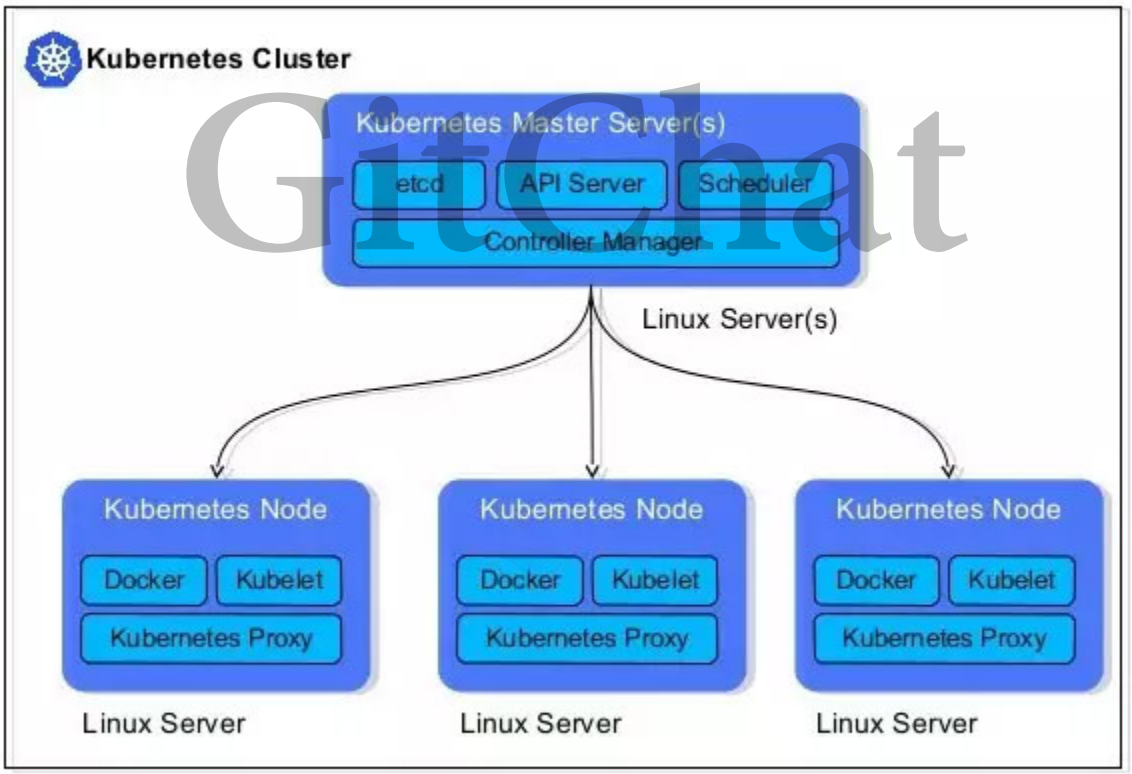


下面为 Kubernetes的架构设计图。

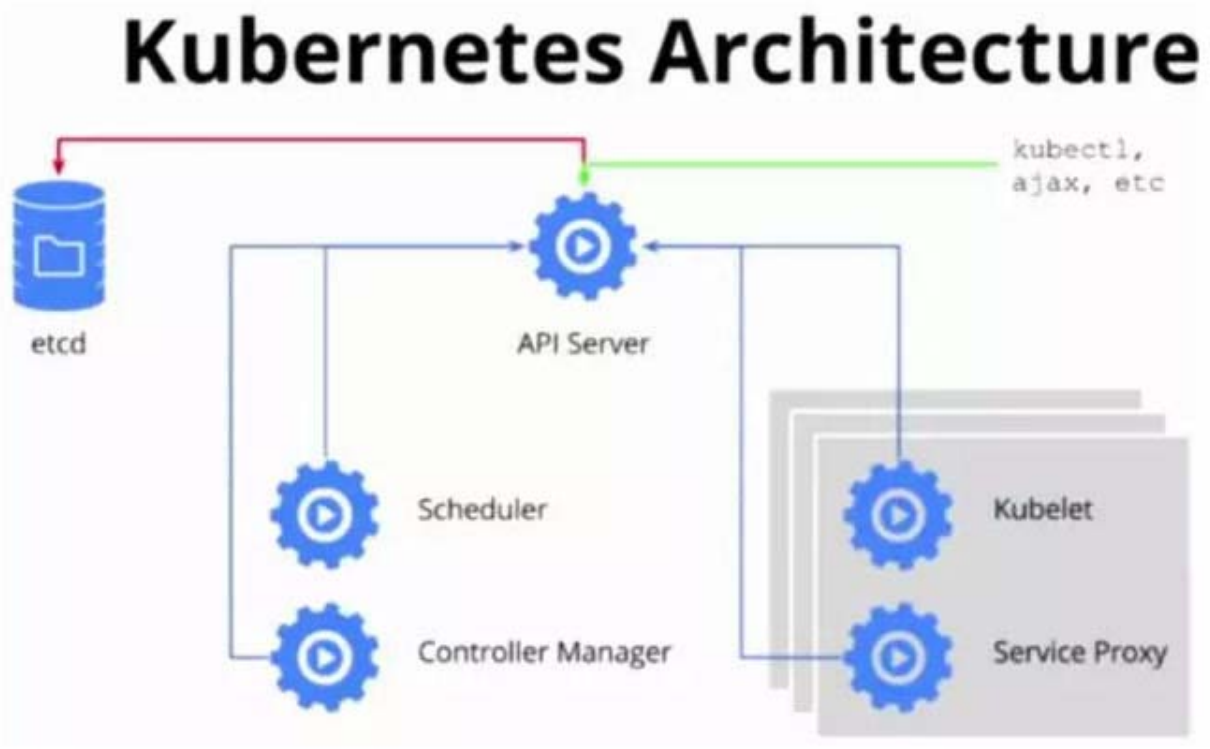
Kubernetes' high-level component architecture



Kubernetes Architectural Overview



@wattsteve redhat



中小型企业如何设计架构 CI 平台——DevOps持续集成的一些架构

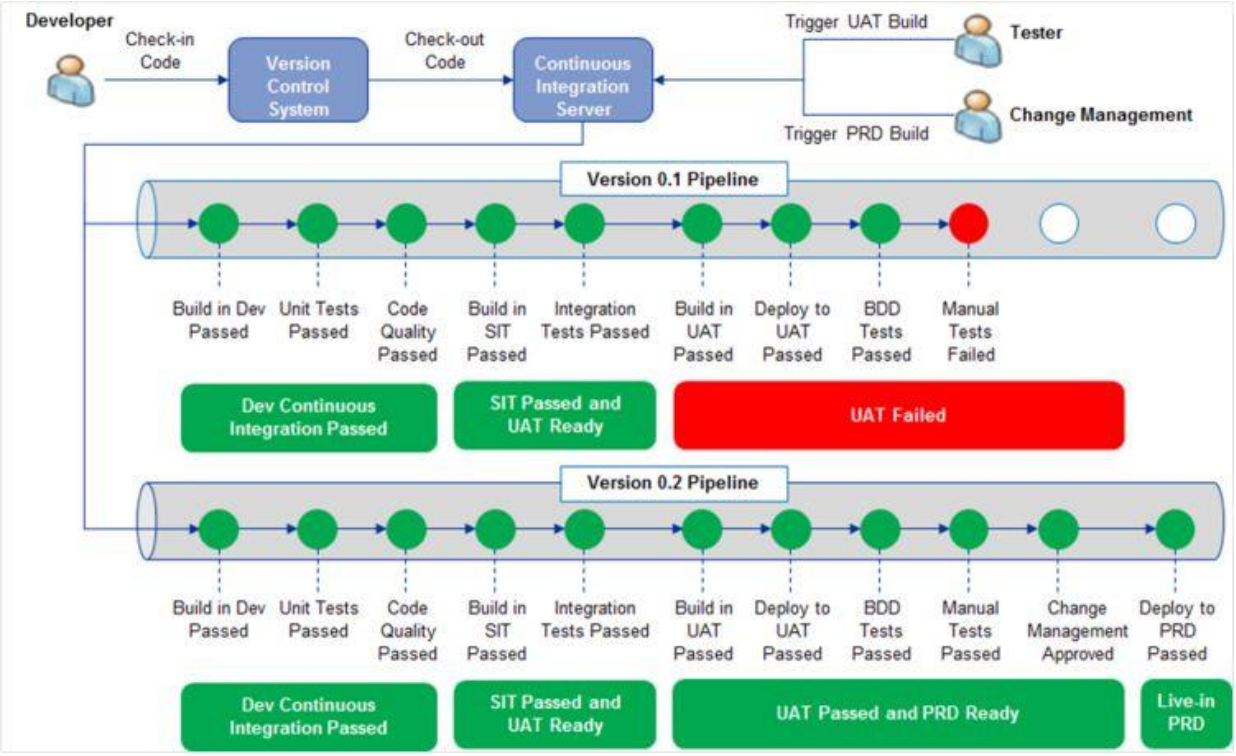
随着软件发布迭代的频率越来越高，传统的“瀑布型”（开发—测试—发布）模式已经不能满足快速交付的需求。2009 年左右 DevOps 应运而生，简单地来说，就是更好的优化开发（ DEV ）、测试（ QA ）、运维（ OPS ）的流程，开发运维一体化，通过高度自动化工具与流程来使得软件构建、测试、发布更加快捷、频繁和可靠。 DevOps 是一个完整的面向IT运维的工作流，以 IT 自动化以及持续集成（ CI ）、持续部署（ CD ）为基础，来优化程式开发、测试、系统运维等所有环节。

纵观各个 DevOps 实践公司的技术资料，最全面最经典的是 Flickr 的10+ deploys per day 最佳实践提到的 DevOps Tools 的技术关键点:

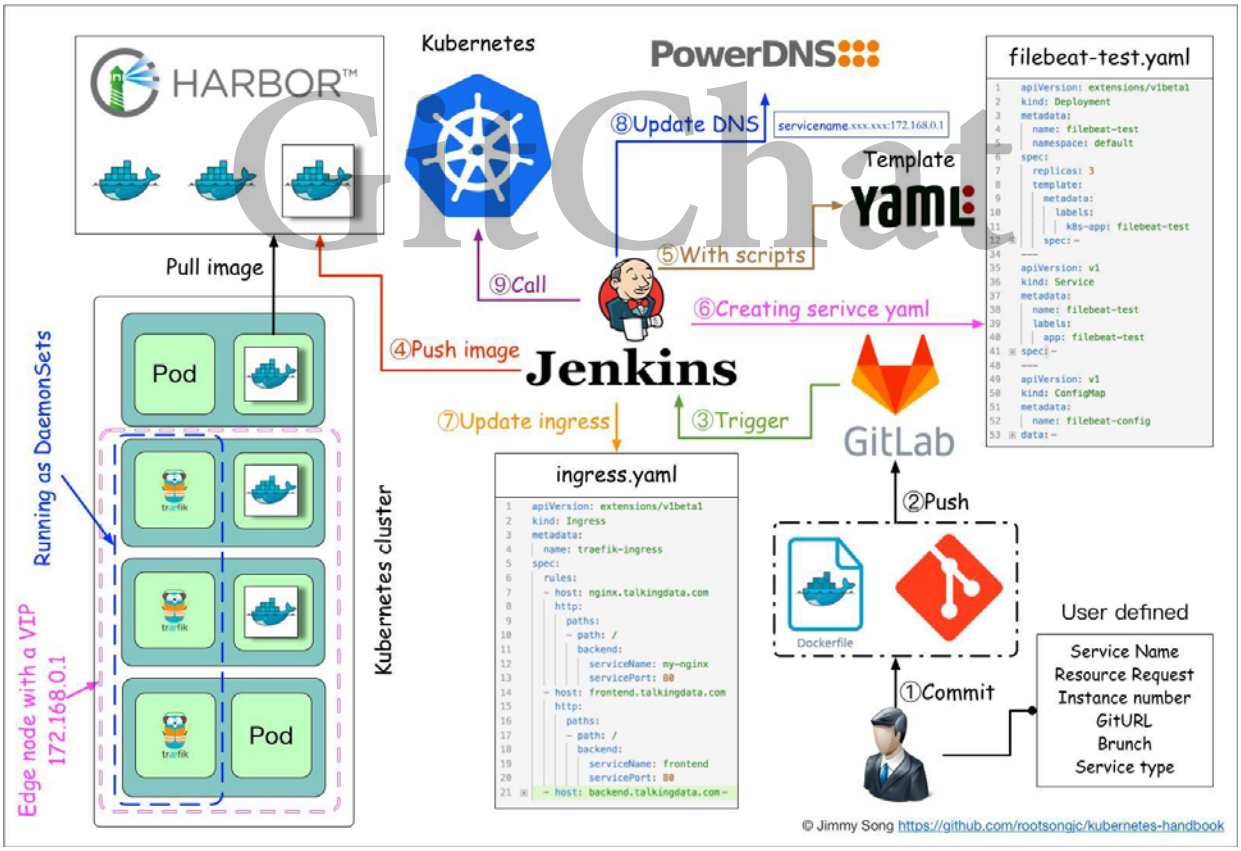
1. Automated Infrastructure（自动化，系统之间的集成）
2. Shared Version Control（ SVN 共享源码）
3. One Step Build and Deploy（持续构建和部署）
4. Feature Flags（主干开发）
5. Shared Metrics
6. IRC and IM Robots（信息整合）

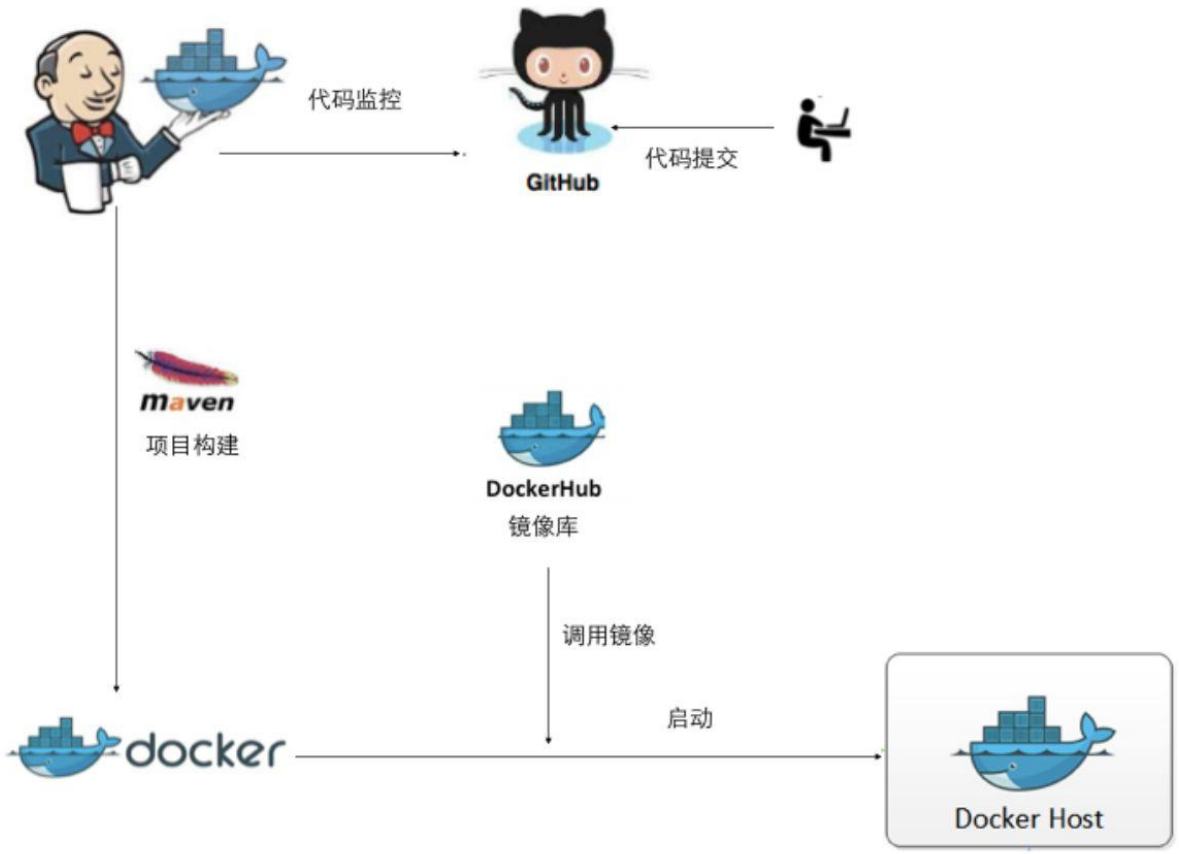
以上的技术要点由持续集成/部署一线贯穿，主干开发是进行持续集成的前提，自动化以及代码周边集中管理是实施持续集成的必要条件。毫无疑问，DevOps 是持续集成思想的延伸，持续集成/部署是 DevOps 的技术核心，在没有自动化测试、持续集成/部署之下，DevOps就是空中楼阁。

下图为持续集成全流程架构：

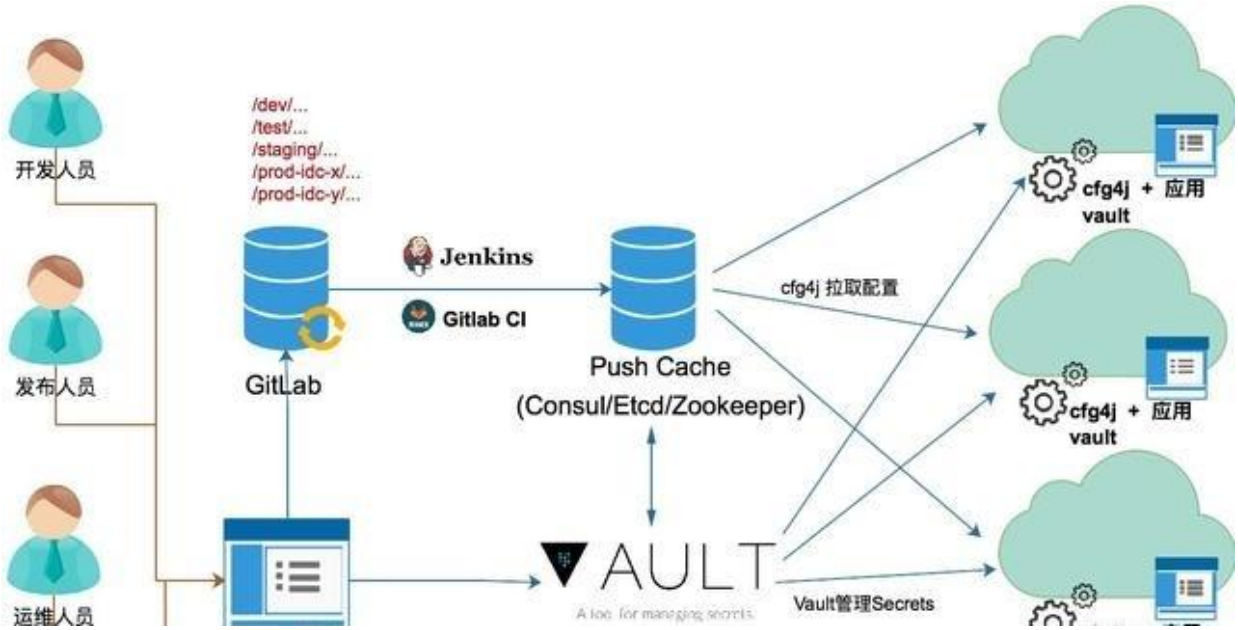


持续集成和 K8s 容器集成的架构，请见下面两张图所示。





持续集成和 DevOps 全流程架构图，请见下面两张图所示。



关于 DevOps 的技术栈与工具链

Everything is Code，DevOps 也同样要通过技术工具链完成持续集成、持续交付、用户反馈和系统优化的整合。Elasticbox 整理了 60+ 开源工具与分类，其中包括版本控制&协作开发工具、自动化构建和测试工具、持续集成&交付工具、部署工具、维护工具、监控，警告&分析工具等等，补充了一些国内的服务，可以让你更好的执行实施 DevOps 工作流。

- 版本控制&协作开发：GitHub、GitLab、BitBucket、SubVersion、Coding、Bazaar。
- 自动化构建和测试：Apache Ant、Maven、Selenium、PyUnit、QUnit、JMeter、Gradle、PHPUnit。
- 持续集成&交付：Jenkins、Capistrano、BuildBot、Fabric、Tinderbox、Travis CI、flow.ci Continuum、LuntBuild、CruiseControl、Integrity、Gump、Go。
- 容器平台: Docker、Rocket、Ubuntu（LXC）、第三方厂商如（AWS/阿里云）。
- 配置管理：Chef、Puppet、CFengine、Bash、Rudder、Powershell、RunDeck、Saltstack、Ansible。
- 微服务平台：OpenShift、Cloud Foundry、Kubernetes、Mesosphere。
- 服务开通：Puppet、Docker Swarm、Vagrant、Powershell、OpenStack Heat。
- 日志管理：Logstash、CollectD、StatsD。
  监控，警告&分析：Nagios、Ganglia、Sensu、zabbix、ICINGA、Graphite、Kibana。

持续集成的实践路线、架构和场景落地

持续集成的实践路线包含代码仓库的纳管、分支的管理策略、自动化测试、测试基线、自动化构件，以及生产环境的维护等等，在一些中小企业落地的时候要根据不同业务和需求场景来衡量。

维护一个单一的源码仓库

这种做法主张对项目的源代码使用一个修订版控制系统。所有需要用来构建该项目的素材都应该放到仓库里。按照惯例，采取这样的做法并且是在一个修订版本控制社区里，该系统应该是可以基于一个全新的签出做构建，而无需任何额外的依赖。极限编程倡导者 Martin Fowler 还提到，在工具支持分支的情况下，对它的使用应该最小化。相反，我们推荐的是将变更集成进来而不是同时维护软件的多个版本。主线（或者主干）应该是软件可工作版本代码的存放位置。

提醒：该原则不能单从字面上去理解，这并不意味着你只需要一个单个的仓库。这里的关键是所有构建项目所需的素材都可以在一个仓库里找到。该项目应当可以基于一个全新的签出构建，并且不需要额外的依赖或者手动步骤。

这里“项目”的定义取决于你，如果代码仓库很小的话它可能意味着整个可交付的产品，或者可以是组织好的代码里任意逻辑模块或者组件。

警告：该原则原本建议不要在版本控制系统里使用分支。相反，它建议项目由始至终仅在一个单一分支下开发。

不过，我并不赞同这一点。在绝大多数组织里，在多个分支下并行开发是很有必要的。企业往往需要支持产品之前发布的版本，修复其中的错误，而其他的团队成员则开始下一个版本的工作。这就需要在代码库里维护多个分支。

构建自动化

构建系统应当一条命令就能办到。许多构建工具，比如 make，已经存在很多年了。其他更多近期涌现的工具经常用在持续集成的环境里。构建的自动化应该包含自动集成，这通常包括部署到一个类生产环境里。在许多情况下，构建脚本不仅可以编译二进制文件，还可以生成文档、网站页面、统计信息和发行版媒介（如 Debian DEB，Red Hat RPM 或 Windows MSI文件）。

提醒：构建的自动化应当包括诸如编译代码，执行单元测试以及集成测试等步骤。它们也许还包括许多其他工具，如前面文章里描述的代码质量检查、语义检查、衡量技术债等。绝大多数现代构建工具都支持这些额外的集成，而且应该可以用于建设持续集成环境。

在现实世界的项目里，不同的团队可能负责开发系统的不同部分，每个团队都拥有自己的仓库。在这种情况下，几乎不可能（没有重大工作的话）而且也完全没必要基于整个产品做自动化构建。一般来说，为系统的每个单独部分开发自动构建就足够了。

警告：定义 CI 流程的目的，除了自动化构建流程外，是否还有其他的投入点？作为 CI 流程的一部分，你计划测量哪些指标？很多时候，我见到的是 CI 设定被视为单独只是开发人员的工具。

延伸之前一点的话，CI 不是敏捷开发 / DevOps，它们只是针对整个组织成功实施 CI 流程所使用的工具之一。敏捷开发 / DevOps 的范式可以超越软件开发的技术层面，并扩展到

组织的文化里。

人人每天都提交到基线

通过定期提交，每位提交者都能借此减少冲突变更的次数。一周工作产出在签入时与其他功能冲突的风险可能很难解决。在更早期的阶段，系统某块领域的小冲突会促使团队成员就其所做的改变进行沟通。至少每天提交一次更改（每次创建一个功能）通常被认为是持续集成定义的一部分。此外，一般建议每晚进行一次构建。这些是下限。业内持续集成的典型频率预计会高得多。

提醒：代码应至少包含单元测试。像 JUnit 这样的框架可用于轻松地模拟依赖。

特定组件与其他模块的交互应当被模拟取代。

（原文这一部分的Tips可能存在谬误，译者注）

警告：已经完成的工作应当提交到主分支。主分支应当总是可工作版本的软件代码。

如果看到哪次构建失败的话请不要提交分支。你应该先验证下是什么导致的错误，然后尝试尽快解决而不是提交自己的代码。为什么在构建失败的时候不应该签入你自己的代码呢？首先，你自己的提交可能存在一些问题，它可能会破坏一些预期的行为。你不会知道这些问题是什么，除非得知上一次签入时构建的状态。而且每一次签入都有可能因为添加了现有的错误让问题变得更糟。

应当构建每一次提交（到基线的）

系统应当构建每一个合并到当前工作版本的提交，从而验证它们集成地很好。常见的做法是利用自动持续集成，尽管这可以手动完成。对大多数情况而言，持续集成是采用自

动持续集成的同义词，一台持续集成服务器或者守护进程会监控校订版本控制系统的变更，随后自动运行构建流程。

提醒：用户应当分离主分支和其他分支的 CI 工作流。这些步骤包括从编译到打包再到测试。主分支的构建一般应当包含更多的测试。主分支的构建也可能需要运行不同的脚本，因为应用可能需要针对不同的部署平台打包成不同的格式。在其他分支上运行的构建可能根本不需要打包这一步，或者通常局限于与开发人员相同的平台的打包。

“夜间构建”也应当在每晚计划好的时间点执行。相比于其他分支而言，该构建应当包含更多的验证过程。它需要花费更长时间去运行并且执行频度更低。

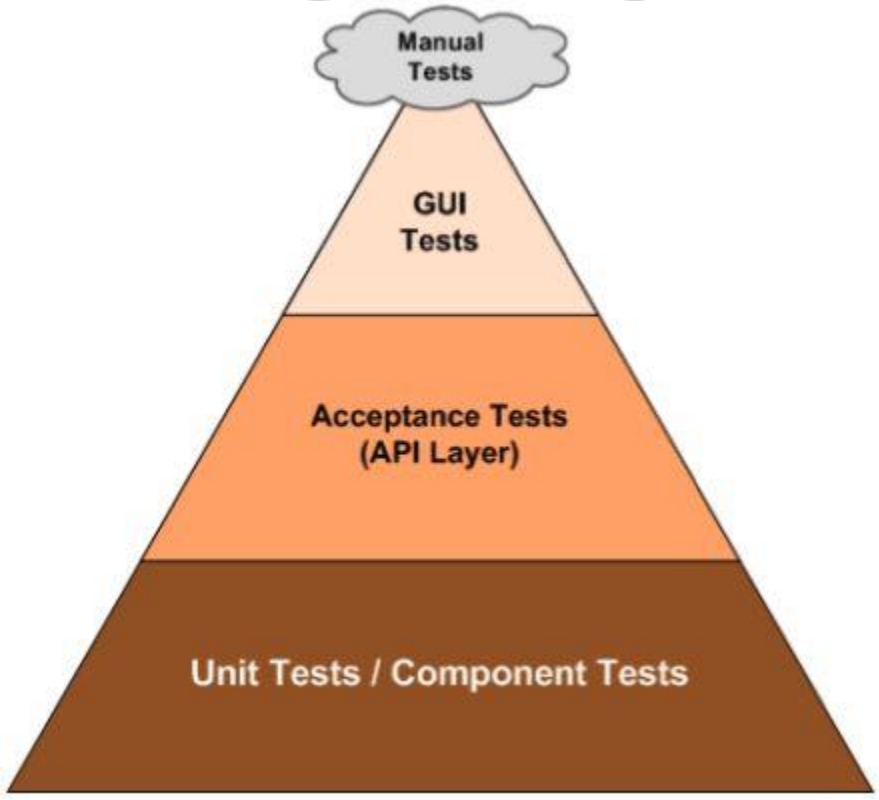
警告：主线分支里不应该注释测试。将测试注释掉的话，我们得到的会是构建状态的错误提示。

引入编码标准的检查是 CI 流程的一部分。代码必须经过自动化工具以及团队成员检查，然后才能签入到主线。

保持构建速度

构建需要快速完成，这样一来如果存在集成问题便会立马被识别出来。

提醒：正如 Martin Fowler 所述，测试金字塔如下图所示。用户的目标应当是拥有更多比例的可以快速执行的测试。这意味着相比于其他类型的测试，用户需要拥有更多的单元测试。



警告：不要依赖大量的 UI 测试，UI 测试是脆弱的，即它们是经常变动的，并且需要花费大量的精力去维护。我建议用户使用像 Selenium 这样的 UI 测试框架来规避 UI 测试过程中遇到的一些问题，例如 UI 元素在屏幕上位置的变动，UI 事件的处理等。

克隆一个生产环境做测试

存在测试环境的话可能会导致测试通过的系统部署到生产环境时发生故障，因为生产环境可能和测试环境有重大差异。然而，建设一个生产环境副本的成本是非常高昂的。相反，测试环境，或是一个单独的预发布环境（staging）应当被建设成实际生产环境的一个可扩展版本，在节省成本的同时维护技术栈的组成和它们之间的细微差别。在这些测试环境里，人们常常使用服务虚拟化以访问那些超出团队控制的依赖（例如，API、第三方应用、服务、大型机等），它们可能仍然在迭代发展，或是在一个虚拟测试实验室里的配置太过复杂。

提醒：这是在现实世界的开发中付诸实践时最难实现的一个原则。这需要构建自动化系统来创建并将软件包部署到反映真实生产环境的一个灰度环境里。除非用户的应用程序是自给自足的，没有任何外部依赖，否则的话这一点很难实现，毕竟，生产环境的复杂度很高。我对复杂产品的建议是投入时间和精力借助虚拟化平台或容器平台（如 Docker）来复制生产环境。持续交付的流水线可用于将构建部署到这些环境。

获取最新的可交付成果变得很容易

为测试和其他相关人员提供构建结果，可以在重建不符合需求的功能时减少所需的返工量。此外，早期测试可以减少代码缺陷在部署前的出镜机会。更早地发现错误，在某些情况下，可以减少解决这些错误所需的工作量。每一位程序员都应该从更新仓库项目代码开始新的一天。这样一来，它们都会保持最新。

提醒：建议使用像 Nexus 这样的资源仓库来存放最新版本的软件包。通常，存储在这样的资源仓库中的包，它们也是通过版本号进行版本控制的。这使得所有参与者都可以轻松获得当前或过去的任意构建包。

警告：只有主线分支中的构建包才能存放在资源仓库里。如果不这么做的话，每当有人新建一个正在工作的分支时会导致现有软件包被覆盖。

人人都可以看到最近一次构建的结果

我们应当能够轻松找出构建是否有问题，如果是的话，谁做了相关的改动。

提醒：所有的现代 CI 服务器都有能力展示包含构建状态的仪表盘，这些也可以被配置用来展示其他的一些指标。

所有 CI 服务器也可以通过配置来实现：当构建完成时发送电子邮件通知。我建议在构建失败时将电子邮件发送给整个团队，以便可以尽快修复。

警告：一次失败的构建并不是奇耻大辱。每个人都会犯错，开发人员也不能幸免。当构建失败时，我们应当将其视为一个受欢迎的结果，因为该问题被及早地发现了。尽早失败并且尽早修复问题是 CI 的关键目标。

CI 不仅仅针对开发人员。通过安装扩展，我们可以从 CI 系统导出各种指标，它们不仅可以用于提高软件质量，还可以提高开发实践的质量。

自动部署

大多数 CI 系统允许在构建完成后运行一些脚本。在大多数情况下，我们可以编写脚本将应用程序部署到每个人都可以查看的一台在线测试服务器。这种思维模式的进一步演变便是持续部署，它需要将软件直接部署到生产环境里，这往往需要额外的自动化手段来防止缺陷或被还原。

警告：并非所有项目都需要自动部署，尤其是当企业服务器运行在客户站点。项目计划决定了客户站点升级到最新版本的时间，这通常是几个月前就计划好的。如果生产站点同样是由正在开发该软件的公司自主托管的话，那么对于持续部署系统的投入将更有收益。持续部署是持续集成流程到位并且运转良好时后续的逻辑步骤。
并非所有的提交都能够产出一个可交付的产品。敏捷社区中最常见的误解是认为每个版本都是可交付的产品。可交付的产品与能正常工作的软件的定义完全不同！

小结

我希望这些信息可以让用户深入了解一些改进 CI 流程实施的最佳做法。CI 在简化软件开发过程中发挥着重要作用。CI 实践的适当调整将提高软件开发过程的整体效率和灵活性。结合这些最佳实践是以更快的上市时间交付高品质软件的诀窍！

本文部分内容参考： https://en.wikipedia.org/wiki/Continuous_integration