如何理解持续集成、持续交付、持续部署?

CICD(持续集成,持续交付/部署)) 介绍

1.CICD 背景;

持续集成(Continuous Integration)是一种软件开发实践，对于提高软件开发效率并保障软件开发质量提供了理论基础。Jenkins 是一个开源软件项目，旨在提供一个开放易用的软件平台，使持续集成变成可能。本文正是从持续集成的基本概念入手，通过具体实例，介绍了如何基于 Jenkins 快速搭建持续集成环境。

持续集成意义

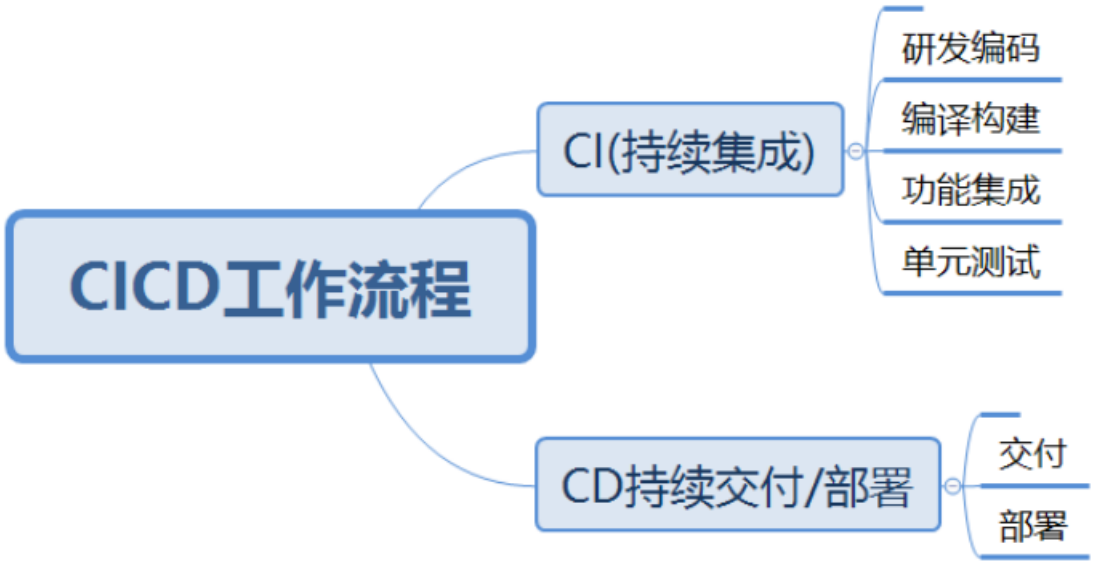
1) 持续集成中的任何一个环节都是自动完成的，无需太多的人工干预，有利于减少重复过程以节省时间、费用和工作量;

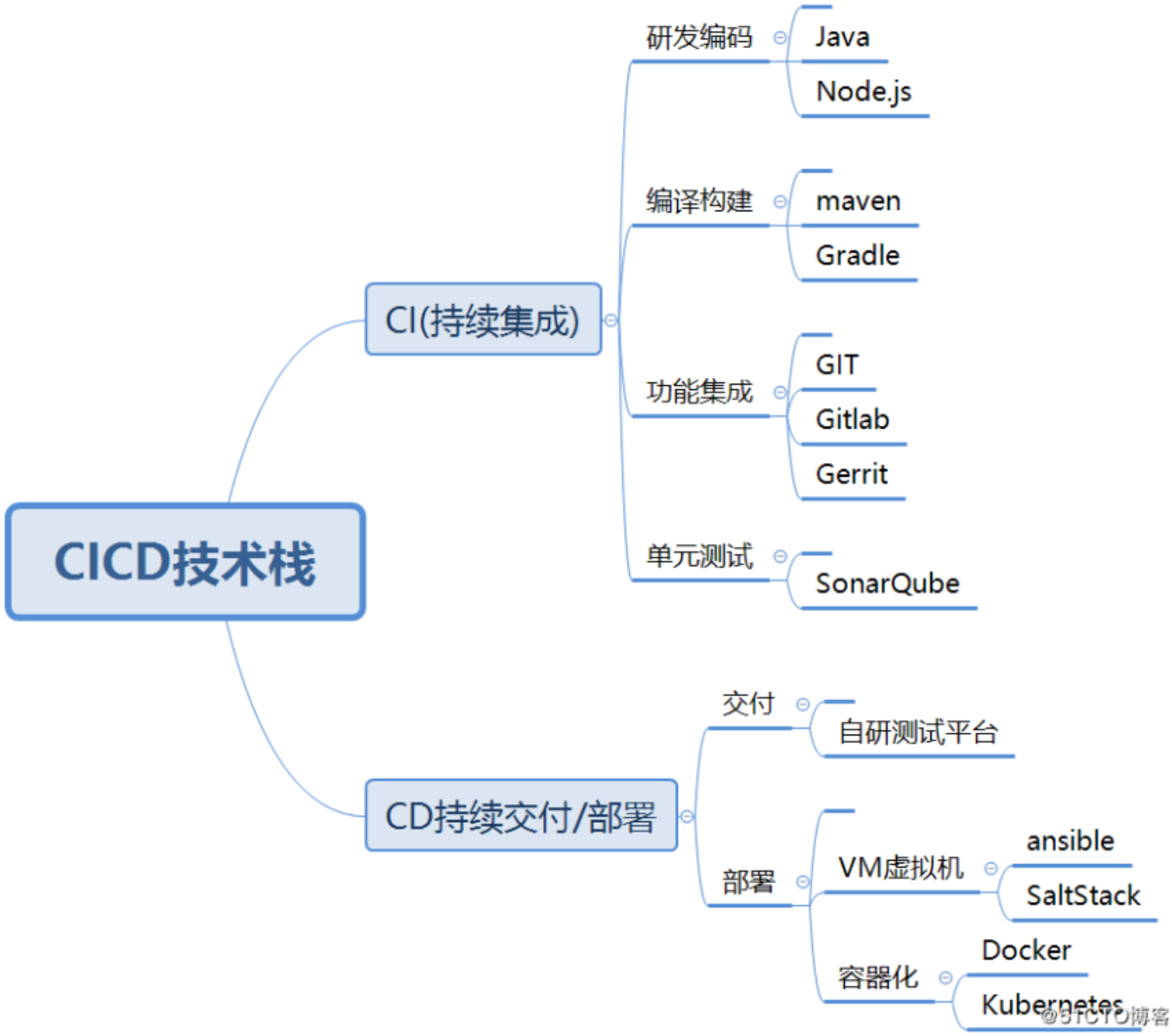
2) 持续集成保障了每个时间点上团队成员提交的代码是能成功集成的。换言之，任何时间点都能第一时间发现软件的集成问题，使任意时间发布可部署的软件成为了可能;

3) 持续集成还能利于软件本身的发展趋势，这点在需求不明确或是频繁性变更的情景中尤其重要，持续集成的质量能帮助团队进行有效决策，同时建立团队对开发产品的信心。

一套标准流程，可以将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





2.**持续集成（Continuous integration，简称 CI）；**
频繁地（一天多次或者 N 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。
3.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。

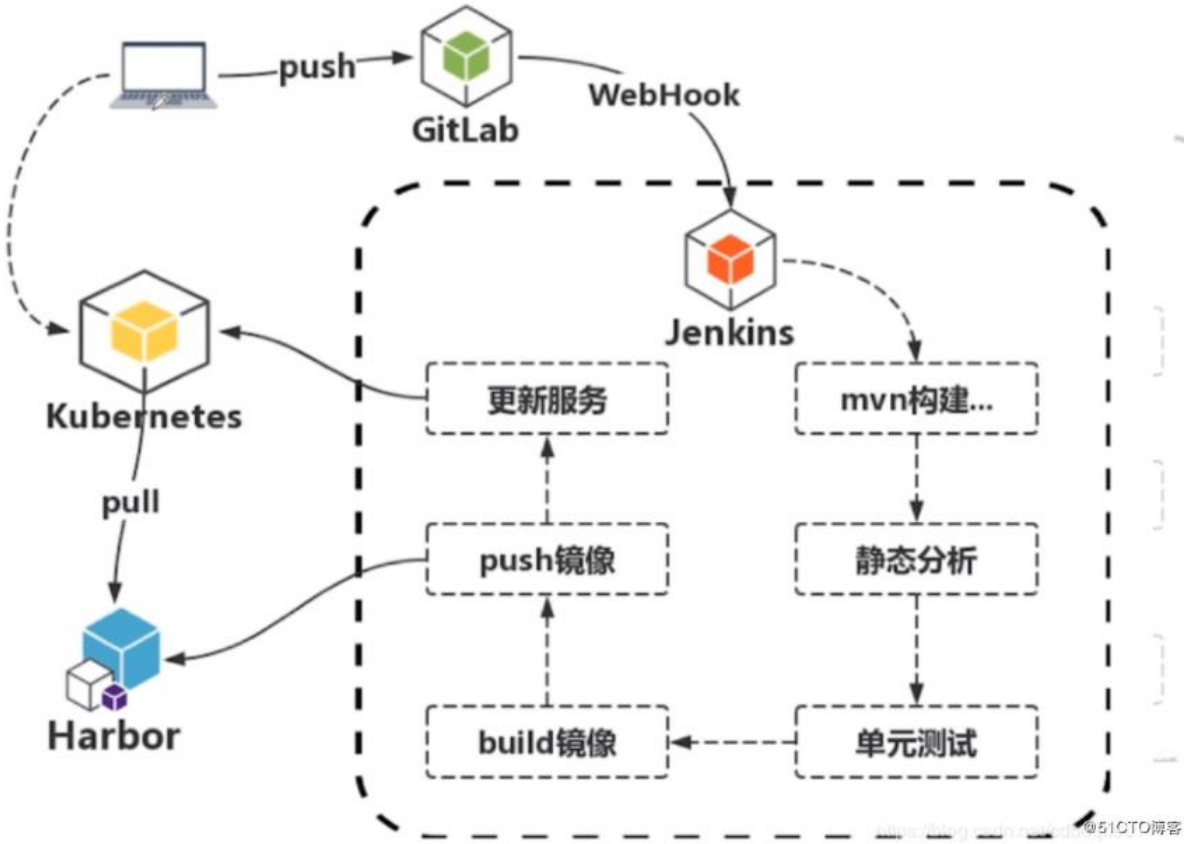
4.持续部署（continuous deployment,简称 CD)

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。

5.CICD 总结

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，能很大地提升开发效率，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。

6.CICD 主流方案;



CI/CD-持续集成/持续交付介绍

在最好的时候创建用户喜欢的高质量应用程序并不是件容易的事情。更何况，要怎样做才能更快地创建用户喜欢的高质量应用程序并且能够不断改进它们呢？这就是需要引入持续集成和持续交付（CI / CD）的地方。

持续集成（CI）

什么是持续集成？

那么，持续集成（CI）究竟是什么呢？它是软件工程师每天频繁地将更新代码的副本传递到共享位置的过程。所有的开发工作都在预定的时间或事件中进行集成，然后自动测试和构建工作。通过 CI，开发过程中出现的错误能被及时发现，这样不仅加速了整个开发周期，而且使软件工程师的工作效率更高。

持续集成有什么好处？

我们不能低估 CI 的好处。因为团队里的人都在同一个产品上进行实时工作，所以在软件开发过程中使用 CI 时，你可以期望实现更快的速度、更好的稳定性和更强的可靠性。并且在开发过程的早期，开发人员能够发现和解决任何编码问题，使它们在成为下游主要问题之前得到纠正。这样可以降低错误代码导致的长期开发（和业务）的成本。

持续集成对于 QA 测试花费的时间也有很大的影响。通过 CI，开发人员不断审查和编辑以前的代码，能够检查到许多小的错误，这些错误在 QA 里通常发现的晚一些。这使得测试人员不仅可以专注研究代码和关注更加紧迫的问题，而且能够同时测试更多的场景。

对开发团队来说，使用 CI 的另一个好处是可以提高编码能力。由于持续发展的自然灵活性，这使得开发人员能够快速、轻松地对代码进行更改，却不会产生运行回归风险。

持续交付（CD）

什么是持续交付？

持续交付（CD）是创建高质量应用程序的第二个难题。CD 是一门软件开发学科，利用技术和工具快速地交付生产阶段的代码。由于大部分交付周期都是自动化的，所以这些交付能够快速地完成。

持续交付有什么好处？

实施持续交付的主要好处是能够加快应用程序的上市时间。使用 CD 的公司能大大增加他们的应用程序发行频率。在没有使用 CD 之前，应用程序发布的频率通常是几个月一次。然而现在使用 CD，你可以一个星期发布一次、甚至每天发布多次应用。在竞争激烈的行业中，速度的提高将会使你处于主要优势。

持续不断的软件版本发布也会根据用户对应用程序的反馈，允许开发团队对其进行微调。这个用户反馈为开发人员提供了所需要的洞察力，并且它优先考虑了用户实际需要的功能请求。同样重要的是，对用户实际上没有用到的应用程序功能，它允许开发人员对其进行优先级排序。

CD 的另一个好处是它能保证每个发行版本的风险较低。当使用 CD 方法发布时，开发团队也会更有信心，因为在整个开发生命周期中，所有内容都经过了多次测试。

任何不考虑转向 CI / CD 的公司都或将被那些使用 CI / CD 方法的竞争对手远远地甩在后面。那么，如何转向 CI / CD？当您准备转向持续集成 / 持续交付（CI / CD）时，需要考量及决定的相关流程有很多。下文将带您了解这些主要流程。

转向 CI/CD 的重要流程

1、分支和合并

你需要组织及考虑的一个主要流程就是你的分支和合并。分支就是开发人员可以在代码的平行部分工作的地方——从一个中央代码库分支出来。分支的好处在于，它允许在不破坏中心代码基础的情况下，在软件构建的不同方面同时进行工作。显然，合并即意味着分支合并到核心代码库。

通过各种版本控制系统，许多开发人员对分支和合并已经很熟悉了。然而，根据您的构建的特别要求，您所分支的内容也有很多不同的策略。有些开发人员将通过维护、功能或团队来进行发布的分支。

您可能会对某种策略非常狂热，但“绝对正确”的分支和合并策略是不存在的，只存在“对您的构建而言正确”的方式与策略。这需要检查您当前的分支和合并策略，并根据您的目标和情况决定需要更改哪些内容。

2、构建自动化

构建自动化意味着您可以自动编译软件构建。持续集成服务器的核心是构建自动化服务器，其工作是在触发或定时的基础上编译和链接源代码。您选择的持续集成服务器将成为您的 CI/CD 环境的支柱。

在查看构建自动化过程时，了解市场上各种可用选项的功能是非常有帮助的。开源公司 Jenkins 现在在 CI/CD 部署中占绝对优势，这通常是一个好的开始。或者至少，在比较其他解决方案时把它作为基准。作为一个开源代码系统，您可能仍需构建一些实用程序,以使构建自动化完全适合您的情况。

3、测试自动化

测试自动化对于 CI/CD 能否按预期工作至关重要。如果没有自动化测试，CI/CD 将很快无法实现快速交付的目标。我们的总体建议是尽可能自动化。这意味着您需要检查您需要执行的各种测试，并决定在您的环境中可以安全地自动进行哪些测试。

建立测试自动化环境可能需要新的技能。然而，这是战略需求，将会提高交付速度，减少错误。至少，您应该自动化代码审查、单元测试、集成测试和系统测试。

4、部署自动化

关于持续交付和持续部署之间的区别，仍然存在一些混淆。简而言之，持续交付意味着持续推出发布就绪代码，而持续部署则意味着持续给用户部署该软件。

无论你在看什么“CD”，对那些不习惯的人来说，这似乎是一个巨大的飞跃。为了让您的组织有信心将软件部署到最终用户，需要一个严密的测试自动化基础设施。

我们的建议是，最好进入流程定义，以实现零接触持续部署的总体目标。虽然领先的持续集成系统通常会考虑自己的持续交付系统，但您可以比现成的参数更进一步。真正的敏捷性需要构建一个基础设施、写好代码，吸引用户使用。

选择开源且完整的 CI/CD 的工具

真正实现 CI / CD 并非易事，pipeline 搭建工作复杂，平滑升级难以保障，服务宕机难以避免……选择一个完整的 CI/CD 工具，将大大助力于 CI/CD 在企业里落地并最终带来生产运维效率的提升。Rancher Labs 新近发布的 CI/CD 工具 Rancher Pipeline，就拥有极简的操作体验，强大的功能整合，并且完全开源。

同时支持多源码管理：在单一环境中同时拉取、使用和管理托管在 GitHub 和 GitLab 的代码；

一键部署，完全可视化的 pipeline 配置，拖拽方式的 pipeline 搭建；

阶段式和阶梯式 pipeline，可自由扩展的步骤系统；

灵活的流程控制：不同的代码分支可以自动匹配不同的 CI 流程，从而支持较为复杂的流程控制

支持多种触发方式：计划任务的触发、来自 GitHub / GitLab 的 webhook 触发、手动触发，以及通过定制化的开发，实现更多种触发方式的支持

良好集成的审批系统：审批系统已与 Rancher 用户管理系统集成，且用户可以在任意阶段插入断点，自由地对任意阶段进行审批

灵活的 pipeline 启停机制：任一环节出错，整个进度可以立即停止，而问题解决之后又可以重新运行。

持续集成、持续交付、持续部署（CI/CD）简介

概述：

软件开发周期中需要一些可以帮助开发者提升速度的自动化工具。其中工具最重要的目的是促进软件项目的持续集成与交付。通过 CI/CD 工具，开发团队可以保持软件更新并将其迅速的投入实践中。CI/CD 也被认为是敏捷开发的最重要实践之一。

一 、持续集成

从上图可以看到，持续集成应该至少包括以下几部分：
自动化构建 Continuous Build
自动化测试 Continuous Test
自动化集成 Continuous Intergration

1、自动化构建

包括以下过程：
将源码编译成为二进制码
打包二进制码
运行自动化测试
生成文档
生成分发媒体（例如：Debian DEB、Red Hat RPM 或者 Windows MSI 文件）
所以，自动化构建，从功能角度分，最关键的是三部分：版本控制工具、构建工具、CI 服务器。而其中最核心的又是构建工具。其他开源的、与持续集成相关的工具也有很多，但大多数是辅助性的工具。

(1)版本控制工具

有时，版本控制又称为配置管理（SCM），所以版本控制工具同时也是配置管理工具。在各类版本控制的开源软件中，最著名的莫过于 CVS、SVN（Subversion）、GIT 三个了。
这三个工具各有千秋。其中，GIT 支持离线工作，更适合开源软件或者开发人员不能集中办公情况下的版本管理工作。同时，SVN 和 GIT 可以配合使用。

（2）构建工具

构建工具是持续集成的核心，它对源代码进行自动化编译、测试、代码检查，以及打包程序、部署（发布）到应用服务器上。从配置管理工具上下载最新源代码后，所有的后续工作几乎都可以通过构建工具完成。
在 java 开发中，比较有名的构建工具就是 Ant、Maven、Gradle。在 PHP 开发中，Phing（基于 Ant）也比较有名。同样的，Maven 也可通过相关的 PHP-Maven 插件完成对 PHP 开发构建的支持。

（3）CI 服务器

CI 服务器的主要作用就是提供一个平台，用于整合版本控制和构建工作，并管理、控制自动化的持续集成。
开源软件中，比较有名的 CI 服务器包括 Jenkins、CruiseControl、Continuum。而比较有名的商业化 CI 服务器是 TeamCity、Bamboo、Pulse 等。

（4）其他工具

很多工具可以通过与构建工具、CI 工具相结合（当然，其中有很多工具也可以单独工作），来完成更多的自动测试、报告生成等工作。根据工具不同，其具体的结合方法也不同，但大体都是通过插件形式进行结合的。**例如：**
Maven 中通过依赖和 plugin 方式引入第三方工具
Jenkins 主要通过各类插件引入第三方工具
这些工具种类实在太多，可以根据实际工作需要进行选择。

2.自动化测试

自动化测试是持续集成必不可少的一部分，基本上，没有自动化测试的持续集成，都很难称之为真正的持续集成。我们希望持续集成能够尽早的暴露问题，但这远非配置一个 Hudson/Jenkins 服务器那么简单，只有真正用心编写了较为完整的测试用例，并一直维护它们，持续集成才能孜孜不倦地运行测试并第一时间报告问题。

测试自动化是使用特定的软件（独立于被测试的软件）来控制测试的执行以及比较实际输出与预期输出。测试自动化可以将某些重复但必要的任务自动化，或者执行某些难以手动执行的额外测试。

自动化测试还包括单元测试、集成测试、系统测试、验收测试、性能测试等，在不同的场景下，它们都能为软件开发带来极大的价值。

二、持续交付

持续交付（Continuous Delivery, CD）是一种软件工程的手段，让软件在短周期内产出，确保软件随时可以被可靠地发布。其目的在于更快、更频繁地构建、测试以及发布软件。通过加强对生产环境的应用进行渐进式更新，这种手段可以降低交付变更的成本与风险。一个简单直观的与可重复的部署过程对于持续交付来说是很重要的。

三、持续部署

如图所示，持续部署与持续交付之间的差异就是前者将部署自动化了。

在持续交付的实践中，交付的目标是 QA，但是实际上，软件最终是要交付到客户手上的。在 SaaS 领域里，持续部署采用得比较广泛，因为服务比较容易做到静默升级。

采用持续部署的前提是自动化测试的覆盖率足够高。

采用持续部署的好处是能减少运维的工作量，缩短新特性从开发到实际交付的周期。

四、CI/CD 具体实现

常见 CI/CD 工具及其比较:

这里的支持，意思应该是直接的支持，例如 Jenkins，其实和 git 结合也很简单，通过脚本就可以实现。

五、持续集成工具集之 Jenkins 简介

Jenkins 是一个可扩展的持续集成引擎。

1.主要用于：

持续、自动地构建/测试软件项目。

监控一些定时执行的任务。Jenkins 拥有的特性包括：

2.Jenkins 拥有的特性包括：

易于安装-只要把 jenkins.war 部署到 servlet 容器，不需要数据库支持。

易于配置-所有配置都是通过其提供的 web 界面实现。

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。

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。

3.Jenkins 的出现

目前持续集成(CI)已成为当前许多软件开发团队在整个软件开发生命周期内侧重于保证代码质量的常见做法。它是一种实践，旨在缓和和稳固软件的构建过程。

并且能够帮助您的开发团队应对如下挑战：

软件构建自动化 ：配置完成后，CI 系统会依照预先制定的时间表，或者针对某特定事件，对目标软件进行构建。

构建可持续的自动化检查 ：CI 系统能持续地获取新增或修改后签入的源代码，也就是说，当软件开发团队需要周期性的检查新增或修改后的代码时，CI 系统会不断确认这些新代码是否破坏了原有软件的成功构建。这减少了开发者们在检查彼此相互依存的代码中变化情况需要花费的时间和精力。

构建可持续的自动化测试 ：构建检查的扩展部分，构建后执行预先制定的一套测试规则，完成后触发通知(Email,RSS 等等)给相关的当事人。

生成后后续过程的自动化 :当自动化检查和测试成功完成，软件构建的周期中可能也需要一些额外的任务，诸如生成文档、打包软件、部署构件到一个运行环境或者软件仓库。这样，构件才能更迅速地提供给用户使用。

部署一个 CI 系统需要的最低要求是，一个可获取的源代码的仓库，一个包含构建脚本的项目。

下图概括了 CI 系统的基本结构：图略

4.使用 Jenkins 的一些理由：

该系统的各个组成部分是按如下顺序来发挥作用的：

开发者检入代码到源代码仓库。

CI 系统会为每一个项目创建了一个单独的工作区。当预设或请求一次新的构建时，它将把源代码仓库的源码存放到对应的工作区。

CI 系统会在对应的工作区内执行构建过程。

（配置如果存在）构建完成后，CI 系统会在一个新的构件中执行定义的一套测试。完成后触发通知(Email,RSS 等等)给相关的当事人。

（配置如果存在）如果构建成功，这个构件会被打包并转移到一个部署目标(如应用服务器)或存储为软件仓库中的一个新版本。软件仓库可以是 CI 系统的一部分，也可以是一个外部的仓库，诸如一个文件服务器或者像 Java.NET、 SourceForge 之类的网站。

CI 系统通常会根据请求发起相应的操作，诸如即时构建、生成报告，或者检索一些构建好的构件。

Jenkins 就是这么一个 CI 系统。之前叫做 Hudson。

是所有 CI 产品中在安装和配置上最简单

基于 Web 访问，用户界面非常友好、直观和灵活，在许多情况下，还提供了 AJAX 的即时反馈。

Jenkins 是基于 Java 开发的(如果你是一个 Java 开发人员，这是非常有用的)，但它不仅限于构建基于 Java 的软件。

Jenkins 拥有大量的插件。这些插件极大的扩展了 Jenkins 的功能；它们都是开源的，而且它们可以直接通过 web 界面来进行安装与管理。

5.Jenkins 的目标

Jenkins 的主要目标是监控软件开发流程，快速显示问题。所以能保证开发人员以及相关人员省时省力提高开发效率。

CI 系统在整个开发过程中的主要作用是控制：当系统在代码存储库中探测到修改时，它将运行构建的任务委托给构建过程本身。如果构建失败了，那么 CI 系统将通知相关人员，然后继续监视存储库。它的角色看起来是被动的；但它确能快速反映问题。

特别是它具有以下优点：

Jenkins 一切配置都可以在 web 界面上完成。有些配置如 MAVEN_HOME 和 Email，只需要配置一次，所有的项目就都能用。当然也可以通过修改 XML 进行配置。

支持 Maven 的模块(Module)，Jenkins 对 Maven 做了优化，因此它能自动识别 Module，每个 Module 可以配置成一个 job。相当灵活。

测试报告聚合，所有模块的测试报告都被聚合在一起，结果一目了然，使用其他 CI，这几乎是件不可能完成的任务。

构件指纹(artifact fingerprint)，每次 build 的结果构件都被很好的自动管理，无需任何配置就可以方便的浏览下载。

CI/CD 持续集成/持续部署 敏捷开发

CI/CD 持续集成/持续部署 敏捷开发

敏捷软件开发（英语：Agile software development），又称敏捷开发，是一种从 1990 年代开始逐渐引起广泛关注的一些新型软件开发方法，是一种应对快速变化的需求的一种软件开发能力。它们的具体名称、理念、过程、术语都不尽相同，相对于“非敏捷”，更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发过程中人的作用。

1，CI/CD 持续集成/持续部署

持续集成(Continuous integration)是一种软件开发实践，即团队开发成员经常集成它们的工作，通过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，从而尽早地发现集成错误。

持续部署（continuous deployment）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 engineering productivity 最大化。

持续交付（英语：Continuous delivery，缩写为 CD），是一种软件工程手法，让软件产品的产出过程在一个短周期内完成，以保证软件可以稳定、持续的保持在随时可以释出的状况。它的目标在于让软件的建置、测试与释出变得更快以及更频繁。这种方式可以减少软件开发的成本与时间，减少风险。

与 DevOps 的关系

持续交付与 DevOps 的含义很相似，所以经常被混淆。但是它们是不同的两个概念。DevOps 的范围更广，它以文化变迁为中心，特别是软件交付过程所涉及的多个团队之间的合作（开发、运维、QA、管理部门等），并且将软件交付的过程自动化。另壹方面，持续交付是壹种自动化交付的手段，关注点在于将不同的过程集中起来，并且更快、更频繁地执行这些过程。因此，DevOps 可以是持续交付的壹个产物，持续交付直接汇入 DevOps;

与持续部署的关系

有时候，持续交付也与持续部署混淆。持续部署意味着所有的变更都会被自动部署到生产环境中。持续交付意味着所有的变更都可以被部署到生产环境中，但是出于业务考虑，可以选择不部署。如果要实施持续部署，必须先实施持续交付。

1.1、jenkins 是什么？

Jenkins 是一个可扩展的持续集成引擎。

主要用用途：

持续、自动的构建/测试软件项目。

监控一些定时执行的任务。、

特点：

易于安装，只要把 jenkins.war 部署到 servlet 容器

易于配置-所有配置都通过其提供的 web 界面实现。

集成 RSS/E-mail 通过 RSS 发布构建结果或当构件完成是通过 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 能够跟踪那次构建生成哪些 jar，那次构建使用哪个版本的 jar

插入支持：支持扩展插件，可以开发适合自己团队的使用的工具。

Jenkins 的目标是监控软件的开发流程，快速显示问题。所以能保证开发人员省事又省力提高开发效率。

2、项目版本迭代控制:、

现有的版本控制工具，如 Github、GitLab、SVN、CVS 等主流工具..

构建及测试:通过 Jenkins 实现自动构建和测试,还有商业软件 BAMBOO 来持续集成。这个收费的。免费就用 Jenkins..

交付:以 Docker 镜像形式进行交付，提交至镜像仓库;

2.1 SVN 服务器：

Subversion 是一个版本控制系统，相对于的 RCS、CVS，采用了分支管理系统，它的设计目标就是取代 CVS。互联网上免费的版本控制服务多基于 Subversion。

Subversion 的版本库可以通过网络访问，从而使用户可以在不同的电脑上进行操作。从某种程度上来说，允许用户在各自的空间里修改和管理同一组数据可以促进团队协作。因为修改不再是单线进行（单线进行也就是必须一个一个进行），开发进度会进展迅速。此外，由于所有的工作都已版本化，也就不必担心由于错误的更改而影响软件质量—如果出现不正确的更改，只要撤销那一次更改操作即可。某些版本控制系统本身也是软件配置管理系统（SCM），这种系统经过精巧的设计，专门用来管理源代码树，并且具备许多与软件开发有关的特性——比如对编程语言的支持或者提供程序构建工具。不过 Subversion 并不是这样的系统，它是一个通用系统，可以管理任何类型的文件集。

2.2 CVS 服务器：

CVS（Concurrent Versions System）版本控制系统是一种 GNU 软件包，主要用于在多人开发环境下源码的维护。Concurrent 有并发的、协作的、一致的等含义。实际上 CVS 可以维护任意文档的开发和使用，例如共享文件的编辑修改，而不仅仅局限于程序设计。CVS 维护的文件类型可以是文本类型也可以是二进制类型。CVS 用 Copy-Modify-Merge（拷贝、修改、合并）变化表支持对文件的同时访问和修改。它明确地将源文件的存储和用户的工作空间独立开来，并使其并行操作。CVS 基于客户端/服务器的行为使其可容纳多个用户。这一特性使得 CVS 成为位于不同地点的人同时处理数据文件（特别是程序的源代码）时的首选。

所有重要的免费软件项目都使用 CVS 作为其程序员之间的中心点，以便能够综合各程序员的改进和更改。这些项目包括 GNOME、KDE、THE GIMP 和 Wine 等。

2.3 GIt/github：

GIT （分布式版本控制系统）

Git 是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 的读音为/gɪt/。

Git 是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。Torvalds 开始着手开发 Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得 BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。例如 很多 Freedesktop 的项目迁移到了 Git 上。

gitHub 是一个面向开源及私有软件项目的托管平台，因为只支持 git 作为唯一的版本库格式进行托管，故名 gitHub。

gitHub 于 2008 年 4 月 10 日正式上线，除了 git 代码仓库托管及基本的 Web 管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。目前，其注册用户已经超过 350 万，托管版本数量也是非常之多，其中不乏知名开源项目 Ruby on Rails、jQuery、python 等。

什么是持续集成（CI）/持续部署（CD）？

在软件开发中经常会提到 持续集成(Continuous Integration)（CI）和 持续交付(Continuous Delivery)（CD）这几个术语。但它们真正的意思是什么呢?
在谈论软件开发时，经常会提到 持续集成(Continuous Integration)（CI）和 持续交付(Continuous Delivery)（CD）这几个术语。但它们真正的意思是什么呢？在本文中，我将解释这些和相关术语背后的含义和意义，例如 持续测试(Continuous Testing)和 持续部署(Continuous Deployment)。

概览

工厂里的装配线以快速、自动化、可重复的方式从原材料生产出消费品。同样，软件交付管道以快速、自动化和可重复的方式从源代码生成发布版本。如何完成这项工作的总体设计称为“持续交付”（CD）。启动装配线的过程称为“持续集成”（CI）。确保质量的过程称为“持续测试”，将最终产品提供给用户的过程称为“持续部署”。一些专家让这一切简单、顺畅、高效地运行，这些人被称为 运维开发(DevOps)践行者。

“持续”是什么意思?

“持续”用于描述遵循我在此提到的许多不同流程实践。这并不意味着“一直在运行”，而是“随时可运行”。在软件开发领域，它还包括几个核心概念/最佳实践。这些是：

- 频繁发布：持续实践背后的目标是能够频繁地交付高质量的软件。此处的交付频率是可变的，可由开发团队或公司定义。对于某些产品，一季度、一个月、一周或一天交付一次可能已经足够频繁了。对于另一些来说，一天可能需要多次交付也是可行的。所谓持续也有“偶尔、按需”的方面。最终目标是相同的：在可重复、可靠的过程中为最终用户提供高质量的软件更新。通常，这可以通过很少甚至无需用户的交互或掌握的知识来完成（想想设备更新）。
- 自动化流程：实现此频率的关键是用自动化流程来处理软件生产中的方方面面。这包括构建、测试、分析、版本控制，以及在某些情况下的部署。
- 可重复：如果我们使用的自动化流程在给定相同输入的情况下始终具有相同的行为，则这个过程应该是可重复的。也就是说，如果我们把某个历史版本的代码作为输入，我们应该得到对应相同的可交付产出。这也假设我们有相同版本的外部依赖项（即我们不创建该版本代码使用的其它交付物）。理想情况下，这也意味着可以对管道中的流程进行版本控制和重建（请参阅稍后的 DevOps 讨论）。
- 快速迭代：“快速”在这里是个相对术语，但无论软件更新/发布的频率如何，预期的持续过程都会以高效的方式将源代码转换为交付物。自动化负责大部分工作，但自动化处理的过程可能仍然很慢。例如，对于每天需要多次发布候选版更新的产品来说，一轮 集成测试(integrated testing)下来耗时就要大半天可能就太慢了。

什么是“持续交付管道”？

将源代码转换为可发布产品的多个不同的 任务(task)和 作业(job)通常串联成一个软件“管道”，一个自动流程成功完成后会启动管道中的下一个流程。这些管道有许多不同的叫法，例如持续交付管道、部署管道和软件开发管道。大体上讲，程序管理者在管道执行时管理管道各部分的定义、运行、监控和报告。

持续交付管道是如何工作的?

软件交付管道的实际实现可以有很大不同。有许多程序可用在管道中，用于源代码跟踪、构建、测试、指标采集，版本管理等各个方面。但整体工作流程通常是相同的。单个业务流程/工作流应用程序管理整个管道，每个流程作为独立的作业运行或由该应用程序进行阶段管理。通常，在业务流程中，这些独立作业是以应用程序可理解并可作为工作流程管理的语法和结构定义的。
这些作业被用于一个或多个功能（构建、测试、部署等）。每个作业可能使用不同的技术或多种技术。关键是作业是自动化的、高效的，并且可重复的。如果作业成功，则工作流管理器将触发管道中的下一个作业。如果作业失败，工作流管理器会向开发人员、测试人员和其他人发出警报，以便他们尽快纠正问题。这个过程是自动化的，所以比手动运行一组过程可更快地找到错误。这种快速排错称为 快速失败(fail fast)，并且在抵达管道端点方面同样有价值。

“快速失败”是什么意思?

管道的工作之一就是快速处理变更。另一个是监视创建发布的不同任务/作业。由于编译失败或测试未通过的代码可以阻止管道继续运行，因此快速通知用户此类情况非常重要。快速失败指的是在管道流程中尽快发现问题并快速通知用户的方式，这样可以及时修正问题并重新提交代码以便使管道再次运行。通常在管道流程中可通过查看历史记录来确定是谁做了那次修改并通知此人及其团队。

所有持续交付管道都应该被自动化吗?

管道的几乎所有部分都是应该自动化的。对于某些部分，有一些人为干预/互动的地方可能是有意义的。一个例子可能是 用户验收测试(user-acceptance testing)（让最终用户试用软件并确保它能达到他们想要/期望的水平）。另一种情况可能是部署到生产环境时用户希望拥有更多的人为控制。当然，如果代码不正确或不能运行，则需要人工干预。
有了对“持续”含义理解的背景，让我们看看不同类型的持续流程以及它们在软件管道上下文中的含义。

什么是“持续集成”?

持续集成（CI）是在源代码变更后自动检测、拉取、构建和（在大多数情况下）进行单元测试的过程。持续集成是启动管道的环节（尽管某些预验证 —— 通常称为 上线前检查(pre-flight checks) —— 有时会被归在持续集成之前）。
持续集成的目标是快速确保开发人员新提交的变更是好的，并且适合在代码库中进一步使用。

持续集成是如何工作的?

持续集成的基本思想是让一个自动化过程监测一个或多个源代码仓库是否有变更。当变更被推送到仓库时，它会监测到更改、下载副本、构建并运行任何相关的单元测试。

持续集成如何监测变更?

目前，监测程序通常是像 Jenkins 这样的应用程序，它还协调管道中运行的所有（或大多数）进程，监视变更是其功能之一。监测程序可以以几种不同方式监测变更。这些包括：

- 轮询：监测程序反复询问代码管理系统，“代码仓库里有什么我感兴趣的新东西吗？”当代码管理系统有新的变更时，监测程序会“唤醒”并完成其工作以获取新代码并构建/测试它。

- 定期：监测程序配置为定期启动构建，无论源码是否有变更。理想情况下，如果没有变更，则不会构建任何新内容，因此这不会增加额外的成本。
- 推送：这与用于代码管理系统检查的监测程序相反。在这种情况下，代码管理系统被配置为提交变更到仓库时将“推送”一个通知到监测程序。最常见的是，这可以以 webhook 的形式完成 —— 在新代码被推送时一个 挂勾(hook)的程序通过互联网向监测程序发送通知。为此，监测程序必须具有可以通过网络接收 webhook 信息的开放端口。

什么是“预检查”（又称“上线前检查”）？

在将代码引入仓库并触发持续集成之前，可以进行其它验证。这遵循了最佳实践，例如 测试构建(test build)和 代码审查(code review)。它们通常在代码引入管道之前构建到开发过程中。但是一些管道也可能将它们作为其监控流程或工作流的一部分。

例如，一个名为 Gerrit 的工具允许在开发人员推送代码之后但在允许进入（ Git 远程）仓库之前进行正式的代码审查、验证和测试构建。Gerrit 位于开发人员的工作区和 Git 远程仓库之间。它会“接收”来自开发人员的推送，并且可以执行通过/失败验证以确保它们在被允许进入仓库之前的检查是通过的。这可以包括检测新变更并启动构建测试（CI 的一种形式）。它还允许开发者在那时进行正式的代码审查。这种方式有一种额外的可信度评估机制，即当变更的代码被合并到代码库中时不会破坏任何内容。

什么是“单元测试”？

单元测试（也称为“提交测试”），是由开发人员编写的小型的专项测试，以确保新代码独立工作。“独立”这里意味着不依赖或调用其它不可直接访问的代码，也不依赖外部数据源或其它模块。如果运行代码需要这样的依赖关系，那么这些资源可以用 模拟(mock)来表示。模拟是指使用看起来像资源的 代码存根(code stub)，可以返回值，但不实现任何功能。

在大多数组织中，开发人员负责创建单元测试以证明其代码正确。事实上，一种称为 测试驱动开发(test-driven develop)（TDD）的模型要求将首先设计单元测试作为清楚地验证代码功能的基础。因为这样的代码可以更改速度快且改动量大，所以它们也必须执行很快。

由于这与持续集成工作流有关，因此开发人员在本地工作环境中编写或更新代码，并通单元测试来确保新开发的功能或方法正确。通常，这些测试采用断言形式，即函数或方法的给定输入集产生给定的输出集。它们通常进行测试以确保正确标记和处理出错条件。有很多单元测试框架都很有用，例如用于 Java 开发的 JUnit 。

什么是“持续测试”？

持续测试是指在代码通过持续交付管道时运行扩展范围的自动化测试的实践。单元测试通常与构建过程集成，作为持续集成阶段的一部分，并专注于和其它与之交互的代码隔离的测试。

除此之外，可以有或者应该有各种形式的测试。这些可包括：

- 集成测试 验证组件和服务组合在一起是否正常。
- 功能测试 验证产品中执行功能的结果是否符合预期。
- 验收测试 根据可接受的标准验证产品的某些特征。如性能、可伸缩性、抗压能力和容量。

所有这些可能不存在于自动化的管道中，并且一些不同类型的测试分类界限也不是很清晰。但是，在交付管道中持续测试的目标始终是相同的：通过持续的测试级别证明代码的质量可以在正在进行的发布中使用。在持续集成快速的原则基础上，第二个目标是快速发现问题并提醒开发团队。这通常被称为快速失败。

除了测试之外，还可以对管道中的代码进行哪些其它类型的验证?

除了测试是否通过之外，还有一些应用程序可以告诉我们测试用例执行（覆盖）的源代码行数。这是一个可以衡量代码量指标的例子。这个指标称为代码覆盖率(code-coverage)，可以通过工具（例如用于 Java 的 JaCoCo ）进行统计。

还有很多其它类型的指标统计，例如代码行数、复杂度以及代码结构对比分析等。诸如 SonarQube 之类的工具可以检查源代码并计算这些指标。此外，用户还可以为他们可接受的“合格”范围的指标设置阈值。然后可以在管道中针对这些阈值设置一个检查，如果结果不在可接受范围内，则流程终端上。SonarQube 等应用程序具有很高的可配置性，可以设置仅检查团队感兴趣的内容。

什么是“持续交付”？

持续交付（CD）通常是指整个流程链（管道），它自动监测源代码变更并通过构建、测试、打包和相关操作运行它们以生成可部署的版本，基本上没有任何人为干预。

持续交付在软件开发过程中的目标是自动化、效率、可靠性、可重复性和质量保障（通过持续测试）。

持续交付包含持续集成（自动检测源代码变更、执行构建过程、运行单元测试以验证变更），持续测试（对代码运行各种测试以保障代码质量），和（可选）持续部署（通过管道发布版本自动提供给用户）。

如何在管道中识别/跟踪多个版本?

版本控制是持续交付和管道的关键概念。持续意味着能够经常集成新代码并提供更新版本。但这并不意味着每个人都想要“最新、最好的”。对于想要开发或测试已知的稳定版本的内部团队来说尤其如此。因此，管道创建并轻松存储和访问的这些版本化对象非常重要。

在管道中从源代码创建的对象通常可以称为 工件(artifact)。工件在构建时应该有应用于它们的版本。将版本号分配给工件的推荐策略称为 语义化版本控制(semantic versioning)。（这也适用于从外部源引入的依赖工件的版本。）

语义版本号有三个部分： 主要版本(major)、 次要版本(minor) 和 补丁版本(patch)。（例如，1.4.3 反映了主要版本 1，次要版本 4 和补丁版本 3。）这个想法是，其中一个部分的更改表示工件中的更新级别。主要版本仅针对不兼容的 API 更改而递增。当以 向后兼容(backward-compatible)的方式添加功能时，次要版本会增加。当进行向后兼容的版本 bug 修复时，补丁版本会增加。这些是建议的指导方针，但只要团队在整个组织内以一致且易于理解的方式这样做，团队就可以自由地改变这种方法。例如，每次为发布完成构建时增加的数字可以放在补丁字段中。

如何“分销”工件?

团队可以为工件分配 分销(promotion)级别以指示适用于测试、生产等环境或用途。有很多方法。可以用 Jenkins 或 Artifactory 等应用程序进行分销。或者一个简单的方案可以在版本号字符串的末尾添加标签。例如，-snapshot 可以指示用于构建工件的代码的最新版本（快照）。可以使用各种分销策略或工具将工件“提升”到其它级别，例如 -milestone 或 -production，作为工件稳定性和完备性版本的标记。

如何存储和访问多个工件版本?

从源代码构建的版本化工件可以通过管理 工件仓库(artifact repository)的应用程序进行存储。工件仓库就像构建工件的版本控制工具一样。像 Artifactory 或 Nexus 这类应用可以接受版本化工件，存储和跟踪它们，并提供检索的方法。
管道用户可以指定他们想要使用的版本，并在这些版本中使用管道。

什么是“持续部署”？

持续部署（CD）是指能够自动提供持续交付管道中发布版本给最终用户使用的想法。根据用户的安装方式，可能是在云环境中自动部署、app 升级（如手机上的应用程序）、更新网站或只更新可用版本列表。
这里的一个重点是，仅仅因为可以进行持续部署并不意味着始终部署来自管道的每组可交付成果。它实际上指，通过管道每套可交付成果都被证明是“可部署的”。这在很大程度上是由持续测试的连续级别完成的（参见本文中的持续测试部分）。
管道构建的发布成果是否被部署可以通过人工决策，或利用在完全部署之前“试用”发布的各种方法来进行控制。

在完全部署到所有用户之前，有哪些方法可以测试部署?

由于必须回滚/撤消对所有用户的部署可能是一种代价高昂的情况（无论是技术上还是用户的感知），已经有许多技术允许“尝试”部署新功能并在发现问题时轻松“撤消”它们。这些包括：

蓝/绿测试/部署

在这种部署软件的方法中，维护了两个相同的主机环境 —— 一个“蓝色” 和一个“绿色”。（颜色并不重要，仅作为标识。）对应来说，其中一个是“生产环境”，另一个是“预发布环境”。
在这些实例的前面是调度系统，它们充当产品或应用程序的客户“网关”。通过将调度系统指向蓝色或绿色实例，可以将客户流量引流到期望的部署环境。通过这种方式，切换指向哪个部署实例（蓝色或绿色）对用户来说是快速，简单和透明的。
当新版本准备好进行测试时，可以将其部署到非生产环境中。在经过测试和批准后，可以更改调度系统设置以将传入的线上流量指向它（因此它将成为新的生产站点）。现在，曾作为生产环境实例可供下一次候选发布使用。
同理，如果在最新部署中发现问题并且之前的生产实例仍然可用，则简单的更改可以将客户流量引流回到之前的生产实例 —— 有效地将问题实例“下线”并且回滚到以前的版本。然后有问题的新实例可以在其它区域中修复。

金丝雀测试/部署

在某些情况下，通过蓝/绿发布切换整个部署可能不可行或不是期望的那样。另一种方法是为 金丝雀(canary)测试/部署。在这种模型中，一部分客户流量被重新引流到新的版本部署中。例如，新版本的搜索服务可以与当前服务的生产版本一起部署。然后，可以将 10% 的搜索查询引流到新版本，以在生产环境中对其进行测试。
如果服务那些流量的新版本没问题，那么可能会有更多的流量会被逐渐引流过去。如果仍然没有问题出现，那么随着时间的推移，可以对新版本增量部署，直到 100% 的流量都调度到新版本。这有效地“更替”了以前版本的服务，并让新版本对所有客户生效。

功能开关

对于可能需要轻松关掉的新功能（如果发现问题），开发人员可以添加 功能开关(feature toggles)。这是代码中的 if-then 软件功能开关，仅在设置数据值时才激活新代码。此数据值可以是全局可访问的位置，部署的应用程序将检查该位置是否应执行新代码。如果设置了数据值，则执行代码；如果没有，则不执行。
这为开发人员提供了一个远程“终止开关”，以便在部署到生产环境后发现问题时关闭新功能。

暗箱发布

在 暗箱发布(dark launch)中，代码被逐步测试/部署到生产环境中，但是用户不会看到更改（因此名称中有 暗箱(dark)一词）。例如，在生产版本中，网页查询的某些部分可能会重定向到查询新数据源的服务。开发人员可收集此信息进行分析，而不会将有关接口，事务或结果的任何信息暴露给用户。
这个想法是想获取候选版本在生产环境负载下如何执行的真实信息，而不会影响用户或改变他们的经验。随着时间的推移，可以调度更多负载，直到遇到问题或认为新功能已准备好供所有人使用。实际上功能开关标志可用于这种暗箱发布机制。

什么是“运维开发”？

运维开发（DevOps）是关于如何使开发和运维团队更容易合作开发和发布软件的一系列想法和推荐的实践。从历史上看，开发团队研发了产品，但没有像客户那样以常规、可重复的方式安装/部署它们。在整个周期中，这组安装/部署任务（以及其它支持任务）留给运维团队负责。这经常导致很多混乱和问题，因为运维团队在后期才开始介入，并且必须在短时间内完成他们的工作。同样，开发团队经常处于不利地位 —— 因为他们没有充分测试产品的安装/部署功能，他们可能会对该过程中出现的问题感到惊讶。
这往往导致开发和运维团队之间严重脱节和缺乏合作。DevOps 理念主张是贯穿整个开发周期的开发和运维综合协作的工作方式，就像持续交付那样。

持续交付如何与运维开发相交?

持续交付管道是几个 DevOps 理念的实现。产品开发的后期阶段（如打包和部署）始终可以在管道的每次运行中完成，而不是等待产品开发周期中的特定时间。同样，从开发到部署过程中，开发和运维都可以清楚地看到事情何时起作用，何时不起作用。要使持续交付管道循环成功，不仅要通过与开发相关的流程，还要通过与运维相关的流程。
说得更远一些，DevOps 建议实现管道的基础架构也会被视为代码。也就是说，它应该自动配置、可跟踪、易于修改，并在管道发生变化时触发新一轮运行。这可以通过将管道实现为代码来完成。

什么是“管道即代码”？

管道即代码(pipeline-as-code)是通过编写代码创建管道作业/任务的通用术语，就像开发人员编写代码一样。它的目标是将管道实现表示为代码，以便它可以与代码一起存储、评审、跟踪，如果出现问题并且必须终止管道，则可以轻松地重建。有几个工具允许这样做，如 Jenkins 2 。

DevOps 如何影响生产软件的基础设施?

传统意义上，管道中使用的各个硬件系统都有配套的软件（操作系统、应用程序、开发工具等）。在极端情况下，每个系统都是手工设置来定制的。这意味着当系统出现问题或需要更新时，这通常也是一项自定义任务。这种方法违背了持续交付的基本理念，即具有易于重现和可跟踪的环境。

多年来，很多应用被开发用于标准化交付（安装和配置）系统。同样， 虚拟机(virtual machine)被开发为模拟在其它计算机之上运行的计算机程序。这些 VM 要有管理程序才能在底层主机系统上运行，并且它们需要自己的操作系统副本才能运行。
后来有了 容器(container)。容器虽然在概念上与 VM 类似，但工作方式不同。它们只需使用一些现有的操作系统结构来划分隔离空间，而不需要运行单独的程序和操作系统的副本。因此，它们的行为类似于 VM 以提供隔离但不需要过多的开销。
VM 和容器是根据配置定义创建的，因此可以轻易地销毁和重建，而不会影响运行它们的主机系统。这允许运行管道的系统也可重建。此外，对于容器，我们可以跟踪其构建定义文件的更改 —— 就像对源代码一样。
因此，如果遇到 VM 或容器中的问题，我们可以更容易、更快速地销毁和重建它们，而不是在当前环境尝试调试和修复。
这也意味着对管道代码的任何更改都可以触发管道新一轮运行（通过 CI），就像对代码的更改一样。这是 DevOps 关于基础架构的核心理念之一。

Devops

最近老是碰到这个名词，所以想了解一下这个到底是撒玩意？
DevOps（Development 和 Operations 的组合词）是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。
它是一种重视“软件开发人员（Dev）”和“IT 运维技术人员（Ops）”之间沟通合作的文化、运动或惯例。透过自动化“软件交付”和“架构变更”的流程，来使得构建、测试、发布软件能够更加地快捷、频繁和可靠。
它的出现是由于软件行业日益清晰地认识到：为了按时交付软件产品和服务，开发和运营工作必须紧密合作。
从定义来看，其实 devops 就是为了让开发、运维和 QA 可以高效协作的流程。（可以把 DevOps 看作开发、技术运营和质量保障（QA）三者的交集。）
DevOps

DevOps 对应用程序发布的影响
在很多企业中，应用程序发布是一项涉及多个团队、压力很大、风险很高的活动。然而在具备 DevOps 能力的组织中，应用程序发布的风险很低，原因如下 [2] ：
（1）减少变更范围
与传统的瀑布模式模型相比，采用敏捷或迭代式开发意味着更频繁的发布、每次发布包含的变化更少。由于部署经常进行，因此每次部署不会对生产系统造成巨大影响，应用程序会以平滑的速率逐渐生长。
（2）加强发布协调
靠强有力的发布协调人来弥合开发与运营之间的技能鸿沟和沟通鸿沟；采用电子数据表、电话会议和企业门户（wiki、sharepoint）等协作工具来确保所有相关人员理解变更的内容并全力合作。
（3）自动化
强大的部署自动化手段确保部署任务的可重复性、减少部署出错的可能性。

为什么大公司一定要使用 DevOps?

0 DevOps 的意图

究竟什么是 DevOps? 要想回答这个问题，首先要明确 DevOps 这个过程参与的人员是谁？即开发团队和 IT 运维团队！那么，DevOps 的意图是什么呢？即在两个团队之间，建立良好的沟通和协作，更快更可靠的创建高质量软件！

事实上，并不是这两个团队之间的协作帮助交付了更好的软件，而是“开发”和“运维”团队之间的统一导致了软件的改进，并以更快的速度交付。我们不要忘记 DevOps 工具在实现自动化方面所扮演的角色。

开发和运维“一体”的感觉是由开发人员和操作工程师之间的技能组合和实践的桥接以及自动化（DevOps）工具的实现引起的。 世界各地的大型互联网公司已采用 DevOps 方法来彻底改进其性能，安全性和团队动态。

在本篇文章中，让我们看看什么是 DevOps，为什么它如此重要！ 我们将首先跟踪导致 DevOps 的软件开发方法的演变，然后探索什么是 DevOps 及其生命周期，并通过评估世界顶级公司，来看看如何使用 DevOps 来获得益处。

1 软件开发的演变

多年来，DevOps 从现有的软件开发策略/方法发展而来，以响应业务需求。让我们简要地看一下这些模型是如何演变的，以及它们最适合的场景。

缓慢而繁琐的瀑布模型演变成敏捷，开发团队在短时间内完成软件开发，持续时间甚至不超过两周。如此短的发布周期帮助开发团队处理客户反馈，并将其与 bug 修复一起合并到下一个版本中。

虽然这种敏捷的 SCRUM 方法为开发带来了敏捷性，但它在运维方面却失去了敏捷实践的速度。开发人员和运维工程师之间缺乏协作仍然会减慢开发过程和发布。

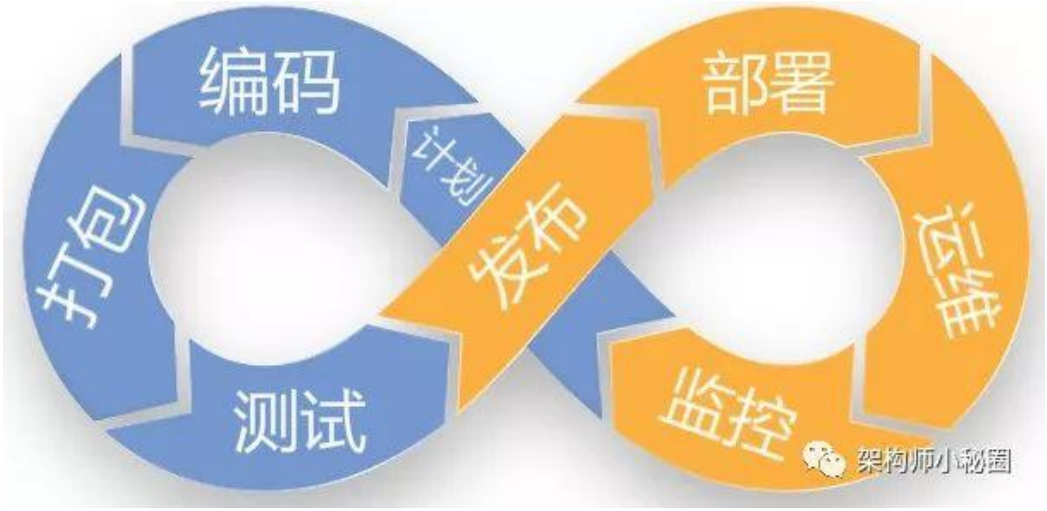
DevOps 方法就是基于对更好的协作和更快的交付的需求而产生的。DevOps 允许用较少复杂问题的持续软件交付来修复和更快地解决问题。

现在我们已经了解了 DevOps 的发展，让我们来详细看看 DevOps 是什么。

2 什么是 DevOps?

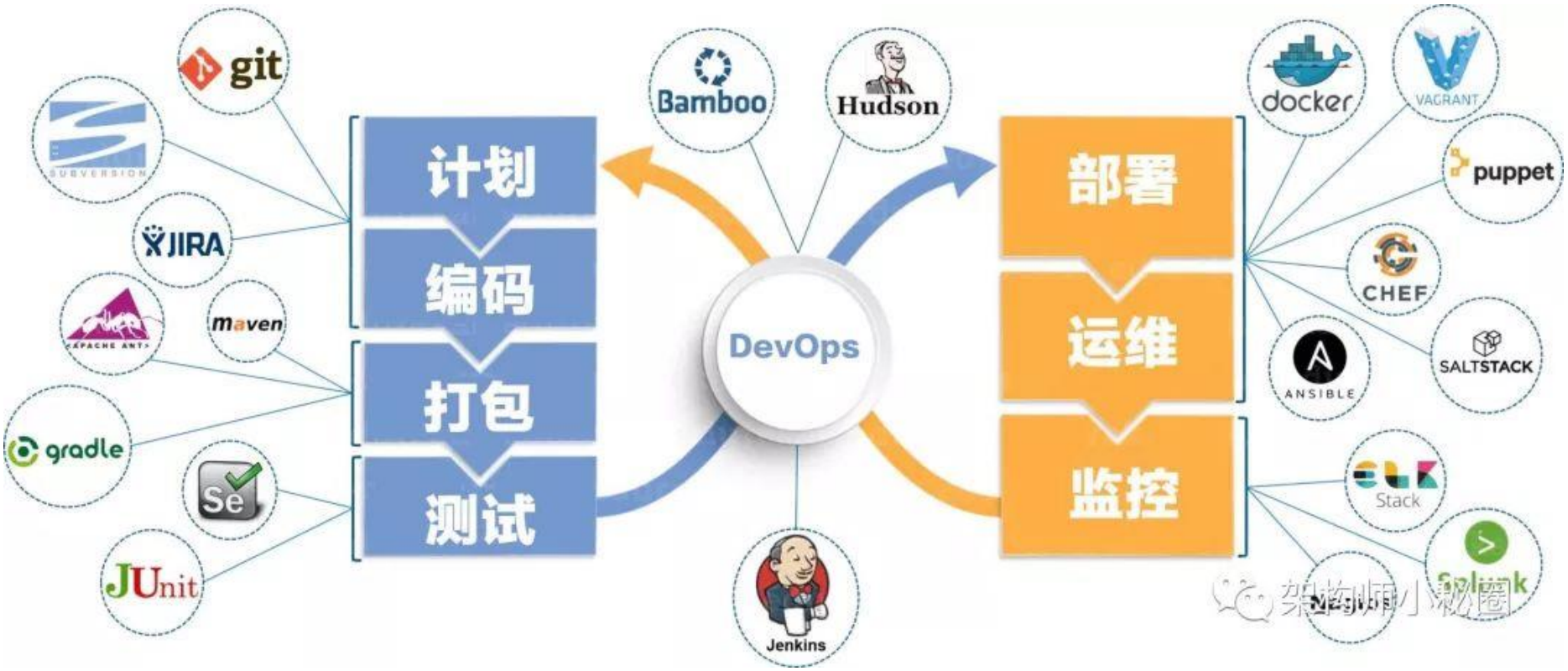
DevOps 是一种软件开发方法，涉及软件在整个开发生命周期中的持续开发，持续测试，持续集成，持续部署和持续监控。 这些活动只能在 DevOps 中实现，而不是敏捷或瀑布，这就是为什么顶级互联网公司选择 DevOps 作为其业务目标的前进方向。 DevOps 是在较短的开发周期内开发高质量软件的首选方法，可以提高客户满意度。

在不了解 DevOps 生命周期的情况下，对 DevOps 的理解也会片面化。现在让我们看看 DevOps 生命周期，并探讨它们如何与下图所示的软件开发阶段相关联。



持续开发：
这是 DevOps 生命周期中软件不断开发的阶段。与瀑布模型不同的是，软件可交付成果被分解为短开发周期的多个任务节点，在很短的时间内开发并交付。
这个阶段包括编码和构建阶段，并使用 Git 和 SVN 等工具来维护不同版本的代码，以及 Ant、Maven、Gradle 等工具来构建/打包代码到可执行文件中，这些文件可以转发给自动化测试系统进行测试。
持续测试：
在这个阶段，开发的软件将被持续地测试 bug。对于持续测试，使用自动化测试工具，如 Selenium、TestNG、JUnit 等。这些工具允许质量管理系统完全并行地测试多个代码库，以确保功能中没有缺陷。在这个阶段，使用 Docker 容器实时模拟"测试环境"也是首选。一旦代码测试通过，它就会不断地与现有代码集成。
持续集成：
这是支持新功能的代码与现有代码集成的阶段。由于软件在不断地开发，更新后的代码需要不断地集成，并顺利地与系统集成，以反映对最终用户的需求更改。更改后的代码，还应该确保运行时环境中没有错误，允许我们测试更改并检查它如何与其他更改发生反应。
Jenkins 是一个非常流行的用于持续集成的工具。使用 Jenkins，可以从 git 存储库提取最新的代码修订，并生成一个构建，最终可以部署到测试或生产服务器。可以将其设置为在 git 存储库中发生更改时自动触发新构建，也可以在单击按钮时手动触发。
持续部署：
它是将代码部署到生产环境的阶段。 在这里，我们确保在所有服务器上正确部署代码。 如果添加了任何功能或引入了新功能，那么应该准备好迎接更多的网站流量。 因此，系统运维人员还有责任扩展服务器以容纳更多用户。
由于新代码是连续部署的，因此配置管理工具可以快速，频繁地执行任务。 Puppet，Chef，SaltStack 和 Ansible 是这个阶段使用的一些流行工具。
容器化工具在部署阶段也发挥着重要作用。 Docker 和 Vagrant 是流行的工具，有助于在开发，测试，登台和生产环境中实现一致性。除此之外，它们还有助于轻松扩展和缩小实例。
持续监控：

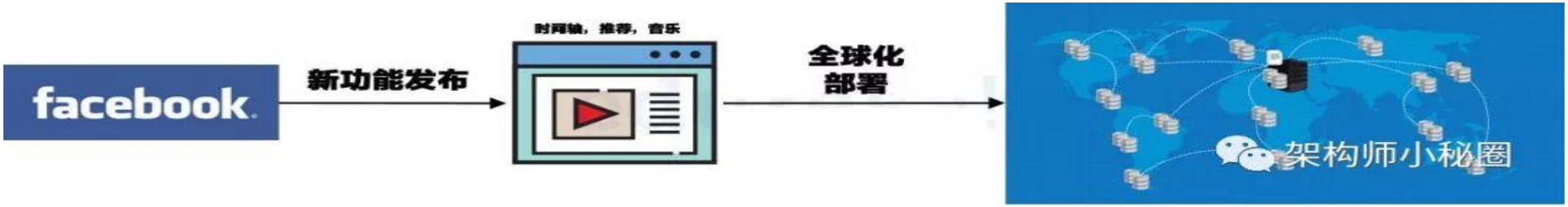
这是 DevOps 生命周期中非常关键的阶段，旨在通过监控软件的性能来提高软件的质量。这种做法涉及运营团队的参与，他们将监视用户活动中的错误/系统的任何不正当行为。这也可以通过使用专用监控工具来实现，该工具将持续监控应用程序性能并突出问题。
使用的一些流行工具是 Splunk，ELK Stack，Nagios，NewRelic 和 Sensu。这些工具可帮助密切监视应用程序和服务器，以主动检查系统的运行状况。它们还可以提高生产率并提高系统的可靠性，从而降低 IT 支持成本。发现的任何重大问题都可以向开发团队报告，以便可以在持续开发阶段进行修复。
这些 DevOps 阶段连续循环进行，直到达到所需的产品质量。下面的图表将显示可以在 DevOps 生命周期的哪个阶段使用哪些工具。



既然我们已经确定了 DevOps 的重要性，并且了解了它的不同阶段以及所涉及的 DevOps 工具，现在让我们看看 Facebook 的一个案例研究，并理解为什么他们从敏捷转向 DevOps。我们将采用 Facebook 曾推出的新特性的用例，这些新特性导致 Facebook 重新评估其产品交付并采用 DevOps 方法。

3 DevOps 案例研究

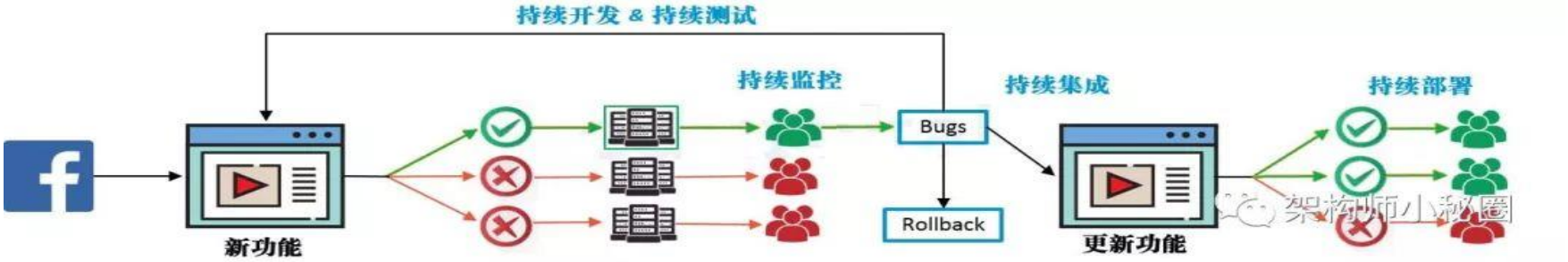
曾经，Facebook 向遍布全球的若干亿用户推出了一系列新功能 - 时间轴，推荐和音乐功能。 发布后 Facebook 上产生的巨大流量导致服务器崩溃。 推出的功能获得了用户的大规模超常规响应，这导致了新功能产生了不可控的结果，使他们没有预料到。



这导致了 Facebook 重新评估和战略调整，从而使 Facebook 推出了暗启动技术。 使用 DevOps 原则，Facebook 为其新版本的发布创建了以下方法。

Facebook 暗启动技术

暗启动是在新功能完全发布给所有用户之前，逐步将新功能，推广到选定的一组用户的过程。 这允许开发团队尽早获得用户反馈，测试错误，并且还可以测试基础架构性能。 这种发布方法是持续交付的直接结果，有助于实现更快，更迭代的版本，确保应用程序性能不会受到影响，并且用户可以很好地更新该版本。



在暗启动技术中，新功能通过专用的部署管道发布给小型用户群。 在上面给出的 Facebook 暗启动图表中，您可以看到只打开了一个部署管道，将新功能部署到一组选定用户。 此时剩余的数百条管道全部关闭。
持续监视部署功能的特定用户群，以收集反馈并识别错误。 这些错误和反馈将被纳入开发，测试和部署在同一用户群中，直到功能变得稳定。 一旦实现稳定性，通过启用其他部署管道，将逐步在其他用户群上部署这些功能。
Facebook 通过将代码包装在功能标记或功能切换中来实现此目的，该切换用于控制谁可以看到新功能以及何时查看。与此同时，模拟向用户启动代码的全部效果，在向用户开放全部功能之前，可以及早的暴露应用程序基础架构的痛点和区域，功能稳定后，将通过多个版本将其部署到其余用户。通过这种方式，Facebook 拥有一个受控或稳定的机制，可以为其庞大的用户群开发新功能。相反，如果功能没有得到很好的响应，他们可以选择完全回滚部署。这也帮助他们为部署准备服务器，因为他们可以预测网站上的用户活动，并相应地扩展服务器。上面给出的图表描述了 Facebook 的暗启动过程。

4 总结

微信，淘宝，以及许多领先的科技巨头，在向所有人发布之前，都使用暗发布逐渐向一小部分用户发布和测试新功能。
DevOps 的目的是更快速，更可靠地创建质量更好的软件，同时开发，运维团队之间进行更多的沟通和协作。 它是一个自动化过程，允许快速，安全和高质量的软件开发和发布，同时保持所有利益相关者在一个循环中。 这就是 DevOps 获得越来越多的大型互联网公司青睐的真正原因。

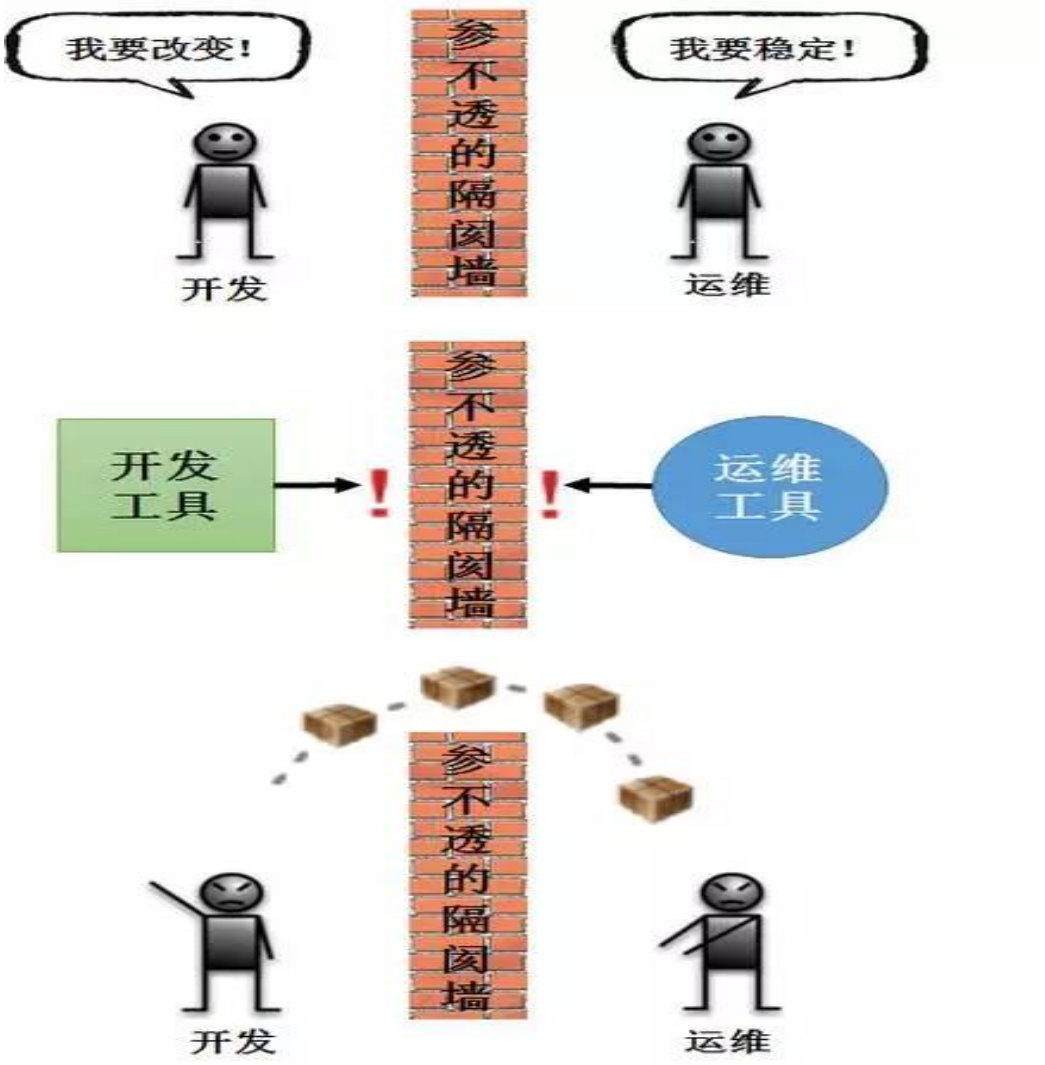
DevOps 简介

简介

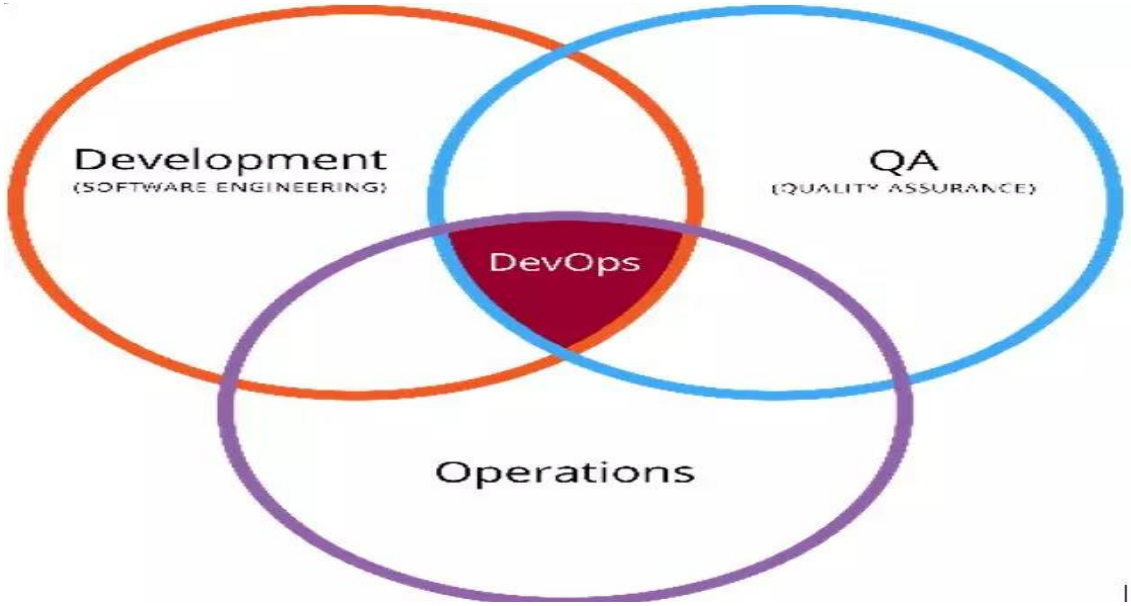
DevOps 是一个完整的面向 IT 运维的工作流，以 IT 自动化以及持续集成（CI）、持续部署（CD）为基础，来优化程式开发、测试、系统运维等所有环节。

DevOps 的概念

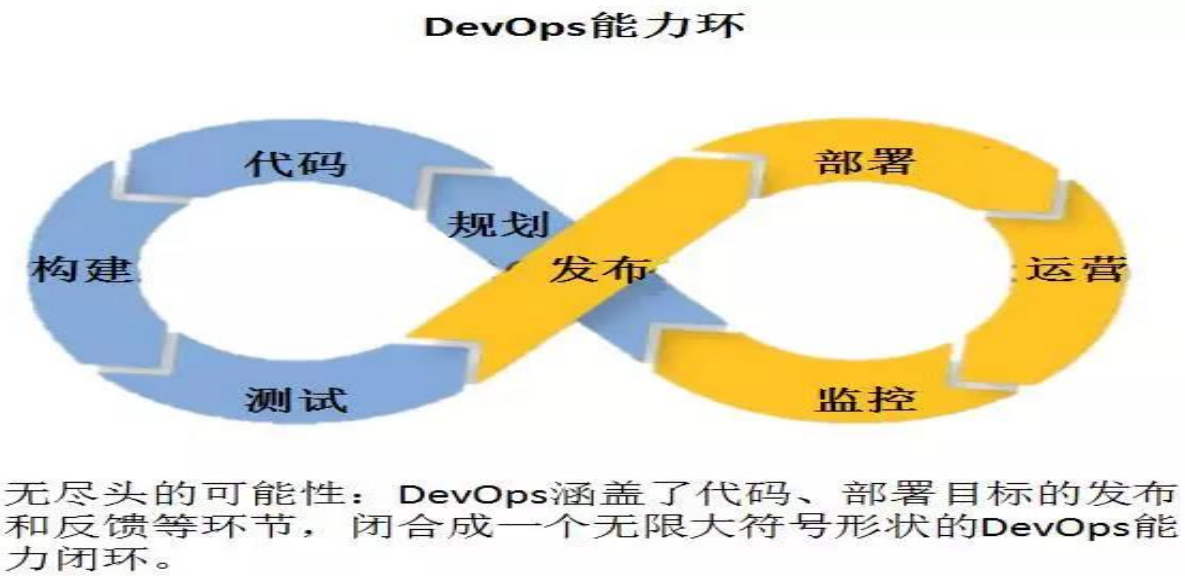
DevOps 一词的来自于 Development 和 Operations 的组合，突出重视软件开发人员和运维人员的沟通合作，通过自动化流程来使得软件构建、测试、发布更加快捷、频繁和可靠。



DevOps 是为了填补开发端和运维端之间的信息鸿沟，改善团队之间的协作关系。不过需要澄清的一点是，从开发到运维，中间还有测试环节。**DevOps 其实包含了三个部分：开发、测试和运维。**



换句话说，DevOps 希望做到的是软件产品交付过程中 IT 工具链的打通，使得各个团队减少时间损耗，更加高效地协同工作。专家们总结出了下面这个 DevOps 能力图，良好的闭环可以大大增加整体的产出。



历史变革

由上所述，相信大家对 DevOps 有了一定的了解。但是除了触及工具链之外，作为文化和技术的方法论，DevOps 还需要公司在组织文化上的变革。回顾软件行业的研发模式，可以发现大致有三个阶段：瀑布式开发、敏捷开发、DevOps。

DevOps 早在九年前就有人提出来，但是，为什么这两年才开始受到越来越多的企业重视和实践呢？因为 DevOps 的发展是独木不成林的，现在有越来越多的技术支撑。微服务架构理念、容器技术使得 DevOps 的实施变得更加容易，计算能力提升和云环境的发展使得快速开发的产品可以立刻获得更广泛的使用。

好处是什么?

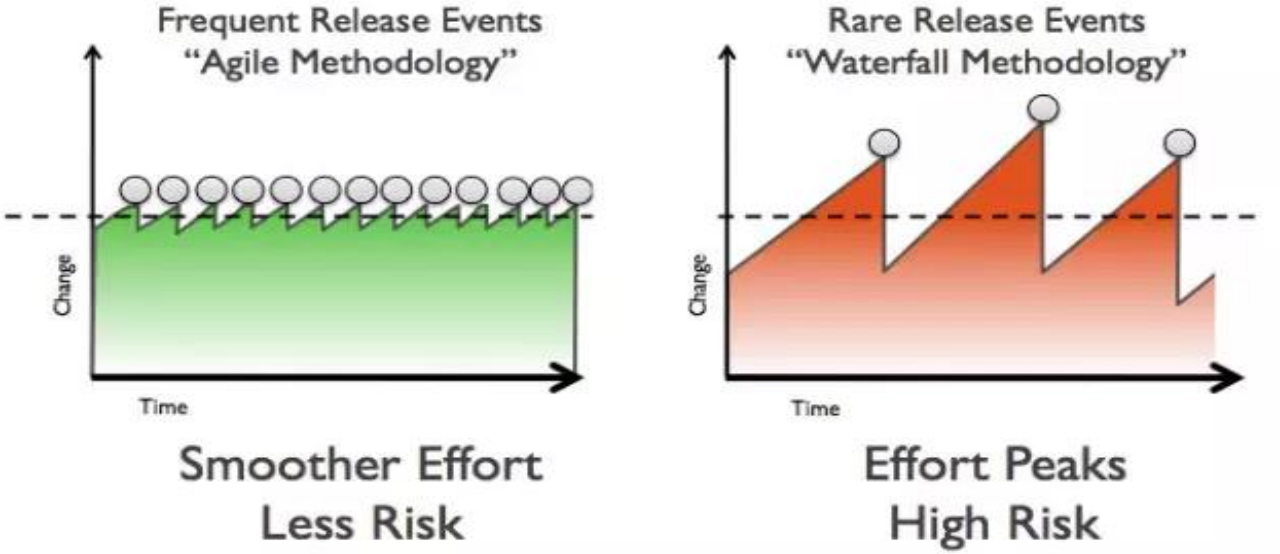
DevOps 的一个巨大好处就是可以高效交付，这也正好是它的初衷。Puppet 和 DevOps Research and Assessment (DORA) 主办了 2016 年 DevOps 调查报告，根据全球 4600 位各 IT 公司的技术工作者的提交数据统计，得出高效公司平均每年可以完成 1460 次部署。

与低效组织相比，高效组织的部署频繁 200 倍，产品投入使用速度快 2555 倍，服务恢复速度快 24 倍。在工作内容的时间分配上，低效者要多花 22%的时间用在为规划好或者重复工作上，而高效者却可以多花 29%的时间用在新的工作上。所以这里的高效不仅仅指公司产出的效率提高，还指员工的工作质量得到提升。

DevOps 另外一个好处就是会改善公司组织文化、提高员工的参与感。员工们变得更高效，也更有满足和成就感；调查显示高效员工的雇员净推荐值（eNPS:employee Net Promoter Score）更高，即对公司更加认同。

快速部署同时提高 IT 稳定性。这难道不矛盾吗?

快速的部署其实可以帮助更快地发现问题，产品被更快地交付到用户手中，团队可以更快地得到用户的反馈，从而进行更快地响应。而且，DevOps 小步快跑的形式带来的变化是比较小的，出现问题的偏差每次都不会太大，修复起来也会相对容易一些。



因此，认为速度就意味着危险是一种偏见。此外，滞后软件服务的发布也并不一定会完全地避免问题，在竞争日益激烈的 IT 行业，这反而可能错失了软件的发布时机

为什么 DevOps 会兴起?

为什么会继续火下去?

条件成熟：技术配套发展

技术的发展使得 DevOps 有了更多的配合。早期时，大家虽然意识到了这个问题的，但是苦于当时没有完善丰富的技术工具，是一种“理想很丰满，但是现实很骨感”的情况。DevOps 的实现可以基于新兴的容器技术；也可以在自动化运维工具 Puppet、SaltStack、Ansible 之后的延伸；还可以构建在传统的 Cloud Foundry、OpenShift 等 PaaS 厂商之上。

来自市场的外部需求：这世界变化太快

IT 行业已经越来越与市场的经济发展紧密挂钩，专家们认为 IT 将会有支持中心变成利润驱动中心。事实上，这个变化已经开始了，这不仅体现在 Google、苹果这些大企业中，而且也发生在传统行业中，比如出租车业务中的 Uber、酒店连锁行业中的 Airbnb、图书经销商 Amazon 等等。能否让公司的 IT 配套方案及时跟上市场需求的步伐，在今天显得至关重要。

DevOps 2016 年度报告给出了一个运维成本的计算公式：

停机费用成本 = 部署频率 * 版本迭代失败概率 * 平均修复时间 * 断电的金钱损失

来自团队的内在动力：工程师也需要

对于工程师而言，他们也是 DevOps 的受益者。微软资深工程师 Scott Hanselman 说过“对于开发者而言，最有力的工具就是自动化工具”（The most powerful tool we have as developers is automation）。

工具链的打通使得开发者们在交付软件时可以完成生产环境的构建、测试和运行；正如 Amazon 的 VP 兼 CTO Werner Vogels 那句让人印象深刻的话：“谁开发谁运行”。（You build it, you run it)

实现 DevOps 需要什么?

硬性要求：工具上的准备

上文提到了工具链的打通，那么工具自然就需要做好准备。现将工具类型及对应的不完全列举整理如下：

- 代码管理（SCM）：**GitHub**、GitLab、BitBucket、SubVersion
- 构建工具：**Ant**、Gradle、**maven**
- 自动部署：Capistrano、CodeDeploy
- 持续集成（CI）：Bamboo、Hudson、Jenkins
- 配置管理：Ansible、Chef、Puppet、SaltStack、ScriptRock GuardRail
- 容器：**Docker**、LXC、第三方厂商如 AWS
- 编排：Kubernetes、Core、Apache Mesos、DC/OS
- 服务注册与发现：**Zookeeper**、etcd、Consul
- 脚本语言：python、ruby、shell

- 日志管理：ELK、Logentries
- 系统监控：Datadog、Graphite、Icinga、Nagios
- 性能监控：AppDynamics、New Relic、Splunk
- 压力测试：JMeter、Blaze Meter、loader.io
- 预警：PagerDuty、pingdom、厂商自带如 AWS SNS
- HTTP 加速器：Varnish
- 消息总线：ActiveMQ、SQS
- 应用服务器：Tomcat、JBoss
- Web 服务器：Apache、Nginx、IIS
- 数据库：MySQL、Oracle、PostgreSQL 等关系型数据库；cassandra、mongoDB、redis 等 NoSQL 数据库
- 项目管理（PM）：Jira、Asana、Taiga、Trello、Basecamp、Pivotal Tracker

在工具的选择上，需要结合公司业务需求和技术团队情况而定。（注：更多关于工具的详细介绍可以参见此文：51 Best DevOps Tools for #DevOps Engineers）

软性需求：文化和人

DevOps 成功与否，公司组织是否利于协作是关键。开发人员和运维人员可以良好沟通互相学习，从而拥有高生产力。并且协作也存在于业务人员与开发人员之间。

出席了 2016 年伦敦企业级 DevOps 峰会的 ITV 公司在 2012 年就开始落地 DevOps，其通用平台主管 Clark 在接受了 InfoQ 的采访，在谈及成功时表示，业务人员非常清楚他们希望在最小化可行产品中实现什么，工程师们就按需交付，不做多余工作。

这样，工程师们使用通用的平台（即打通的工具链）得到更好的一致性和更高的质量。此外，DevOps 对工程师个人的要求也提高了，很多专家也认为招募到优秀的人才也是一个挑战。

DevOps 的采用现状

哪些公司在用?

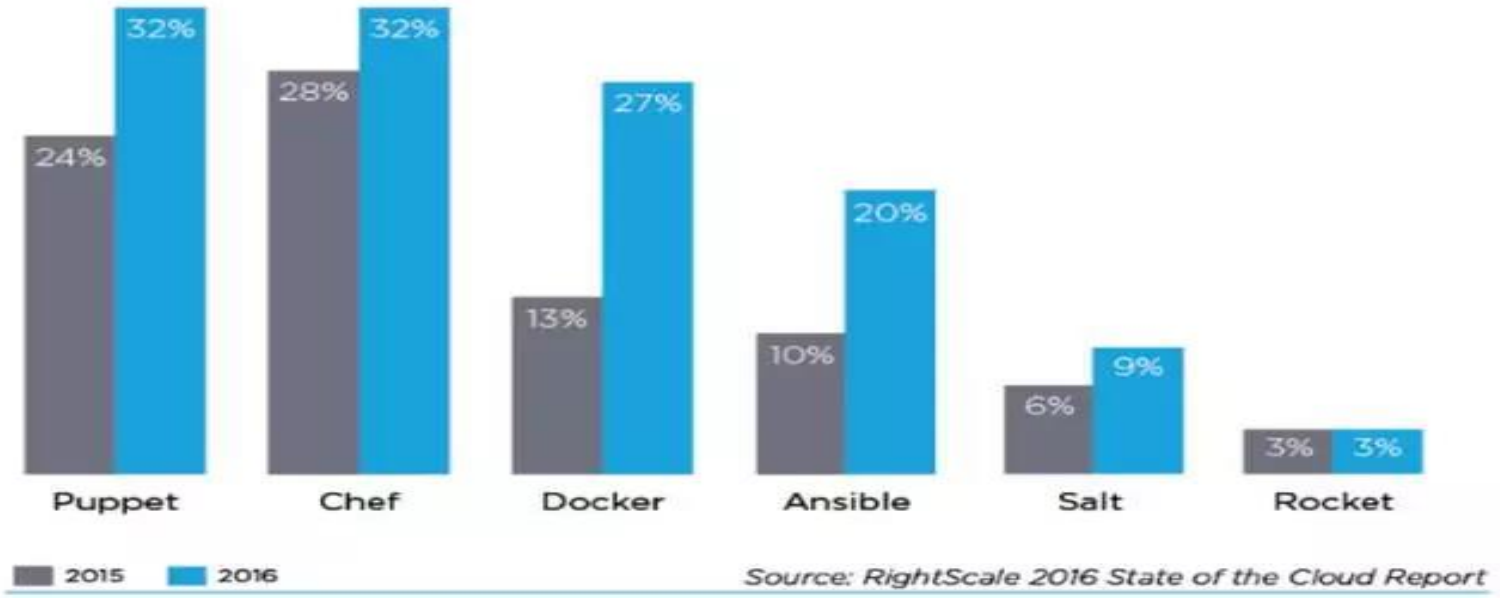
DevOps 正在增长，尤其是在大企业中：调查发现，DevOps 的接受度有了显著提高。74%的受访者已经接受了 DevOps，而去年这一比例为 66%。目前，在 81%的大企业开始接受 DevOps，中小企业的接受度仅为 70%。

那么具体而言都有些公司在采用 DevOps 呢？Adobe、Amazon、Apple、Airbnb、Ebay、Etsy、Facebook、LinkedIn、Netflix、NASA、Starbucks、Target（泛欧实时全额自动清算系统）、Walmart、Sony 等等。

他们怎么实施的?

首先，大企业正在自下而上接受 DevOps，其中业务单位或部门（31%）以及项目和团队（29%）已经实施 DevOps。不过，只有 21%的大企业在整个公司范围内采用了 DevOps。

其次，在工具层面上，DevOps 工具的用量大幅激增。Chef 和 Puppet 依然是最常用的 DevOps 工具，使用率均为 32%。Docker 是年增长率最快的工具，用量增长一倍以上。Ansible 的用量也有显著增加，使用率从 10%翻倍至 20%。



5 大 DevOps 工具，你用过几个?

DevOps 的概念在软件开发行业中逐渐流行起来。越来越多的团队希望实现产品的敏捷开发，DevOps 使一切成为可能。有了 DevOps ，团队可以定期发布代码、自动化部署、并将持续集成 / 持续交付作为发布过程的一部分。

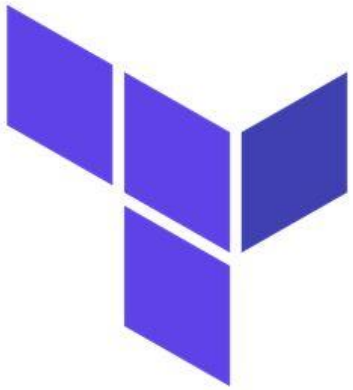
虽然 DevOps 背后有各种各样的概念，但幸好有一些工具可以让你更容易地理解和实现。在本文中，你将了解这些工具，并将它们作为软件发布 / 维护工具包工作的一部分开始使用。

DevOps 有很多可使用的工具，在一篇文章中几乎不可能介绍完它们。**本文将介绍五种最流行、功能最强大的 DevOps 工具：**

- Terraform
- Ansible

- Packer
- Docker
- Kubernetes

Terraform



类型：配置
语言：Go
推荐的第一个 DevOps 工具是来自 Hashicorp 的**Terraform**。Terraform 是一个基础设施管理工具，允许您正确地构建、更改和管理基础设施。您可以将 Terraform 视为一种供应工具。它帮助您设置服务器、数据库和其他支持全面应用程序的基础设施。
Terraform 并不局限于任何特定的云服务提供商，它可以与多个云提供商和环境协同工作。云服务提供商如 AWS、Microsoft Azure、谷歌云都与 Terraform 无缝集成。版本控制系统托管提供商，如 Github 和 Bitbucket，都可以很好地使用它。
Terraform 有一个企业版和开源版，还可以安装在 macOS、Linux 和 Windows 系统上。

Ansible

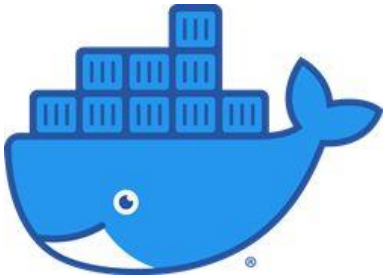
类型：配置
语言：Python、PowerShell、Shell 和 Ruby
与 Terraform 类似，Ansible 也是一个基础设施管理工具。Ansible 可以帮助你部署应用程序，供应和配置管理的服务器。Ansible 是用 Python 构建的，由 RedHat 维护，但它仍然是免费和开源的。
作为一个配置管理系统，您可以使用 Ansible 来设置和构建多个服务器。你可以在控制机器上安装 Ansible，而不需要 Ansible 在其他服务器上运行，这些服务器可以从 web 到应用程序再到数据库服务器。
与 Terraform 不同，Ansible 不使用 HCL 作为它的代码。相反，配置是写在 Ansible 剧本，这是 YAML 文件。Ansible 使用声明性和程序性模式的混合。这与 Terraform 不同，后者仅仅是声明性的。
Linux 是安装 Ansible 最合适的操作系统。不过，它在 macOS 上也运行良好。对于 Windows 用户，可以通过 Linux 的 Windows 子系统的 bash shell 使用 Ansible。

Packer

类型：配置
语言: Go
Packer 是另一个来自 Hashicorp 的 DevOps 工具。用 Golang 编写的 Packer 可以帮助你自动创建虚拟镜像。手动构建镜像的过程可能令人沮丧，因为它容易出错，但 Packer 消除了所有这些。

对于单个 JSON 文件，可以使用 Packer 创建多个镜像。当它第一次工作时，由于没有任何东西会干扰它的自动化过程，因此可以保证它能百分百地工作。许多云服务提供商都使用镜像，您可以无缝地与这些提供商合作，因为 Packer 标准化了用于云环境的镜像的创建。
Packer 不是一个独立的工具。您可以将其与 Ansible、Chef 和 Jenkins 集成，以便在部署管道中进一步使用这些映像。安装过程并不复杂，您可以学习如何开始使用该工具。

Docker



类型：容器
语言：Go
Docker 是一种容器技术，可让您在特殊环境中隔离应用程序。容器化与虚拟化类似，不同之处在于容器不会启动完整的操作系统。
使用 Docker 容器，您可以在这些自定义环境中开发和部署应用程序，从而不必担心兼容性问题。您的应用程序可以在任何位置运行，只要它们位于容器中即可。
要启动 Docker 容器，您必须通过 Dockerfiles 创建 Docker 映像。从 Docker 映像启动时，Dockerfile 包含 Docker 容器中所需的规范。您不必总是构建自己的 Docker 映像，因为 Docker Hub 上可以使用官方映像。
Docker 本身可以在 Linux 上运行，并且在 macOS 上也可以正常运行，因为它类似于 Unix。对于 Windows 用户，也可以通过 Docker 工具箱来使用 Docker。
通常情况下，你可能会使用多个 Docker 容器，这就引出了本文中的最后一个 DevOps 工具。

Kubernetes

类型：容器
语言：Go
Kubernetes（K8s）是一个谷歌开源工具，它可以让你管理 Docker 容器。由于在生产中经常有大量的容器在运行，因此，Kubernetes 使编排这些容器成为可能。
首先要了解编排 Docker 容器的原因。当有许多容器在运行时，很难手动监视这些容器，并使它们彼此通信；另外，这种扩展以及负载平衡也变得困难。
使用 Kubernetes，可以控制所有这些容器，因此可以将这组机器作为一台机器进行管理。与 Docker Compose 相比，Kubernetes 是不同的，因为它使部署，扩展和监视容器变得更加容易。当它们中的任何一个崩溃时，它们都可以自愈，而 Kubernetes 可以制造新的来代替。使用 K8s，可以轻松地进行存储编排、服务发现和负载平衡。
您可以在 macOS，Linux 和 Windows 上安装 Kubernetes，并通过Kubernetes 命令行工具使用它。

结论：

DevOps 的概念对于使大型应用程序在不同负载或流量下保持高性能是非常有益的。它还使软件部署管道易于管理。
如果没有可用的工具，DevOps 概念很难实现。这个领域有很多工具，每个公司都有不同的选择。尽管如此，Terraform、Ansible、Packer、Docker、Kubernetes 都是拥有大量用户社区的工具，能够在各种软件项目中实现 DevOps 的工具。如果您打算使用它们，可以做进一步的研究。