DevOps

背景

20 世纪 80 年代，软件开发主要遵循瀑布式模型1，软件开发生命周期被清晰的划分为指定的若干阶段，其带来的好处非常明显，即明确目标，按序执行，但同时也存在致命的缺点，例如在软件开发生命周期后期阶段发现了严重问题，那么排错过程无疑是非常耗时的。为解决瀑布模式中遇到的问题，2001 年，十七位敏捷开发的发起者和实践者聚集发表了 “敏捷软件开发宣言”，其中包含了敏捷开发的十二项原则2，简而言之，敏捷开发是以人为核心、迭代、循序渐进的开发方式，遵循在短周期内持续测试和交付软件，其相比于瀑布式模型有着截然不同的理念，实现层面而言，敏捷开发习惯于将软件项目需求拆分为多个迭代项目，并在每个迭代项目上完成开发、测试、部署、交付，从而具备快速响应变化、可集成化、可运行及快速交付等特点，但敏捷开发也暴露了一些不足，例如迭代式开发造成的开发成本高问题，再如敏捷开发遵循的 “软件运行优先级高于详尽的文档” 这一原则，将会在团队协作中增添不必要的繁琐沟通环节。由于敏捷开发注重软件开发阶段的同时未兼顾运维阶段，因而易造成开发和运维间协作效率低的问题，为解决此问题，DevOps 的理念应用而生，DevOps 采用容器技术、容器编排、微服务、持续集成和交付软件等促进了开发、运维、测试之间的高效协同，为软件开发生命周期揭开了新的篇章。

DevOps 介绍

DevOps 方法论

DevOps 全称为 Development & Operations，其代表的并非一种具体实现技术，而是一种方法论，并在 2009 年被提出3。DevOps 的出现最终目是为了打破开发人员与运维人员之间的壁垒和鸿沟，高效的组织团队通过自动化工具相互协作以完成软件生命周期管理，从而更快且频繁地交付高质量稳定的软件。

---

1 https://en.wikipedia.org/wiki/Waterfall_model
2 https://en.wikipedia.org/wiki/Agile_software_development
3 https://en.wikipedia.org/wiki/DevOps

图 为 DevOps 闭环流程图，左边代表开发流程，可以看到其中包括应用需求（plan）、编码实现（code）、应用构建（build）、应用测试（test）。而图 1 右边则代表运维流程，其中包括应用版本发布（release）、应用部署（deploy）、应用操作（operate）和应用监控（moniter），DevOps 将以上流程形成闭环从而实现开发运维的无缝连接。

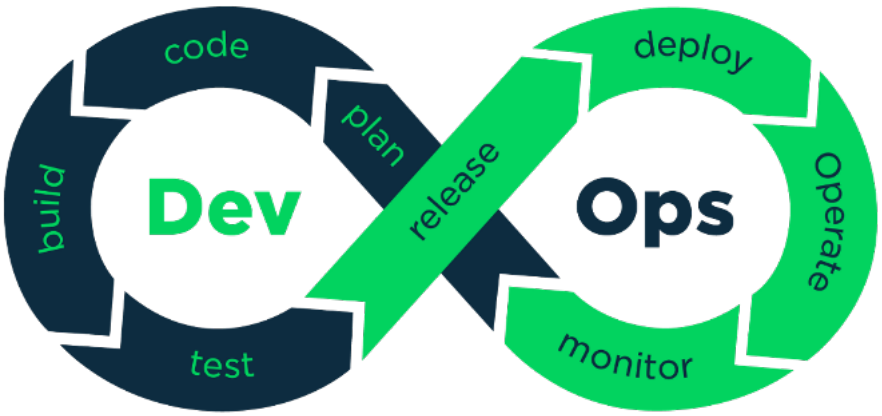

图 1 DevOps 闭环流程图

在 DevOps 生态发展的 10 多年里，许多企业完成了 DevOps 相应落地，从而极大增加了企业应用开发运营效率，为客户和用户交付最大化价值及高质量成果。当然，这段期间内，DevOps 生态中也囊括了许多开源及商业软件，如图 所示。

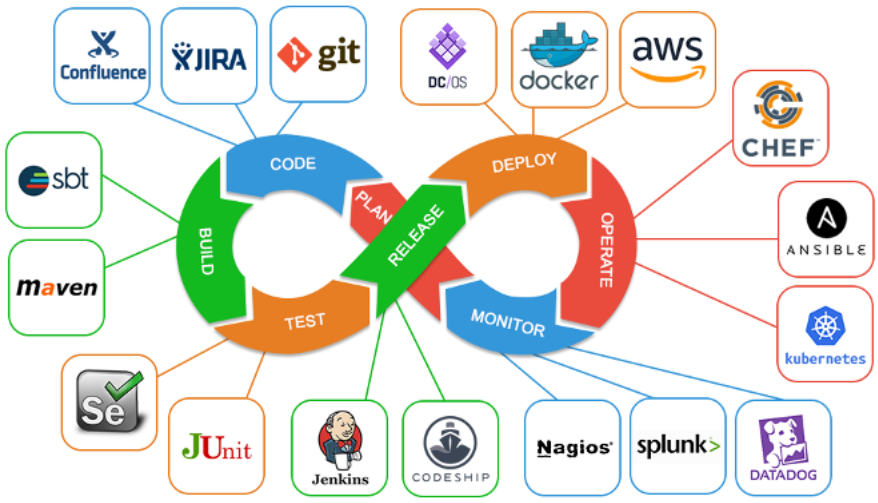

图 2 图 DevOps 生态图

DevOps 与云原生

云原生倡导敏捷、容错、自动化的特点使得 DevOps 成为云原生基础不可或缺的一环，其原因不外乎于以下几点：

1. 云原生提供 DevOps 基础设施

容器与容器编排技术提供了云原生的标准运行环境及基础架构。DevOps 的核心点在于软件的持续集成、持续交付，而容器作为云原生应用的标准发布，促进了 DevOps 在云原生环境下的流行，与此同时，基于容器的 PaaS 平台，例如 Kubernetes，可进一步为 DevOps 落地提供土壤。

2. 微服务架构加速应用 DevOps

微服务架构实现了云原生应用固有的特点，即无状态性、弹性扩展、高内聚、低耦合。在此架构下，试想在生产环境中，由于一个庞大的应用将被拆分为几十上百个服务，每个服务的开发、构建、部署过程必然遵循快速发布的原则，因而在敏捷性、自动化工具链上对流程提出了较高要求，在此基础上，DevOps 自动化、协作、敏捷的文化将会在很大程度上加速微服务的开发效率、降低沟通成本、提升部署速率。

3. DevOps 赋能服务网格

服务网格是一套微服务治理框架，主要实现各个微服务间的网络通信，虽然服务网格技术本身和 DevOps 关系不大，但由于其是建立在微服务架构下，因而也需与 DevOps 进行相应融合才能实现微服务的持续集成交付。

4. DevOps 加速 Serverless 应用迁移

Serverless 为云原生应用的最终形态，即服务端托管云厂商，开发者只需维护好一段函数代码即可，这一新型云计算模式背后秉承的理念实际与 DevOps 是相互契合的，DevOps 遵循消除开发者与运维人员之间的壁垒，而 Serverless 架构责任划分原则使得开发人员和运维人员不再有界限。

此外，Serverless 应用较微服务有更快地交付频率，随着开发团队生产力的不断提升，面对开发者开发的大量函数，或大量函数组成的应用时，如何成功的将其迁移至云厂商取决于 DevOps，尤其在早期采用阶段。

DevOps 应用现状

云原生技术实现了应用的敏捷开发，不仅很大程度上提升了交付速度、降低了业务的试错成本、还能够迅速响应用户需求、加速业务创新。DevOps 理念持续在用户侧强化，随着 CI/CD 工具链的不断成熟，应用的迭代效率呈指数增长[i]。

2020 年云原生产业联盟发布了《中国云原生用户调查报告》，在 DevOps 应用现状方面，该报告针对“应用及软件发布周期和方式”及“容器技术使用现状”两部分进行了统计说明。根据调研，主要有以下结论：

首先，用户应用及软件发布近年来整体趋于高频。有近 6%的用户每日发布应用，28%的用户每周发布应用，27%的用户每月发布应用，此外还有 39%的用户不定时发布应用。

其次，用户应用及软件的发布方式近年来趋于向自动化方式转变。调查显示已超过 25%的用户实现了自动化发布应用， 55%的用户采用自动化与手动相结合的发布方式，选择手动发布应用的用户仅为 20%[i]。

最后，根据对包括互联网、金融、制造、服务业、政府、电信、能源、医疗、化工等行业的用户调研，在容器技术使用现状中分析指出， 56%的用户将容器技术应用于 DevOps 自动化运维的构建[i]，如图 所示。另外，近七成用户使用 Jenkins 作为容器的 CI/CD 工具。在本次调研的用户中，69%的用户使用 Jenkins 作为容器的 CI/CD 工具，24%的用户选用 Spinnaker，14%的用户选用 Drone，选用 Prow、Flux、Tekton Pipelines 技术的用户占比分别为 10%、10%、9%，还有 2%的用户选用商业 CI 产品。如图 所示。

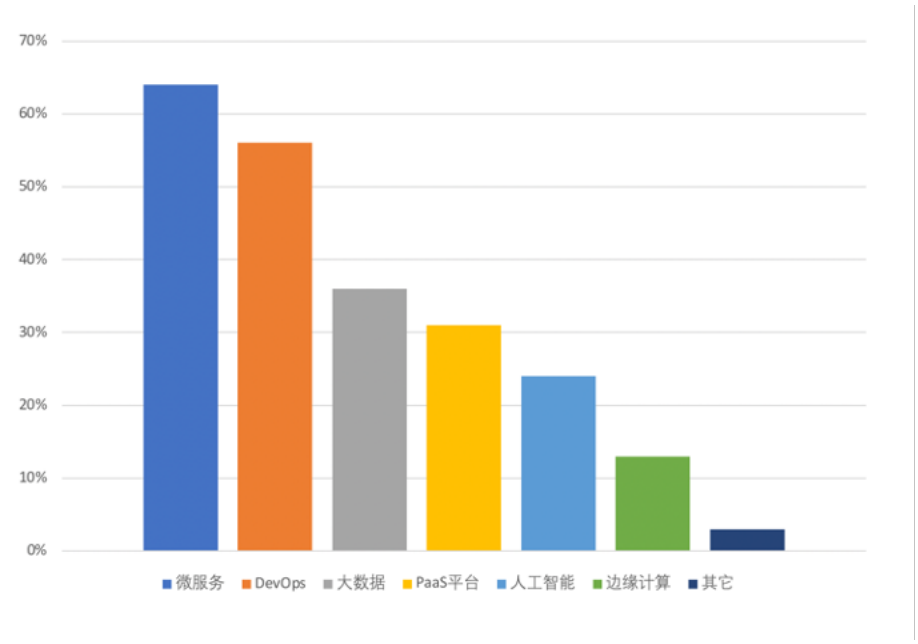

图 3 容器技术主要使用场景

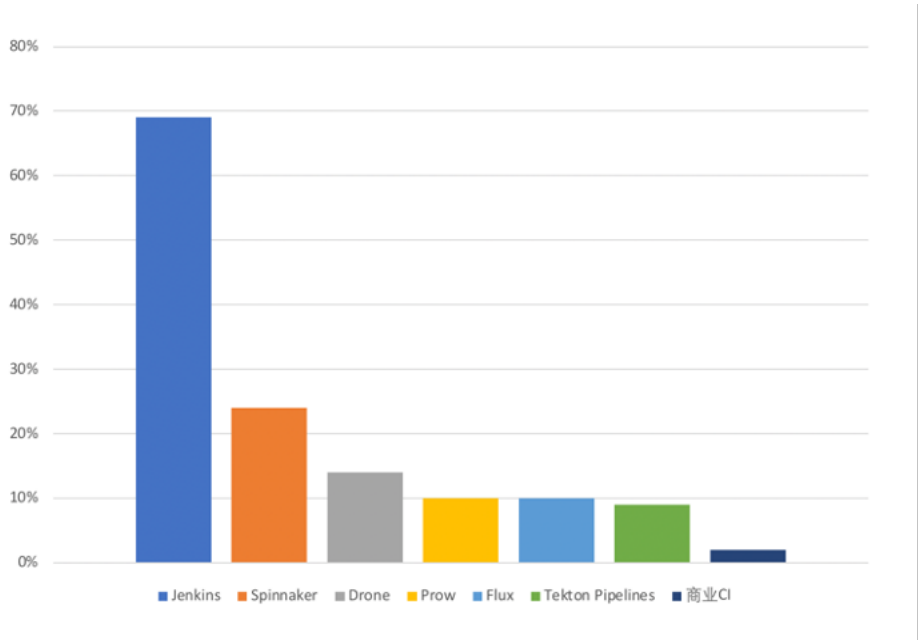

图 4 容器 CI/CD 使用情况

主流 CI/CD 工具介绍

如果说 DevOps 理念实现了软件的快速交付、那么 CI/CD （Continuous integration&Continuous Delivery&Continuous Deployment）便是实现这一理念的主要方

法，CI/CD 的核心概念是持续集成，持续交付，持续部署。依托 CI/CD，应用的整个生命周期（从集成和测试阶段，到交付和部署阶段）可达到持续自动化和持续监控的效果。

软件开发过程中，开发人员会频繁地提交代码到 Gitlab 主分支，而在代码最终合并至主分支前则需要一系列的自动化编译及测试流程，CI 主要用在源代码变更后自动进行检测、拉取和构建。目的是为了更快的将变更后的代码与原有代码正确的集成在一起[ii]。

CD 指的是持续交付/持续部署，其中持续交付通常指开发人员将 CI 流中合并的代码上传至存储库，例如 Gitlab，Github 等，再由运维人员将其部署至生产环境中，目的是为了确保尽可能减少部署新代码时的工作量[2]。持续部署则是将持续交付中的部署生产环境过程由“手动”变为“自动”，主要是为了解决手动可能造成的应用交付速率以及人力成本问题，图 展示了 CI/CD 流程中各自的责任划分：

图 5 CI/CD 流程责任划分

以上介绍可以看出，一款优秀的 CI/CD 工具成为企业落地 DevOps 的关键，下面我们以当下市场上使用较多的 Jenkins 为例进行介绍。

Jenkins4早期由 Sun 公司创建，而后迅速扩展为最大的开源 CI 工具5之一，Jenkins 可帮助开发团队实现应用的自动化构建、测试及部署，并且支持多平台部署（Windows、Linux、Mac OSX、Unix 等）。Jenkins 的一大优势为其丰富的插件系统，目前提供超过上千款插件，可以集成市面上几乎所有的工具，Jenkins 的主要架构如下图所示：

4 https://jenkins.io/
5 https://github.com/jenkinsci/jenkins

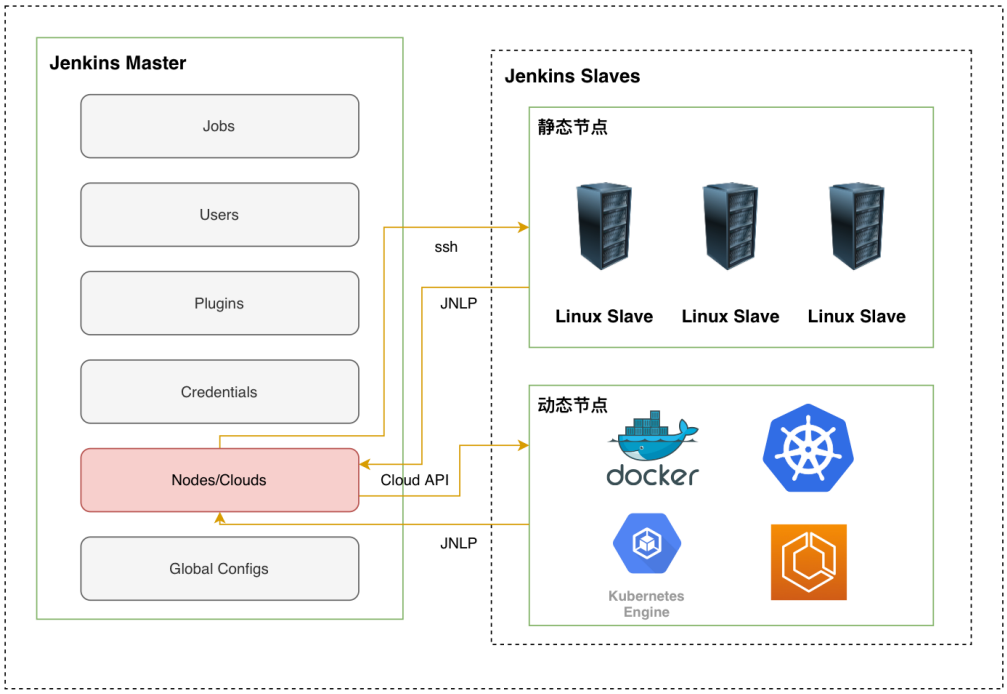

图 6 Jenkins 架构图

上图可以看出 Jenkins 架构包含一个 Master 节点和多个 Slave 节点。

Master 节点

Master 节点拥有许多关键的组件，其中包括用于构建源码和测试源码的 Jobs 组件、用于身份验证的 Users 组件、用于为 Jenkins 提供各类插件的 Plugins 组件、用于配置 Node，Clouds 节点从而执行 jenkins 作业的 Node/Clouds 组件等等。上述 Master 涉及的所有组件都包含一组配置文件，这些配置文件通过 Node/Clouds 组件分发至 Jekins Agent 并由其进行相应的持续集成操作。

Slave 节点

由图 可以看出 Slave 节点包含静态节点和云节点，静态节点的服务器将一直处于运行状态并保持与 Master 节点的连接，其通常通过一组脚本进行服务器的启停及重启操作；而云节点可以理解为动态服务器，即 Master 节点触发了一个任务后，云节点为其创建一个容器，并在任务完成后删除，从而在大规模使用持续集成部署时减轻一定的基础设施成本。

关于 Jenkins 更为详细的内容可参考官方文档[6]。

当然，除了 Jenkins 之外，还有许多 CI/CD 工具也被大量使用，例如 Drone CI[7]、Circle CI[8]、Gitlab CI[9]、Travis CI[10]、Spinnaker[11]等，其中每个 CI/CD 工具都有各自的特点，例如 Spinnaker 支持多云环境，可以在不同的云提供商间进行应用的发布和更改，Travis CI 为一个托管的 CI 服务，主要用于构建和测试托管在 GitHub 和 Bitbucket 上的项目，Drone CI 支持在云上、本地和 Kubernetes 上部署，并与 Kubernetes 有着良好的兼容性。读者可以各取所需。

1.1 小结

本附录对 DevOps 和 CI/CD 进行了相应介绍，云原生环境中，DevOps 作为必不可少的一环，通过其持续集成、持续交付、持续部署的特性为云原生赋能，进而更快、更稳定的交付云原生应用。随着云原生安全的不断普及，安全左移的理念产生 — 安全因素需要纳入应用开发的早期阶段，这与本章主题也息息相关，即在开发（Dev）与运维（Ops）之间加入安全 Sec，从而可使云原生应用的整个生命周期得到较为可靠的安全保障。

---

[i]《中国云原生用户调查报告》

[ii] https://www.redhat.com/zh/topics/devops/what-is-ci-cd

---

6 https://www.jenkins.io/doc/

7 https://www.drone.io/

8 https://circleci.com/

9 https://about.gitlab.com/product/continuous-integration/

10 https://travis-ci.com

11 https://www.spinnaker.io/