Devops-工具链平台的建设和思考

——持续集成领域

技术总体部 刘智勇

摘要：IT 时代的新潮流在缓慢涌动。而云计算，微服务，devops 作为这涌动浪潮中的绽放的绚烂浪花，深深吸引了技术人的目光。无论是在 IT 界，金融科技界还是公司内部，一直都在深耕这方面的研究和实践。本文立足于这个大背景，聚焦于 devops 的工具链平台建设，下接云计算（具体指 PaaS）平台的建设，上承微服务架构的推广，力求阐述清楚现有 devops 工具链平台的建设概况和设计思想，同时也对未来的演进做进一步的思考。

关键词：持续集成，三级跃迁，代码库，制品库，服务集群；

一、前言

关于 devops 的概念描述这里不再赘余。引用最常用的概念，其是一种重视“开发人员”和“运维人员”之间沟通合作的文化、运动和惯例。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**。

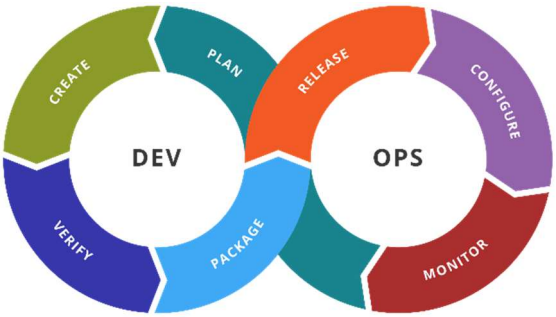

图 1

图 1 是 devops 工具链中比较著名的一个螺旋图。涵盖了 devops 工具链平台所有重点关心的 7 个领域，是对 IT 研发流程的一个简要抽象。依托这个图，业界的各大厂商（譬如国外的 gitlab，jfrog artifactory，redhat openshift 等，国内的阿里云效，easyops，谐云等），开源组织和政府机构（如信通院）有些在工具链研发上发力，有些力求在行业标准拥有权威，有些立足于工具集成和管理咨询上。形成了百花齐放，产品众多，实践多样化的现状。诚然，不同的组织有着各自的利益考量和实施环境，但是 devops 的初心“使得构建、测试、发布软件能够更加地**快捷、频繁和可靠**”应该成为每个从业者的追求。

图 2 是在分析公司内部项目发布流程的基础上，总结了现有工具链的使用情况图。由于各个项目在产品架构、研发模式、需求种类以及历史原因等，从 code 到 program 展现出了不同的路径。但是，以容器化平台支撑，服务于微服务架构的流水线平台将是一种软件发展的一种趋势。

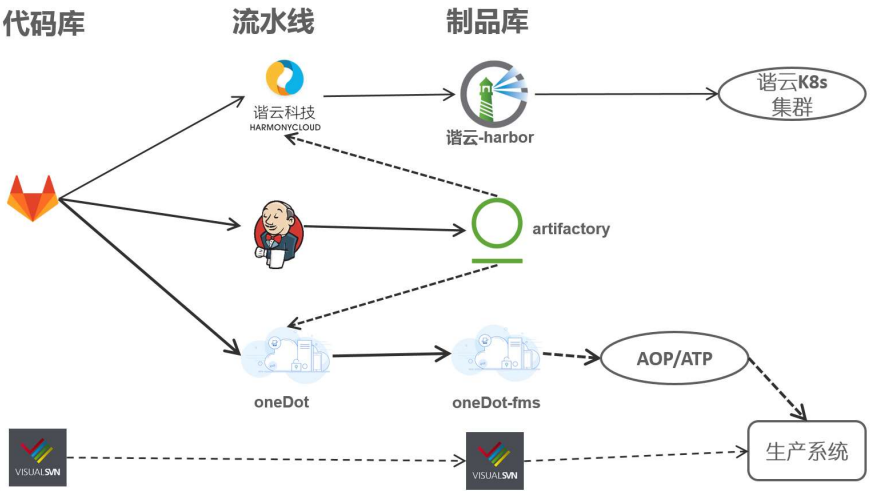

图 2

由于工具链平台的建设是一个多部门协作的工作。本文将站在开发部门，开发人员的角度，着重从代码管理，依赖管理，开发流水线对 devops 做一些工具介绍和模型设计。同时也注重与测试、运维部门的融合。

二、代码和分支模型

随着公司内部的代码管理从 svn 逐步转向 git。Gitlab 的运营模式以及代码库的分支模型也已经逐步有了清晰的实践和管理模式。从 Gitlab 的使用方的角度来看，需要熟练掌握 git 的使用，分支模型的设计从而实现对代码库的有效管理。目前 gitlab 在高可用、数据保密还有些欠缺，在分支模型上的实践还有很大的改进空间。下面重点阐述三种分支模型：

2.1 git flow

图 3 是 git flow 分支模型的概况图。对于 git flow 来说，存在两个长期分支：master，develop 分支。其中 master 分支追踪生产基线，develop 分支追踪开发基线。短期分支一般有三种：feature 分支，release 分支，hotfix 分支。发布对应 develop 分支本身的跳跃，develop 分支到 master 分支的跳跃，master 分支本身的跳跃。git flow 分支模型比较契合传统的开发流程，适合于开发，测试和运维分立的组织。但其本身分支过多，分支的协调多，对于快速敏捷的研发流程不太适应。在互联网组织，devops 的理念，这种方式是一种基本的分支模型。在我们公司内部，大部门项目会使用 git flow 或者 git flow 变种作为分支模型。

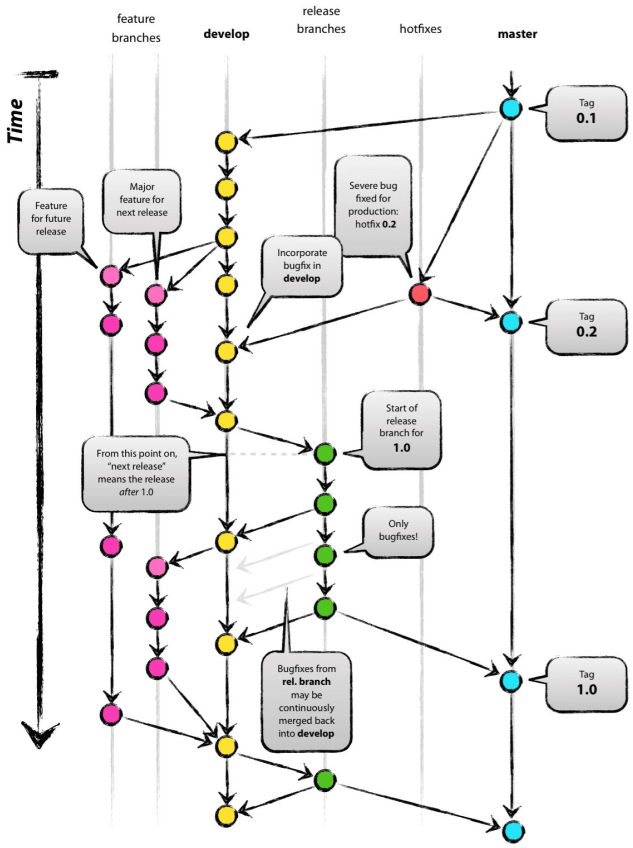

图 3

2.2 github flow

github flow 顾名思义，是 github 上面的开源项目广泛使用的分支模型。这种分支模型依赖两个非常重要的操作 fork 和 pull request（在 gitlab 里面称作 merge request）。fork 操作将代码库分为母库和子库，pull request 操作将子库的更新推送到母库上。pull request 操作的含义比较丰富，代码合并过程中可以搭配 code 流水线检测，code review，issue 交流等功能。通过 fork 和 pull request 操作，github flow 创造了一种开源项目的研发流程。它兼具开放、包容、民主和自由的特性，通过一种协作讨论的方式来创造产品。当然它的缺点也是明显的，disuss and review 过程有时候会很长，在商业化的产品里面容易对市场响应不够，从而错过良机。该分支模型适合于研究型项目。

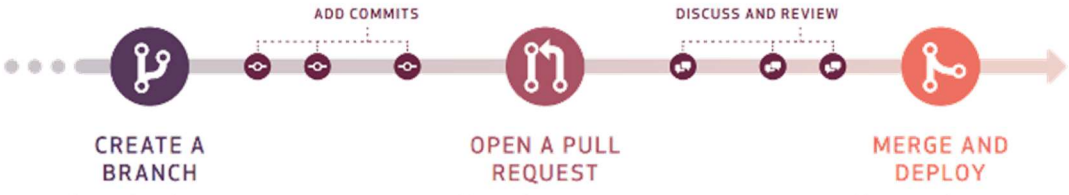

图 4

2.3 gitlab flow

gitlab flow 是一种全新的分支模型。它分为两种使用场景。

第一种使用场景称之为持续交付场景，在这种场景下，不太会有版本的概念，分支的存在和环境有关系。譬如有三类环境（开发，仿真和生产），那么如图五所示，可以有三个分支（master，pre-production，production），分别追踪开发，仿真和生产三个环境的基线。代码的跳跃严格从 master—>pre-production--→production，不允许有其他的跳跃方式。该模型其实要求组织内部有良好地 CICD 系统，可以毫不费力地实现稳定的多次发布（一天多次发布这种程度），对于很多组织是难以达到的。

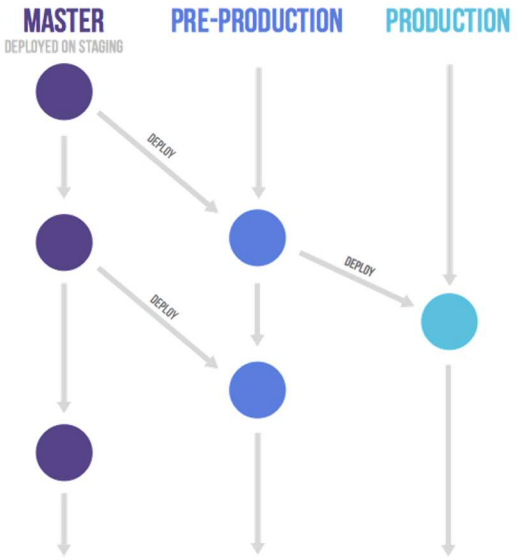

图 5

第二张场景称之为版本发布场景，也称为主线发布场景。见图 6，长期分支只有 master 是存在的，发布一个版本从 master 拉出 release-X.X.X-STABLE 分支。短期的版本分支一段时间后面积累过多，可以有一定的归档策略，可以建立类似分支回收器这样的自动化工具。release 的版本号规范可以参加 github 推出语义化版本规范，也可以参见《技术总体部—软件版本号编制规范》。

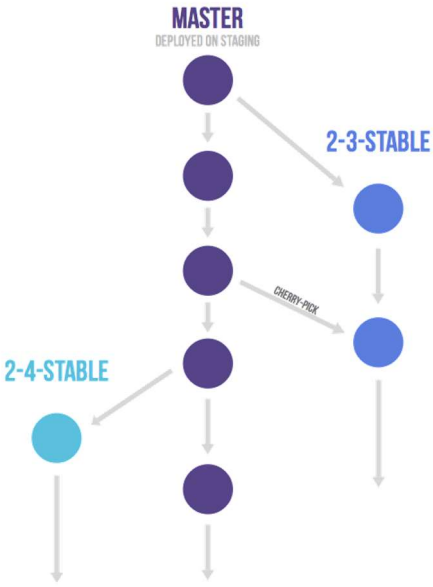

图 6

三、依赖和制品模型

软件的开发流程愈来愈倾向于小而美，团队之间要形成共享共建的合作氛围。为了适应这种思想体系。软件的组件化、抽象化、接口化越来越重要，接口化对于微服务架构很重要。大量的软件模块提高了重用性和组装的方便，但也带来了管理上的复杂（换一个角度其实高效的管理促进生产力的进步）。目前来看，在公司内部，二方库（公司内部的研发软件集合）和三方库（公司引用的外部软件集合）的概念和实践已经逐步清晰，对于依赖管理工具 maven，npm 等的认可度也有了很大提升。而一个功能齐全、管理完备、可靠稳定的依赖和制品平台就显得很重要。Jfrog artifactory 目前就是在为这一目标而努力。下面将重点阐述 artifactory 提供的依赖体系和库模型。

3.1 依赖体系

基于 OpenStack 的云平台很好地提供了主机的虚拟化，提供了很好的一个 IaaS 平台。对于 PaaS 平台的建设，现在看来，依赖体系（OS，组件，中间件，服务件）的建设，持续交付平台的建设，应用虚拟化的支撑平台建设成为三个重要的方向。由图 7 可以见到，基础的 OS 的依赖管理由 debian（ubuntu 体系）和 rpm，yum（centos 体系）完成，由此在 IaaS 层的基础上丰富了 os 的功能；在此基础上，编程语言的依赖管理由 maven（Java 体系），cocan（C/C++），npm（JavaScript），nuget（C#），gradle（Android），cocoapods（ios）等来完成，从而支撑应用程序的开发，编译，调试，测试以及运行过程。更上一层，使用容器化技术 docker 对应用程序进行封装，从而将部署上线过程抽象化，保证不同时间不同地点的环境一致性。最后，在 k8s 集群和 helm 编排工具的支撑下，服务群将会成为最终的交付对象，从而打通并融合到微服务架构上来。由此可见，依赖体系和制品库对于 PaaS 建设至关重要，依赖体系很好地契合了小而美和共享共建的研发体系，而制品库是支撑软件产品云化和微服务化的重要基础设施。

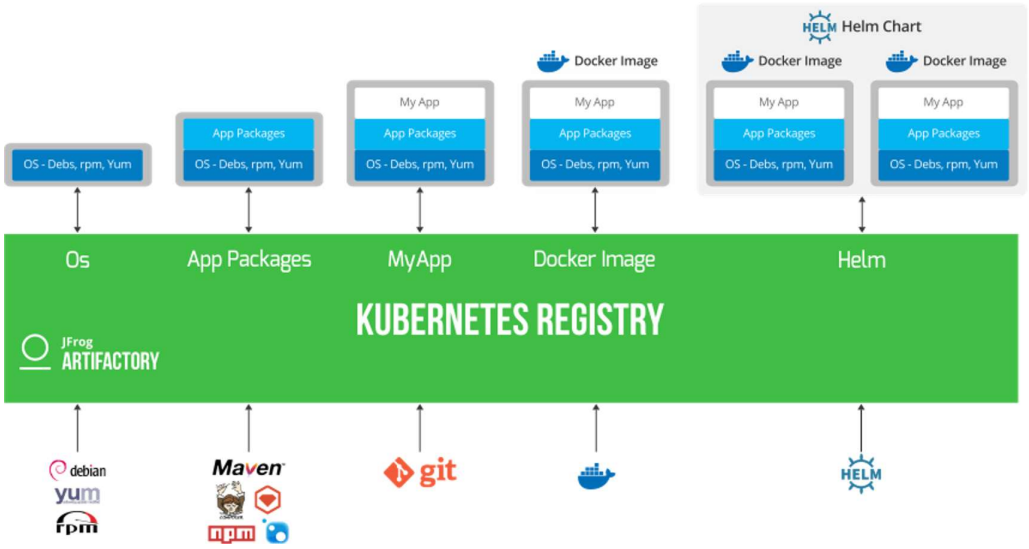

图 7

从实践和推广来看，依赖体系的递进难度也是逐次递升的。对于 maven 推广，只要熟悉 pom 配置，maven 制品库和 maven 生命周期即可。而对于 docker 的推广，则涉及到 Dockerfile 编写，docker 镜像库，docker 容器运行平台和 docker 的命令行 API。而 docker Helm 的推广，则更为繁杂，它需要一个稳健可靠的 k8s 运行平台，属性 Helm 配置文件以及前一

递进的基础支撑。单单一个文件可靠的 k8s 运行平台就涉及很多内容，集群本身的组件就比较繁杂，从集群管理上，API server，etcd，scheduler，controller manager 这四大组件就涉及分布式数据一致性，集群调度，旁路控制等思想。从集群的运行上，kubelet，kube-proxy 是 pod 和 service 的主要管理者，kubelet 会调用 docker 等容器引擎，而 kube-proxy 作为网络代理，使用 iptables 等技术实现了负载均衡，流量控制灯杆。此外，更好的 PaaS 服务还需要丰富 k8s 集群的附加组件，增加集群的稳定性，丰富规范应用程序的支撑镜像等。

3.2 库模型

Jfrog artifactory 通过内容 hash 存储技术，完善的依赖工具集成，良好地制品元数据管理技术为我们实践依赖体系提供了重要支撑。而对于 devops 的落地来说，组织内部需要结合自身实践设计良好地库模型，部署结构和使用流程。在设计思想上，遵循“一次构建，处处升级”的原则（见图 8），并充分考虑制品语言，部署网段和制品来源的多样性。在此基础上，我部现在已经制定形成了《制品平台网段模型》,《制品库模型》,《软件版本号编制规范》等，并成功在内外网 artifactory 平台上予以实践。

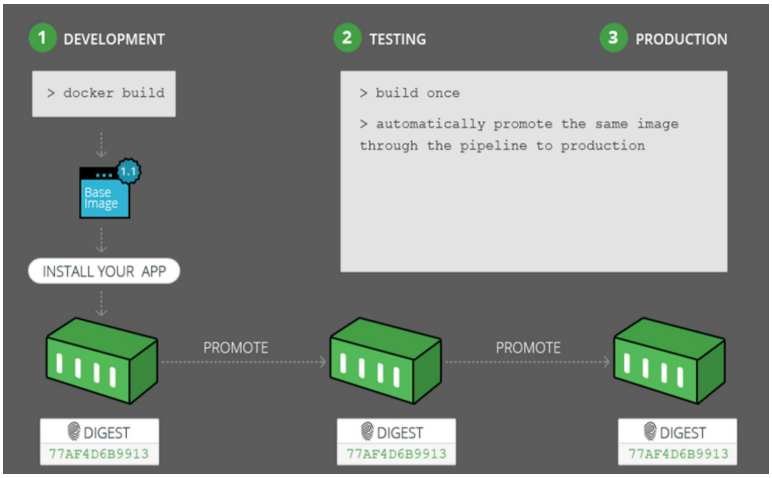

图 8

图 9 是制品库模型的简易逻辑图。其解释如下：

制品库的对外交流：开发者（developer），终端用户（end-user），互联网第三方依赖（Internet remote）；

对应的三个虚拟仓库：remote，prod-virtual，dev-virtual

（1）remote 代理众多的外部第三方依赖的 cache 库；

（2）pro-virtual 代理组织内部上生产的制品库 pro-local；

（3）dev-virtual 代理 remote，prod-virtual 和开发的本地库 dev-local；

（4）pro-virtual 是软件中心一个数据来源；

对应组织内部流程，区分出 dev-local，testing，staging，prod-local 等本地二方库。这些二方库是支撑三级跃迁流水线的基础。

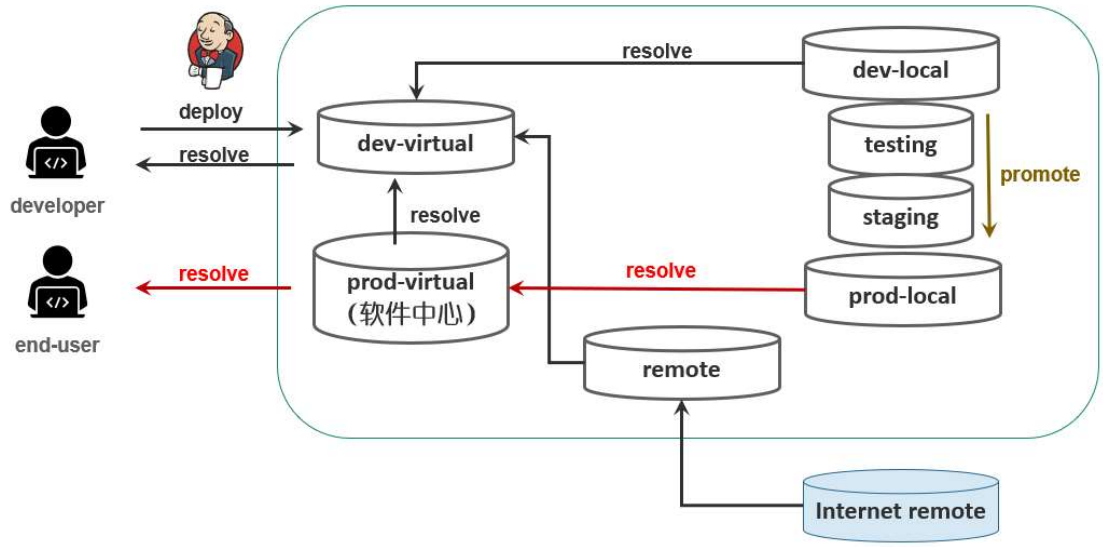

图 9

从使用诉求来说，制品库的服务对象分为两类，开发者（包括开发，测试和运维）和终端用户。两类服务对象的不同点左右，开发者从制品库获取和存储软件半成品，其使用制品库的目的在于开发完善其软件产品。终端用户的目的是从制品库直接获取软件并创造业务价值。对用户使用诉求的区分，可以比较好地将制品资产分为半成品资产和成品资产来管理，在流程设计上也可以很好地加以区分。

成品资产的管理其实就是软件中心概念，它的产品形态非常类似于苹果的 app store，360 的应用商店等此类产品。其来源分为内部自建（内部自建流程会和半成品资产的管理串联起来）和外部引入（外部引入又区分为付费和免费）。

半成品资产的管理是后续三级跃迁流水线的基础，是“一次构建，处处升级”思想的实践。从 gitlab 的代码到 artifactory 的制品为一级跃迁。后续二级和三级跃迁嵌入了组织内部的流水线，并体现在制品库的（dev-local，testing，staging，pro-local）的设计上。

四、CICD 和容器化流水线模型

CI（持续集成）和 CD（持续交付）是 devops 中重点推进的两个研发实践。CICD 需要文化，反馈和管理上的改进，在工具链的建设主要是流水线的建设，流水线比较广泛依赖的是 Jenkins 流水线平台。原生的 Jenkins 流水线经过多年的发展，具备了支撑了多过程（stages），多项目（project）和多发布版本（release）的能力。新开启的 Jenkins X 项目更添加了对 k8s 集群的支撑，从而为容器化 Jenkins 流水线平台，容器化构建集群，模板化构建工具提供了可能。

4.1 容器化流水线平台

容器化技术作为新技术，拥有很多令人激动的特性。广泛的容器化应用可以提升整个研发团队的效率，改变软件的研发模式等。但容器化的落地是一个知易行难的过程，涉及到应用架构，组织流程等。不同的应用对于容器化的诉求各异。对于容器化应用，有必要采取先易后难，先开发测试后生产，先无状态后有状态，先低保障应用后高保障应用，先小应用后大应用，先内部应用后外部应用的策略。

流水线平台中的构建集群作为构建过程中的主力支撑，具有容易，开发测试环境，无状

态，不需要保障，应用小，内部应用的特点。作为 k8s 集群的研究和经验积累样例是比较合适。此外，容器化带来的构建工具模板化，集群伸缩功能又会很好地提升 CI 过程的效率。

目前容器化流水线平台的建设主要集中在 CI 过程，左连 gitlab，右接 artifactory。以 k8s 集群为支持，在后面会给我们的构建过程带来很好的体验提升。

4.2 容器化三级跃迁流水线模型

在代码库（gitlab），流水线平台，制品库（artifactory），容器集群（k8s）建设完备的情况下，流水线的内容成为 CICD 的主要努力方向。为了更好地梳理出建设思路，依据“一次构建，处处升级”的思想，提出一种三级跃迁的流水线建设模型。图 10 是跃迁的总体的概览图（实际上有四个跃迁，0 级跃迁在开发者的环境上）。

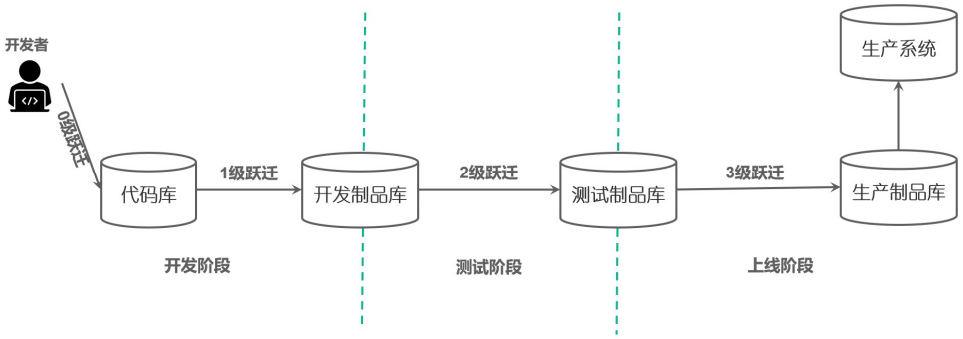

图 10

4.2.1 第 0 级跃迁

从开发机上的代码到 gitlab 的代码跃迁过程。如图 11 所示，具体的内容包括 coding，debug，UT，package，smock test 等。Docker desktop 和 minikube 可以提供 docker 和 k8s 的调试环境，便于开发者对自己的代码镜像容器化构建和调试，同时制品库需要提供相应的镜像支持服务（参见第三章依赖和制品模型）。代码规范和安全规则校验是这其中的重要研究内容（包含 IDE 检查和入库前的 code review）。

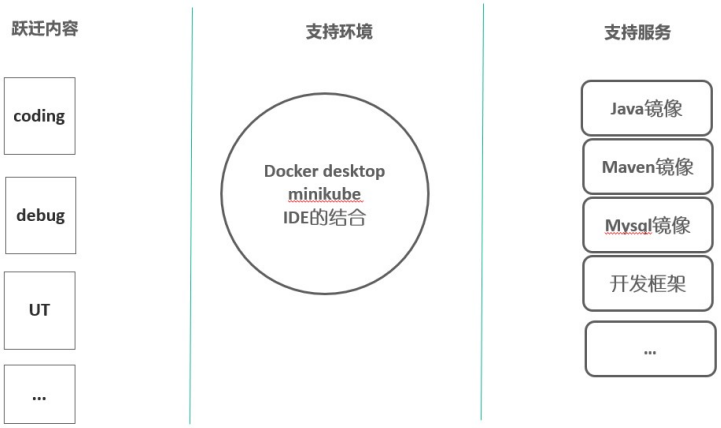

图 11

4.2.2 第 1 级跃迁

从 gitlab 代码到开发制品的跃迁过程。如图 12 所示，具体的内容包括构建，白盒测试，打包等。代码规范和安全规则校验应该重点关注（包括 sonarqube 检查和入库后的 code

review），API 接口，软件版本号规范也是重点建设内容（API 接口库和软件版本号编制规范）。

图 12

4.2.2 第 2 级跃迁

从开发制品到测试制品的跃迁过程。如图 13 所示，具体内容需要参照测试的分级规划和对软件产品的剖析程度。K8s 的测试集群已经众多的镜像可以有效地支持测试的自动化和增加测试的完备度。

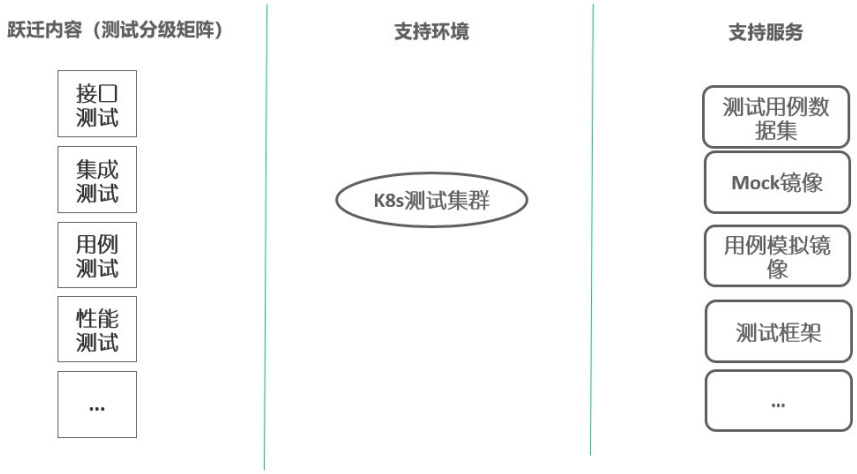

图 13

4.2.2 第 3 级跃迁

从测试制品到生产制品的跃迁过程。如图 14 所示，具体内容是上线，回退等过程。生产系统中配置，监控，日志等是开发测试所没有的。这些也是上线不同于前两个跃迁的地方。而 k8s 生产集群的运维难度相比于前两个集群需要做的工作将会提升好几个级别。

图 14

五、拓展实施和运营

本文的第二到第四章，分别介绍了 devops 工具链中现有重要的三个系统：Gitlab 代码库，artifactory 制品库和 Jenkins 流水线平台。这三大系统的基本可以很好地完成 CI 的功能，部分支撑 0 级跃迁和 1 级跃迁的内容。然而，如果拓展到持续交付，甚至是整个 devops 工具链，我们做的工作还很不够。从 gitlab 左移，如何让 jira 和 confluence 融入到这条线内，需求和缺陷管理的建设还需要再推进一些。从 artifactory 右移，除了对依赖体系的层层推进，k8s 的构建集群，k8s 的测试集群，k8s 的生产集群目前还处于实验室阶段。要让这些集群和镜像集合真正发挥技术价值，还需要更多的努力和推广。

从 devops 落地实施的角度来看，我们的项目处于传统和现代的交织点。有一些项目渐渐有了容器化和微服务的雏形，有一些还处于传统的 IT 体系下。很多依赖工具，流水线内容等我们还没有做。相关的培训和落地推广还很欠缺。

最后，整个工具链的运营也需要进一步加强。SSO 在过去一年取得了很大进步，但依然有 svn 等一些工具没有纳入进来；域名服务，AD 服务，邮件服务与整个工具链网络连通的问题还没有彻底解决；工具链本身也需要不定时地升级和维修；重要节点需要做高可用架构部署和容灾备份方案；整个工具链的安全维度的关注还比较少，运营规范和使用规范需要加强，数据资产的保密性和人员权限分配还有随意的地方；整个工具链的日志和监控体系也没有纳入进来；工具链平台的运营分析（平台分析，内容分析）也有很大的拓展空间。

六、结语

本文通过对 gitlab，artifactory 和 Jenkins 容器化流水线的剖析，概述了 devops 持续集成工具链的建设概况和设计思想。其中还有很多技术细节没有阐述到。譬如各个依赖体系工具的详细介绍，制品库的网段方案，k8s 集群的原理和运营思路，流水线和各个服务的集成等。这里面有一些是技术原理，有一些是运营指南，也有一些是使用规范。相信随着我们研究和实践的深入，这些问题都会一一解决，从而将我们的研发生产力提高的一个新的水平，让又好又快的创造符合业务需求的软件产品不再是难事。

参考文献：

[1]. JezHumble, DavidFarley, 亨布尔, 法利, & 乔梁. (2011). 持续交付：发布可靠软件的系统方法. 人民邮电出版社.
[2]. 浙江大学 SEL 实验室. Docker：容器与容器云[M]. 人民邮电出版社, 2015.
[3]. 张慧. GIT 分支开发的研究与应用[J]. 经营管理者, 2009(2X):144-144.
[4]. 周莹, 欧中红, 李俊. 基于 Jenkins 的持续集成自动部署研究[J]. 计算机与数字工程, 2016(2):267-270.
[5]. Scaling CD Pipelines, Thought works. 2017.11.8
[6]. Jfrog. https://www.jfrog.com/confluence/display/RTF/Artifactory+User+Guide
[7]. Kubernetes. https://kubernetes.io/docs/home/