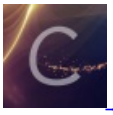
DevOps-深入浅出详解

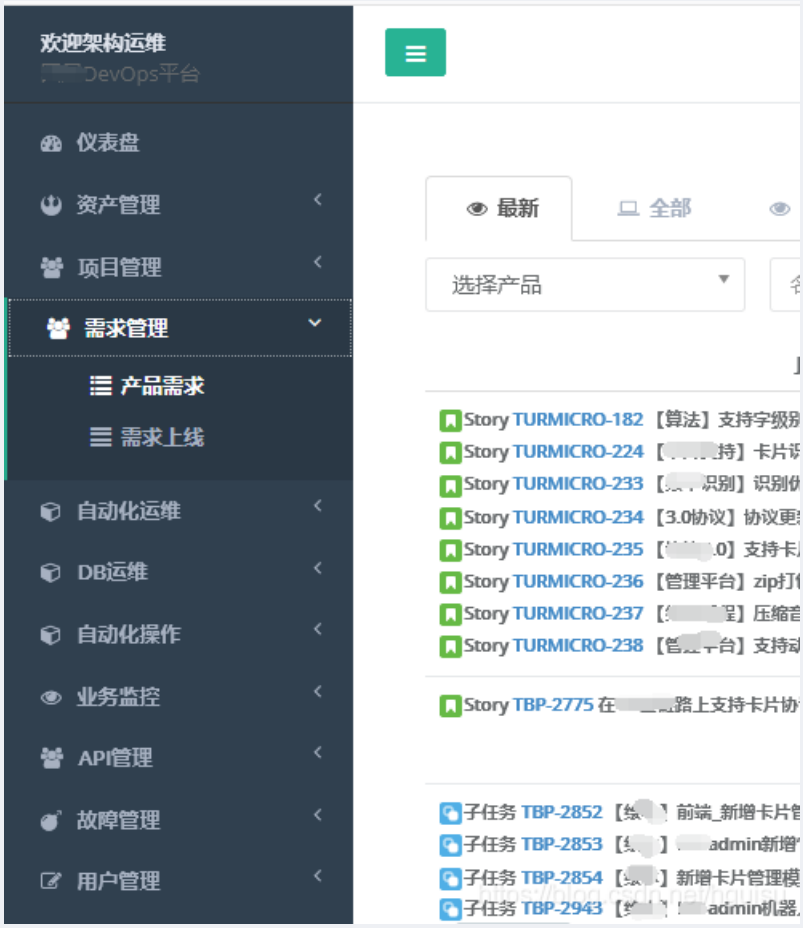
原创

hguisu 于 2020-08-20 19:14:53 发布 1910 收藏 15
分类专栏： 服务治理

本文链接：https://blog.csdn.net/hguisu/article/details/103298873



本文2018年总结网上相关文章，纯属学习笔记使用。感谢原创，在学习过程中，我们也按照DevOps指导思想实现DevOps的落地实施。



DevOps标准：研发运营一体化（DevOps）能力成熟度模型

-= 研发运营一体化（DevOps）能力成熟度模型 =-

能力类	一、研发运营一体化（DevOps）过程																
能力域	敏捷开发管理(2)			持续交付(3)							技术运营(4)						
能力子域	价值交付管理	敏捷过程管理	敏捷组织模式	配置管理	构建与持续集成	测试管理	部署与发布管理	环境管理	数据管理	度量与反馈	监控管理	事件与变更管理	配置管理	容量与成本管理	高可用管理	连续性管理	用户体验管理
能力项	需求工件	价值流	敏捷角色	版本控制	构建实践	测试分层策略	部署与发布模式	环境管理	测试数据管理	度量指标	数据采集	事件管理	运营配置管理	容量管理	应用高可用管理	风险管理	业务认知管理
	需求活动	仪式活动	团队结构	变更管理	持续集成	代码质量管理	持续部署流水线		数据变更管理	度量驱动改进	数据管理	变更管理		成本管理	数据高可用管理	危机管理	体验管理
						自动化测试					数据应用					应急管理	
能力类	二、研发运营一体化（DevOps）应用设计																
能力类	三、研发运营一体化（DevOps）安全及风险管理																
能力类	四、研发运营一体化（DevOps）评估方法																
能力类	五、研发运营一体化（DevOps）系统和工具																

前言

提到DevOps这个词，我相信很多人一定不会陌生。作为一个热门的概念，DevOps近年来频频出现在各大技术社区和媒体的文章中，备受行业大咖的追捧，也吸引了很多吃瓜群众的围观。

那么DevOps是什么呢？

有人说它是一种方法，也有人说它是一种工具，还有人说它是一种思想。更有甚者，说它是一种哲学。

越说越玄乎，感觉都要封神啦！

DevOps这玩意真的有那么夸张吗？

它到底是干嘛用的？

为什么行业里都会对它趋之如鹜呢？

一、DevOps的起源

从程序最开始讲起吧。

程序员（Programmer）：

上个世纪40年代，世界上第一台计算机诞生。从诞生之日起，它就离不开程序（Program）的驱动。而负责编写程序的人，就被称为“程序员”（Programmer）。程序员是计算机的驾驭者，也是极其稀缺的人才。那个时候，只有高学历、名校出身的人，才有资格成为程序员，操控计算机。

软件（software）：

随着人类科技的不断发展，PC和Internet陆续问世，我们进入了全民拥抱信息化的时代。越来越多的企业开始将计算机作为办公用的工具，用以提升生产力。而普通个人用户也开始将计算机作为娱乐工具，用以改善生活品质。于是，计算机的程序，开始变成了一门生意。程序，逐步演进为“软件（software）”，变成了最赚钱的产品之一。

软件开发工程师（Software Development Engineer）：

在软件产业里，程序员有了更专业的称谓，叫做“软件开发工程师（Software Development Engineer）”，也就是我们常说的“码农”。

软件开发过程

我们知道，一个软件从零开始到最终交付，大概包括以下几个阶段：规划、编码、构建、测试、发布、部署和维护。

最初，程序比较简单，工作量不大，程序员一个人可以完成所有阶段的工作。

软件开发分工

随着软件产业的日益发展壮大，软件的规模也在逐渐变得庞大。软件的复杂度不断攀升。一个人已经hold不住了，就开始出现了精细化分工。

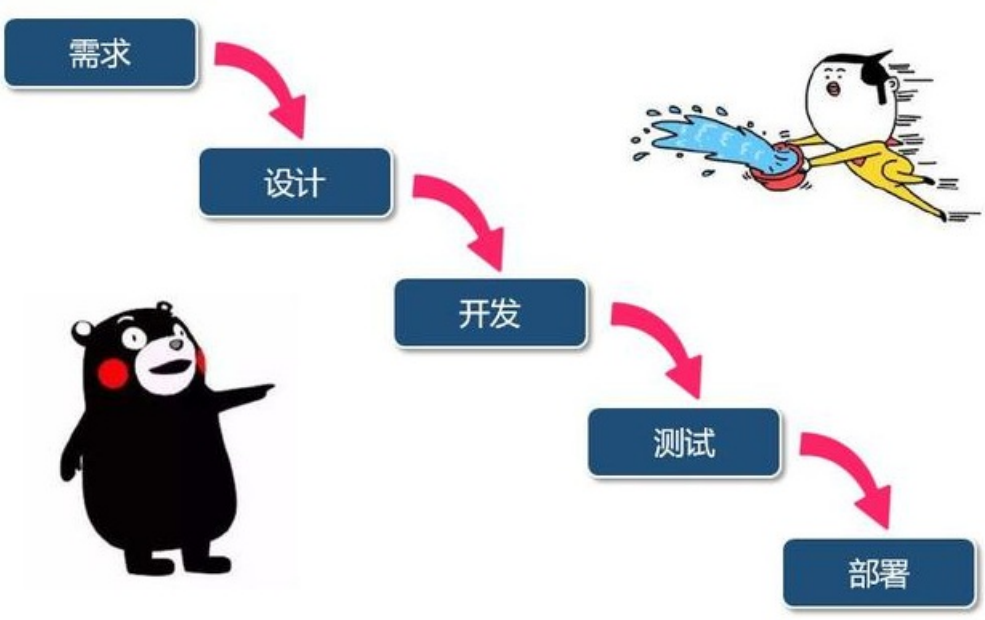
码农的队伍扩大，工种增加。除了**软件开发工程师**之外，又有了**软件测试工程师**，**软件运维工程师**。

分工之后，传统的软件开发流程是这样的：

软件开发人员花费数周和数月编写代码，然后将代码交给QA（质量保障）团队进行测试，然后将最终的发布版交给运维团队去布署。所有的这三个阶段，即开发，测试，布署。

开发模式：瀑布（Waterfall）模型

早期所采用的软件交付模型，称之为**“瀑布（Waterfall）模型”**。



瀑布模型，简而言之，就是等一个阶段所有工作完成之后，再进入下一个阶段。

这种模型适合条件比较理想化**（用户需求非常明确、开发时间非常充足）**的项目。大家按部就班，轮流执行自己的职责即可。

但是，项目不可能是单向运作的。客户也是有需求的。产品也是会有问题的，需要改进的。

开发模式：敏捷开发（Agile Development）

随着时间推移，用户对系统的需求不断增加，与此同时，用户给的时间周期却越来越少。在这个情况下，大家发现，笨重迟缓的瀑布式开发已经不合时宜了。

于是，软件开发团队引入了一个新的概念，那就是大名鼎鼎的——**“敏捷开发（Agile Development）”**。

敏捷开发在2000年左右开始被世人所关注，是一种能应对快速变化需求的软件开发能力。其实简单来说，就是把大项目变成小项目，把大时间点变成小时间点，然后这样：

CI是Continuous Integration（持续集成）和Continuous Delivery（持续交付/部署）

有两个词经常会伴随着DevOps出现，那就是CI和CD。CI是Continuous Integration（持续集成），而CD对应多个英文，Continuous Delivery（持续交付）或Continuous Deployment（持续部署）。

美其名曰：“**持续（Continuous）**”，其实就是“加速——反复——加速——反复……”，这样子。

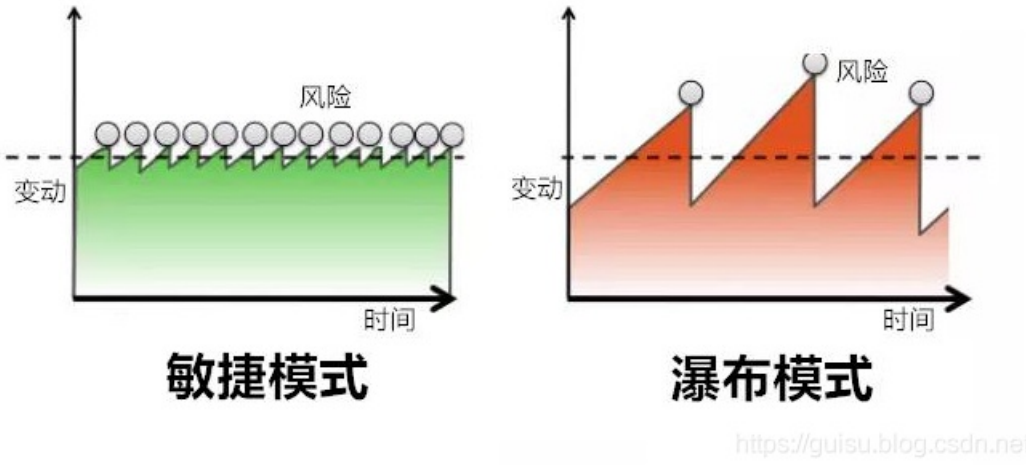
画个图大家可能更明白一点：

敏捷开发大幅提高了开发团队的工作效率，让版本的更新速度变得更快。

很多人可能会觉得，“更新版本的速度快了，风险不是更大了吗？”

其实，事实并非如此。

敏捷开发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**版本变化**是比较小的，风险会更小（如下图所示）。即使出现问题，修复起来也会相对容易一些。



虽然敏捷开发大幅提升了软件开发的效率和版本更新的速度，但是**它的效果仅限于开发环节**。研发们发现，运维那边，依旧是铁板一块，成为了新的瓶颈。

到底什么地方出问题了？

运维工程师，和开发工程师有着完全不同的思维逻辑。运维团队的座右铭，**很简单，就是“稳定压倒一切”。运维的核心诉求，就是不出问题。**

什么情况下最容易出问题？发生改变的时候最容易出问题。所以说，运维非常排斥“改变”。

于是乎，矛盾就在两者之间集中爆发了。

这个时候，我们的DevOps，隆重登场了。

DevOps概念从2009年提出已有8个年头。可是在8年前的那个时候，为什么DevOps没有迅速走红呢？即便是在2006年Amazon发布了ECS，微软在2008年和2010年提出和发布了Azure，DevOps的重要性似乎都没有那么强烈。我分析其**原因**主要有：

1.第一个很重要的**原因**是因为那时候云计算还是小众产品，更多的与虚拟化、虚拟机相关，它们还是重量级的IT基础设施。

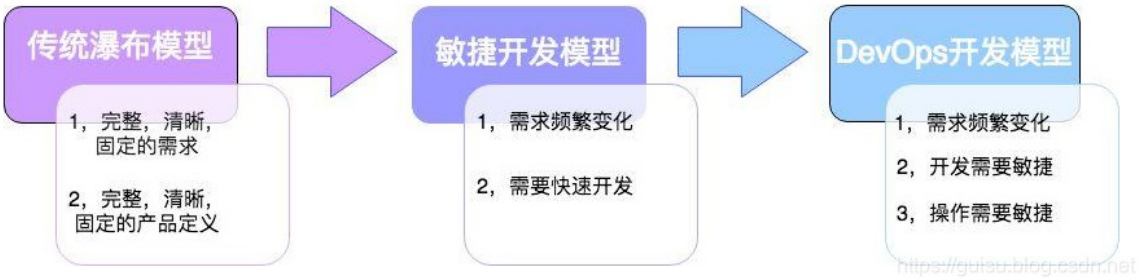
2.第二个很重要的**原因**是容器相关技术（Docker为代表）还没有横空出世，直到2013年7月。

3.第三个很重要的**原因**是，Martin Fowler在2014年3月提出了Micro Service，这为DevOps的推广也打了兴奋剂。

可以看出，当前DevOps概念的深入人心，离不开云计算、容器/Docker、微服务、敏捷等相关概念和实施的成熟发展。

二、软件开发模型的演变

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。

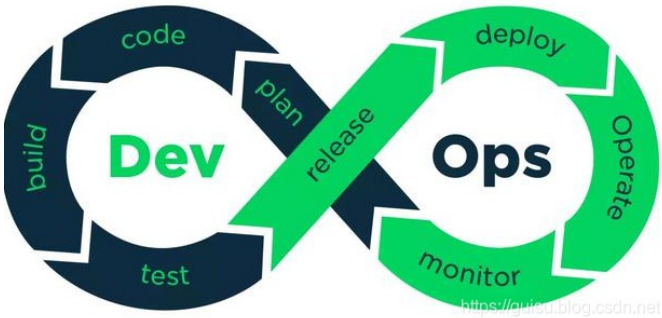
虽然这种敏捷的SCRUM方法为开发带来了敏捷性，但它在运维方面却失去了敏捷实践的速度。开发人员和运维工程师之间缺乏协作仍然会减慢开发过程和发布。

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。

三、DevOps到底是什么

DevOps这个词，其实就是Development和Operations两个词的组合。它的英文发音是 /de'vɒps/，类似于“迪沃普斯”。



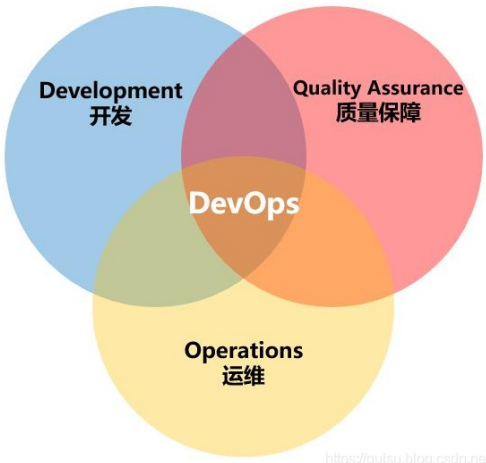
DevOps的维基百科定义是这样的：

DevOps是一组过程、方法与系统的统称，用于促进开发、技术运营和质量保障（QA）部门之间的沟通、协作与整合。

2、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。

3、**DevOps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。DevOps通过自动化完成“软件交付”和“架构变更”流程，来更加快捷、频繁和可靠地构建、测试、发布软件。可以把DevOps看作开发（软件工程）、技术运营和质量保障（QA）三者的交集。**

DevOps示意图

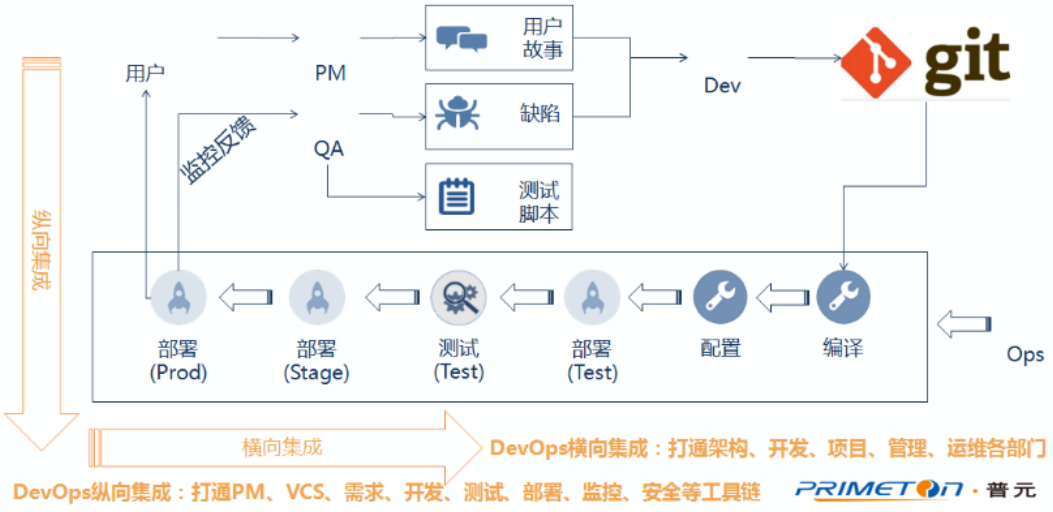


维基百科对DevOps的定义比较拗口，稍微有点抽象，但是并不难理解。它不是某一个特定软件、工具或平台的名字。

从目标来看，DevOps就是让开发人员和运维人员更好地沟通合作，通过自动化流程来使得软件整体过程更加快捷和可靠。

其实往简化里讲DevOps是提倡开发和IT运维之间的高度协同，从而在完成高频率部署的同时，提高生产环境的可靠性、稳定性、弹性和安全性。

从另外一个维度，广义上来说，DevOps不仅需要打通开发运维之间的部门墙，我们认为DevOps更多的需要从应用的全生命周期考虑，实现全生命周期的工具全链路打通与自动化、跨团队的线上协作能力。



第一，**纵向集成**，打通应用全生命周期（需求、设计、开发、编译、构建、测试、打包、发布、配置、监控等）的工具集成。纵向集成中DevOps强调的重点是跨工具链的「自动化」，最终实现全部人员的「自助化」。举个例子，项目组的开发人员可以通过DevOps的平台上，自主申请开通需要的各种服务，比如开通开发环境、代码库等。

第二，**横向集成**，打通架构、开发、管理、运维等部门墙。横向集成中DevOps强调的重点是跨团队的「线上协作」，也即是通过IT系统，实现信息的「精确传递」。举个例子，传统的系统上线部署方式，可能是一个冗长的说明文档，上百页都有可能，但在DevOps的平台下，就应该是通过标准运行环境的选择、环境配置的设置、部署流程的编排，实现数字化的「部署手册」，并且这样的手册，不仅操作人员可以理解，机器也能够执行，过程可以被追踪和审计。

DevOps是通过工具链与持续集成、交付、反馈与优化进行端到端整合，完成无缝的跨团队、跨系统协作。

破墙工具：

DevOps的目的是将开发和运维的对立面打破，使两者融合，你中有我，我中有你。

随着业务复杂化和人员的增加，开发人员和运维人员逐渐演化成两个独立的部门，他们工作地点分离，工具链不同，业务目标也有差异，这使得他们之间出现一条鸿沟。开发要求变化，运维要求稳定，开发团队和运维团队的工作方式和思维方式有巨大的差异：开发团队和运维团队生活在两个不同的世界，而彼此又坚守着各自的利益，所以在这两者之间工作到处都是冲突。

一般而言，当企业希望将原本笨重的开发与运营之间的工作移交过程变得流畅无碍，他们通常会遇到以下三类问题：

发布管理问题：

很多企业有发布管理问题。他们需要更好的发布计划方法，而不止是一份共享的电子数据表。他们需要清晰了解发布的风险、依赖、各阶段的入口条件，并确保各个角色遵守既定流程执行。

发布/部署协调问题：

有发布/部署协调问题的团队需要关注发布/部署过程中的执行。他们需要更好地跟踪发布状态、更快地将问题上升、严格执行流程控制和细粒度的报表。Ops 所在的部门绩效分为两块：一块为常规运维绩效（保证系统稳定性），另一块为 DevOps 项目绩效（保证开发顺利性），可以根据具体工作状况来设置这样的工作比率。

发布/部署自动化问题：

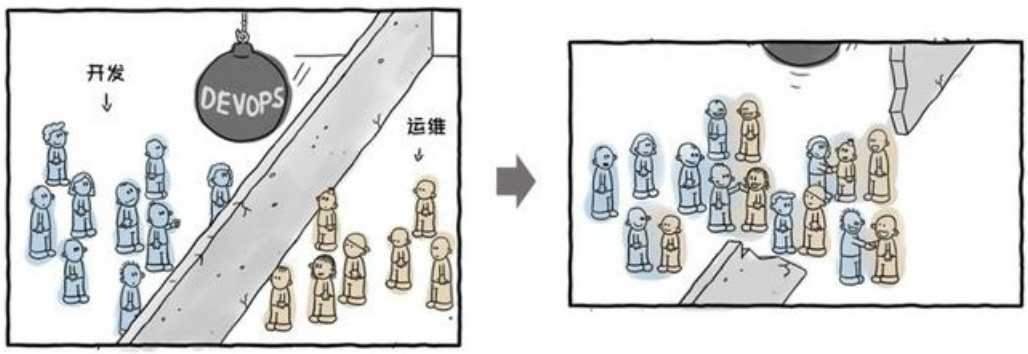
这些企业通常有一些自动化工具，但他们还需要以更灵活的方式来管理和驱动自动化工作——不必要将所有手工操作都在命令行中加以自动化。理想情况下，自动化工具应该能够在非生产环境下由非运营人员使用。

DevOps打破了开发人员和运维人员之间历来存在的壁垒和沟鸿，加强了开发、运营和质量保证人员之间的沟通、协作与整合。从而形成了一种通过持续交付来优化资源和扩展应用的新方式。DevOps和云原生相结合，能够让企业不断改进产品开发流程，更好地适应市场变化，提供更优质的服务。

1、运维人员会在项目开发期间就介入到开发过程中，了解开发人员使用的系统架构和技术路线，从而制定适当的运维方案；

2、开发人员，也会在运维的初期参与到系统部署中，并提供系统部署的优化建议。

DevOps的实施，促进开发和运维人员的沟通，增进彼此的理解。通过合力共事，在问题出现时就能一起承担，迅速解决。



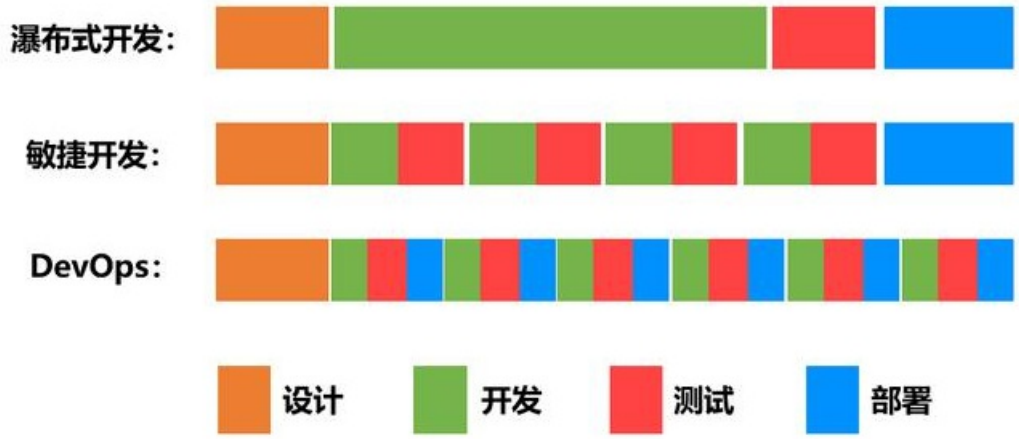
高效快速CI/CD（持续集成/持续部署）

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。

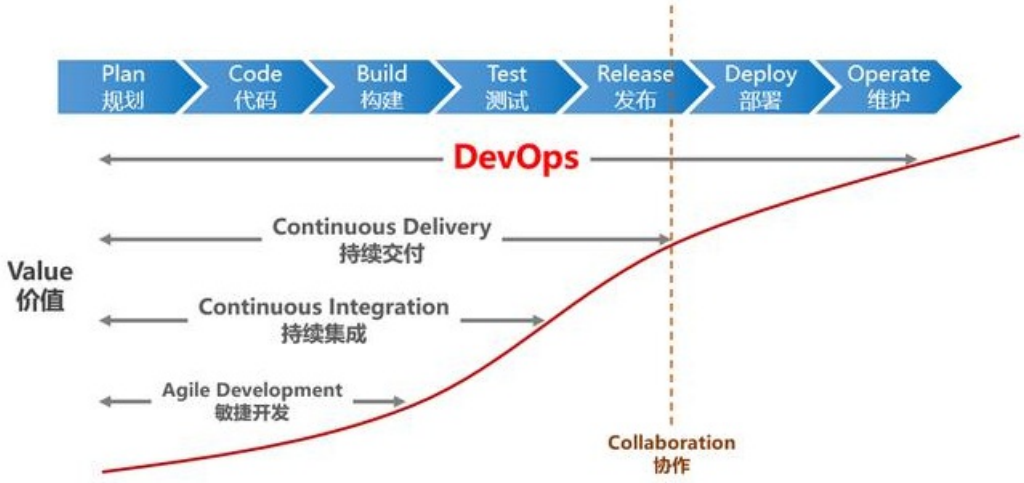
CI/CD（持续集成/持续部署）管道可以说是实施 DevOps 的一大重要成果，可帮助企业在需要很少的人工干预的情况下，更快速、更频繁地向客户交付应用，并不断改进产品的质量，增加服务功能，实现精益求精的发展。在整个生命周期内，CI/CD都引入了持续自动化和持续监控，从而能够快速识别和改正问题与缺陷，实现敏捷开发。

四、DevOps价值

对比前面所说的瀑布式开发和敏捷开发，我们可以明显看出，DevOps贯穿了软件全生命周期，而不仅限于开发阶段。



下面这张图，更明显地说明了DevOps所处的位置，还有它的价值：



DevOps考虑的还不止是软件部署。它是一套针对这几个部门间沟通与协作问题的流程和方法。DevOps的好处更多在于持续部署与交付，这是对于业务与产品而言。而DevOps 文化与技术方法论，它是部门间沟通协作的一组流程和方法，有助于改善公司组织文化、提高员工的参与感。

1、价值1：破墙工具：

DevOps的目的是将开发和运维的对立面打破，使两者融合，你中有我，我中有你。

随着业务复杂化和人员的增加，开发人员和运维人员逐渐演化成两个独立的部门，他们工作地点分离，工具链不同，业务目标也有差异，这使得他们之间出现一条鸿沟。开发要求变化，运维要求稳定，开发团队和运维团队的工作方式和思维方式有巨大的差异：开发团队和运维团队生活在两个不同的世界，而彼此又坚守着各自的利益，所以在这两者之间工作到处都是冲突。

一般而言，当企业希望将原本笨重的开发与运营之间的工作移交过程变得流畅无碍，他们通常会遇到以下三类问题：

发布管理问题：

很多企业有发布管理问题。他们需要更好的发布计划方法，而不止是一份共享的电子数据表。他们需要清晰了解发布的风险、依赖、各阶段的入口条件，并确保各个角色遵守既定流程执行。

发布/部署协调问题：

有发布/部署协调问题的团队需要关注发布/部署过程中的执行。他们需要更好地跟踪发布状态、更快地将问题上升、严格执行流程控制和细粒度的报表。Ops 所在的部门绩效分为两块：一块为常规运维绩效（保证系统稳定性），另一块为 DevOps 项目绩效（保证开发顺利性），可以根据具体工作状况来设置这样的工作比率。

发布/部署自动化问题：

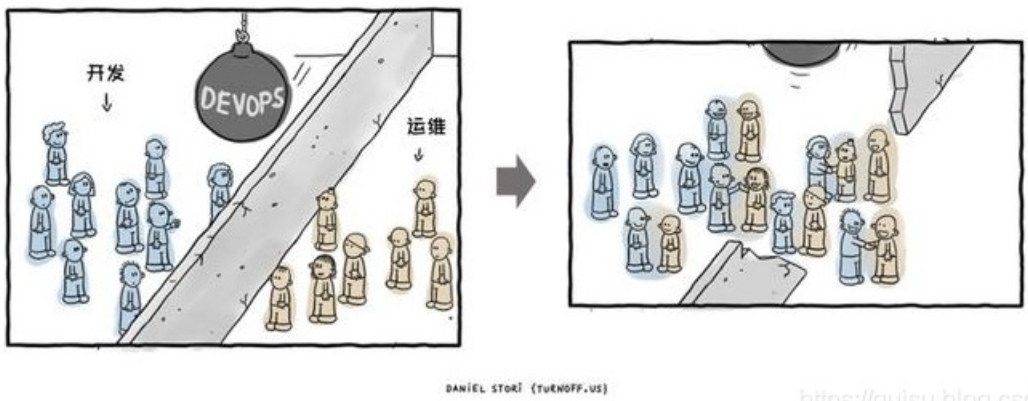
这些企业通常有一些自动化工具，但他们还需要以更灵活的方式来管理和驱动自动化工作——不必要将所有手工操作都在命令行中加以自动化。理想情况下，自动化工具应该能够在非生产环境下由非运营人员使用。

DevOps打破了开发人员和运维人员之间历来存在的壁垒和沟鸿，加强了开发、运营和质量保证人员之间的沟通、协作与整合。从而形成了一种通过持续交付来优化资源和扩展应用的新方式。DevOps和云原生相结合，能够让企业不断改进产品开发流程，更好地适应市场变化，提供更优质的服务。

1、运维人员会在项目开发期间就介入到开发过程中，了解开发人员使用的系统架构和技术路线，从而制定适当的运维方案；

2、开发人员，也会在运维的初期参与到系统部署中，并提供系统部署的优化建议。

DevOps的实施，促进开发和运维人员的沟通，增进彼此的理解。通过合力共事，在问题出现时就能一起承担，迅速解决。



2、高效快速CI/CD（持续集成/持续部署）

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。

CI/CD（持续集成/持续部署）管道可以说是实施 DevOps 的一大重要成果，可帮助企业在需要很少的人工干预的情况下，更快速、更频繁地向客户交付应用，并不断改进产品的质量，增加服务功能，实现精益求精的发展。在整个生命周期内，CI/CD都引入了持续自动化和持续监控，从而能够快速识别和改正问题与缺陷，实现敏捷开发。

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

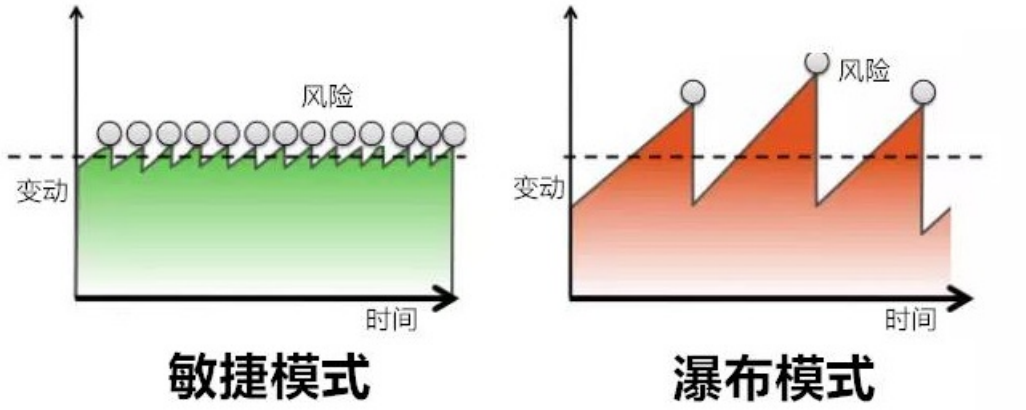
与传统的瀑布模式模型相比，采用敏捷或迭代式开发意味着更频繁的发布、每次发布包含的变化更少。由于部署经常进行，因此每次部署不会对生产系统造成巨大影响，应用程序会以平滑的速率逐渐生长。

（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。



与传统开发方法那种大规模的、不频繁的发布（通常以“季度”或“年”为单位）相比，敏捷方法大大提升了发布频率（通常以“天”或“周”为单位）。

1、更小、更频繁的变更——意味着更少的风险
2、让开发人员更多地控制生产环境
3、更多地以应用程序为中心来理解基础设施
4、定义简洁明了的流程
5、尽可能地自动化
6、促成开发与运营的协作

五、DevOps误区

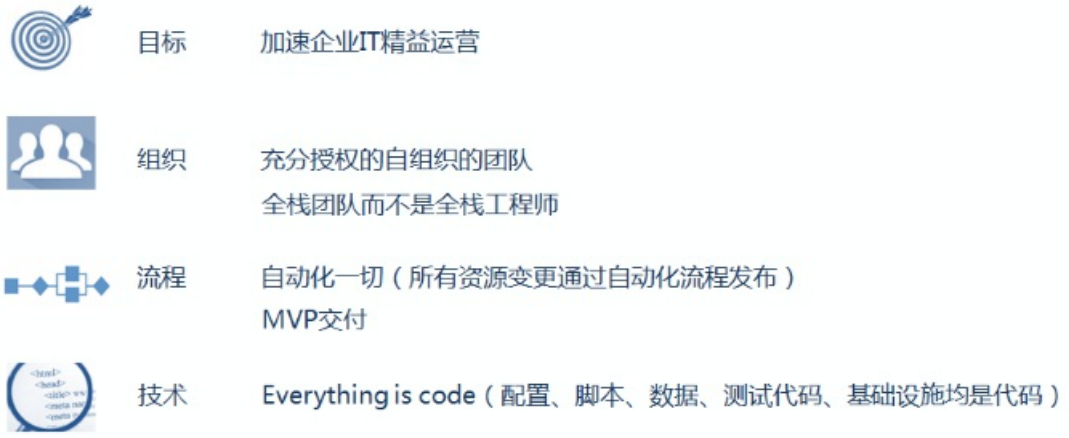
在团队使用DevOps时，存在误区是必然的。在我们同大量的客户交流中，大致有这几种误区认知：

1、没有使用云相关产品（IaaS、PaaS），组织很难开展DevOps；
2、微服务架构开发的应用适合DevOps，传统SOA应用不适合；实施DevOps和应用架构无关，无论是微服务架构，还是SOA类型应用，都可以开展DevOps工作；
3、认为将一组自动化工具的运用等同于DevOps的成功，那就太小瞧DevOps了。采用自动化工具本身不是DevOps，只有将这些工具与持续集成、持续交付、持续的反馈与优化进行端到端的整合时，这些工具才成为DevOps的一部分；
4、设立独立的DevOps团队是很多组织开启DevOps之旅的另外一个误区。事实上，如果这么做，将会导致更多的竖井。在责任没有清晰定义的情况下，成立这些团队，会创造更多的混乱，不要试图把。
5、DevOps不仅仅是自动化。毫无疑问，自动化是DevOps非常重要的一部分，但不是唯一的部分，一定程度的部署自动化往往会与DevOps混为一谈，实施DevOps需要从敏捷、持续、协作、系统性、自动化五个维度进行建设与改进。

六、DevOps如何落地

很多人可能觉得，所谓DevOps，不就是Dev+Ops嘛，把两个团队合并，或者将运维划归开发，不就完事了嘛，简单粗暴。

注意，这个观点是不对的。这也是DevOps这些年一直难以落地的主要原因。



想要将DevOps真正落地：

首先第一点，是思维转变，文化和人

首先第一点，是**思维转变**，也就是“洗脑”。不仅是运维的要洗，开发的也要洗。员工要洗，领导更要洗。

DevOps并不仅仅是组织架构变革，更是企业文化和思想观念的变革。如果不能改变观念，即使将员工放在一起，也不会产生火花。

一个中大型的系统，开发和运维这两个部门必须同时存在。开发部门不可能通过DevOps去取代运维部门，同理，运维部门更不可能通过DevOps去取代开发部门。开发和运维，两者都具有同等的重要性！因为，DevOps设计的初衷，是融合，而不是取代！

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。

其次第二重新梳理全流程的规范和标准

除了洗脑之外，就是根据DevOps思想重新梳理全流程的**规范和标准**。

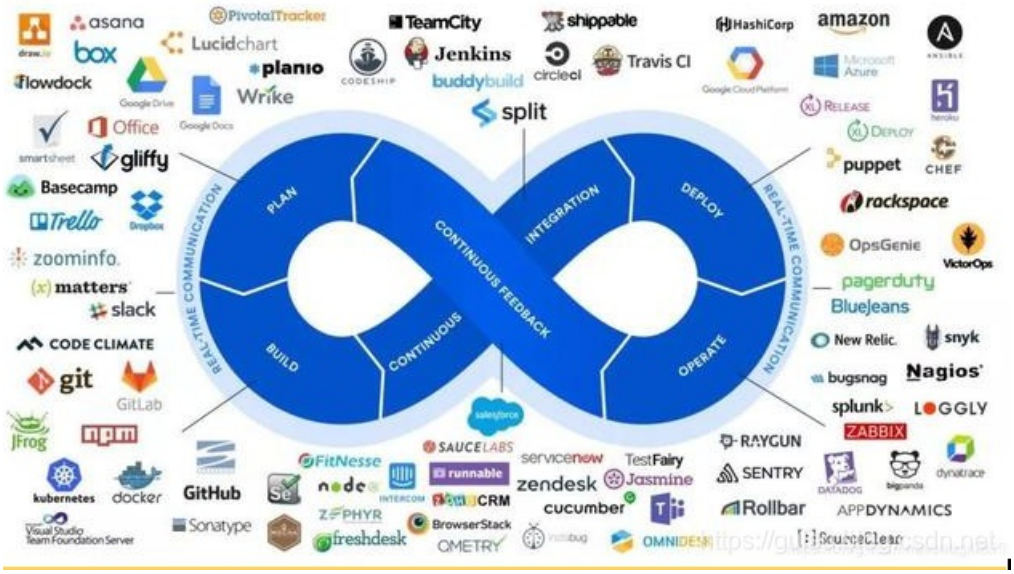
在DevOps的流程下，运维人员会在项目开发期间就介入到开发过程中，了解开发人员使用的系统架构和技术路线，从而制定适当的运维方案。而开发人员也会在运维的初期参与到系统部署中，并提供系统部署的优化建议。

DevOps的实施，促进开发和运维人员的沟通，增进彼此的理（gan）解（qing）。

最后是 软件和平台的支持：打通的工具链

在思维和流程改变的同时，想要充分落地DevOps，当然离不开**软件和平台**的支持。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。

目前支持DevOps的软件实在是太多了。限于篇幅，就不一一介绍了。话说回来，现在DevOps之所以被吹得天花乱坠，也有这些软件和平台的功劳，可以趁机卖钱啊。



DevOps生态圈中令人眼花缭乱的工具

上述这些关键要素里面，技术（工具和平台）是最容易实现的，流程次之，思维转变反而最困难。

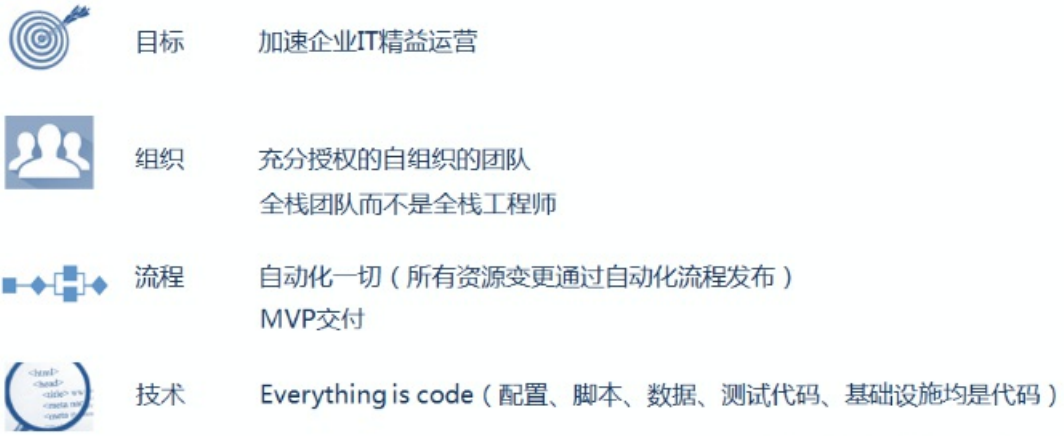
换言之，DevOps考验的不仅是一家企业的技术，更是管理水平和企业文化。

关键的 DevOps 合作方式

1，共同进行架构设计
2，共同进行技术决策
3，持续交付流水线的建立
4，共同 Pair 和 Review 代码和环境的配置
5，共同参与回顾会议
6，通过定期的内部 Session 增加相互的理解
7，共同处理运维的问题

七、DevOps具体实施

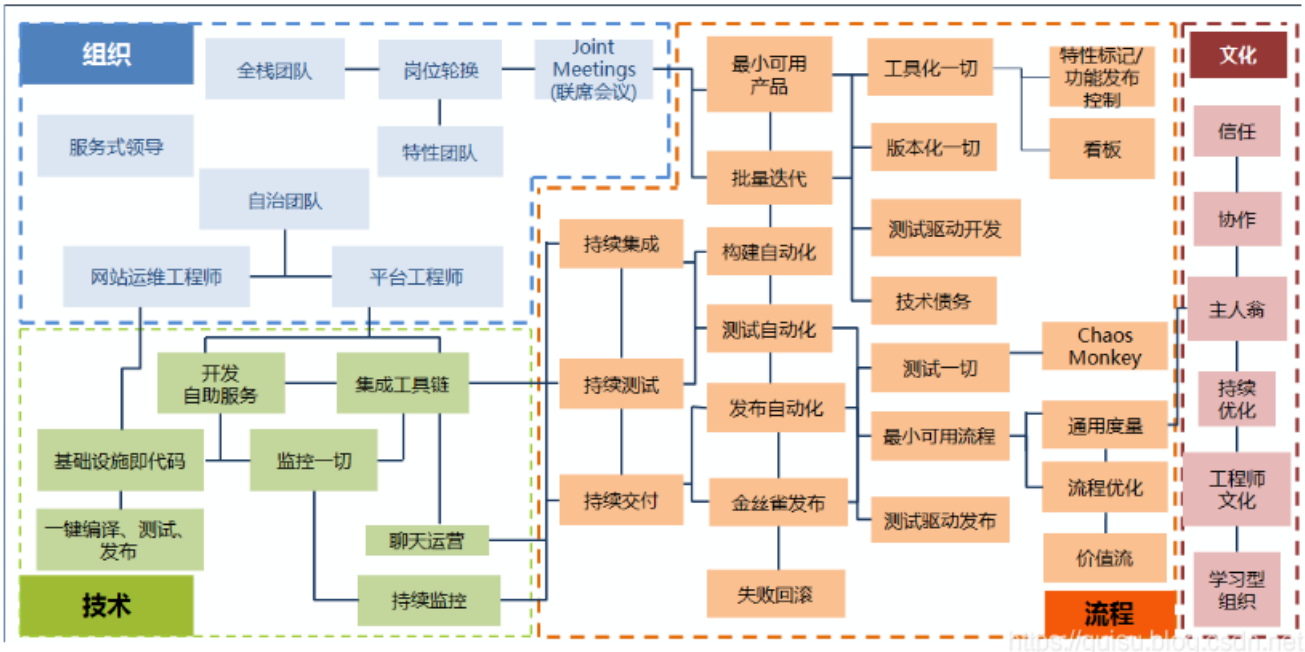
在DevOps实施过程中，团队经过总结积累，制定了团队的DevOps宣言，支撑团队从敏捷型组织转向DevOps（企业敏捷）。



DevOps企业实践

实施DevOps的核心目标是加速团队、企业的IT精益运行，从根本上提升IT的生产效率，加速部门、企业的业务创新能力。让团队从IT支撑部门，转向为IT创新部门。

实施DevOps过程中，需要从**组织**、**技术**、**流程**三个维度进行持续的优化与改进。



实施DevOps，可以参考总结的“DevOps实践模型”，从组织、技术、流程三个维度中选择关键的活动项进行最佳实践活动。

可以梳理出目前团队中欠缺但又容易改进的点，逐步将更多的实践活动纳入团队当中。团队实施DevOps的目的在于，将重复、价值低的事情交由DevOps平台实现，让团队成员做更有创新、更有价值的事情。

根据我们的实施经验，在传统企业中，技术方面的实践最容易在团队中实现、流程次之、组织的优化与变革最为艰难；大家尝试的时候，可以由易入难。

组织方面

如何实施DevOps成为众多企业迫切面临的问题，本文作者刘相，有10多年的从业经验，他结合自身企业实施DevOps的经验，梳理出DevOps在企业的组织、技术、流程等方面的最佳实践与价值，以及如何搭建DevOps平台来支撑DevOps的落地工作。

技术方面

集成工具链：打通应用应用开发工具链：需求、项目、代码、构建、测试、打包、发布、配置、监控；

基础设施即编码：将基础环境服务化、可编程化，基础设施让项目团队可以自助获取；让基础设施从物理机、虚拟机、走向容器；

一键编译、测试、部署：开发人员可以从代码开始，一键获得可访问的环境，根据需要可以推送开发、测试、预发、生产环境；

ChatOps：开发以及运营人员在内的团队成员将沟通、工具和过程整合在一起的**协作模型**。基于对话驱动开发，将工具植入对话中，保障团队能够自动执行任务与协作。最近比较流行的hubot可以认为是ChatOps的探路者。

流程方面

看板：在DevOps中不能仅仅把看板当做任务协调沟通的机制；把看板作为在制品管制平台，量化组织生产能力的工具；

MVP：采用MVP（最小可行产品）原则，快速拥抱变化。最短时间内快速交付产品原型，然后通过测试并收集用户的反馈，快速迭代，不断修正产品，最终适应市场的需求。

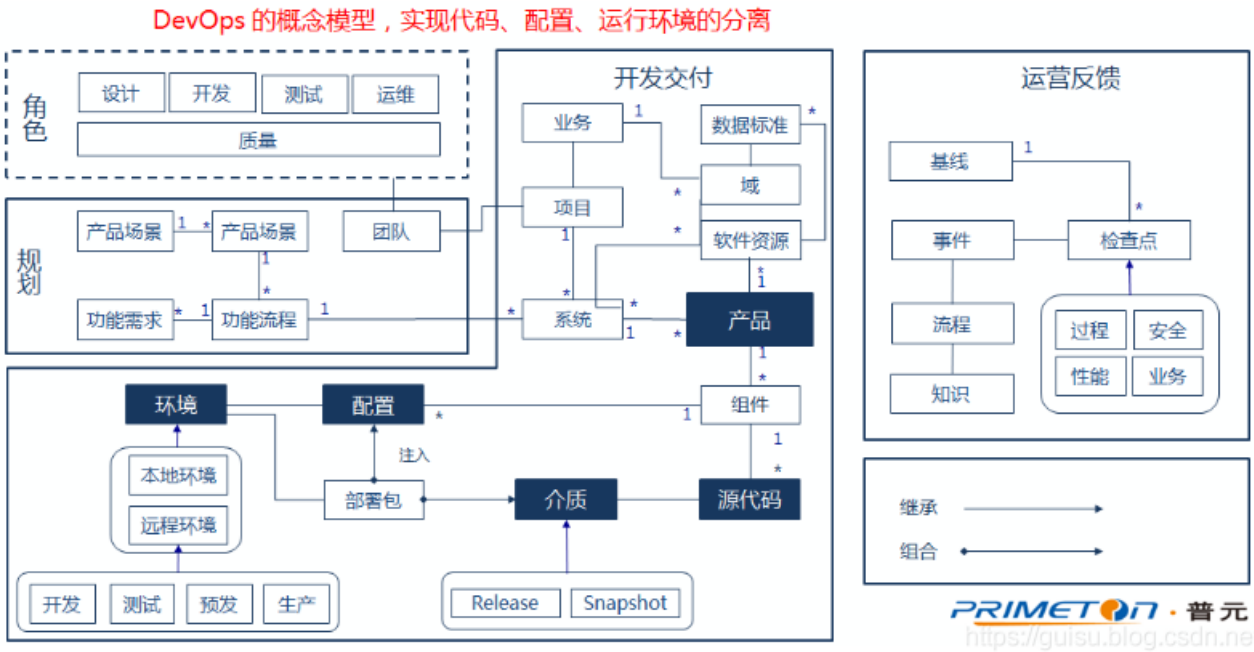
发布：建立持续发布机制，形成自动化、自助化两种能力，支持常见的灰度发布、金丝雀、蓝绿、回滚、A/B测试等；

软件度量：通过软件度量（包括过程度量、质量度量、用户度量、成本度量），推算出组织的各种有效指标；一则掌控组织的生产力水平，二则通过度量数据，反向优化组织瓶颈点；

一切皆代码：文档（用户故事、用户场景、功能特性等）、配置（应用配置、环境配置、脚本等）、环境（基础设施、中间件环境等）、发布包（二方库、三方库、部署包）需要统一看待成代码，纳入版本管理，同时建立5者间的关系，提供全视角的链路追踪。举个例子，每个发布的版本，可以追溯其对应的配置，代码、文档，发布的功能点。

组织、技术、流程三个维度中，技术、流程可以通过平台或者工具进行最佳实践的固化。

基于此，我们规划了DevOps平台，支持广义的DevOps，帮助客户快速实现DevOps建设。



平台建设第一步，梳理出DevOps的整体概念模型。从**角色、规划设计、开发交付、运营反馈**四个维度进行梳理。

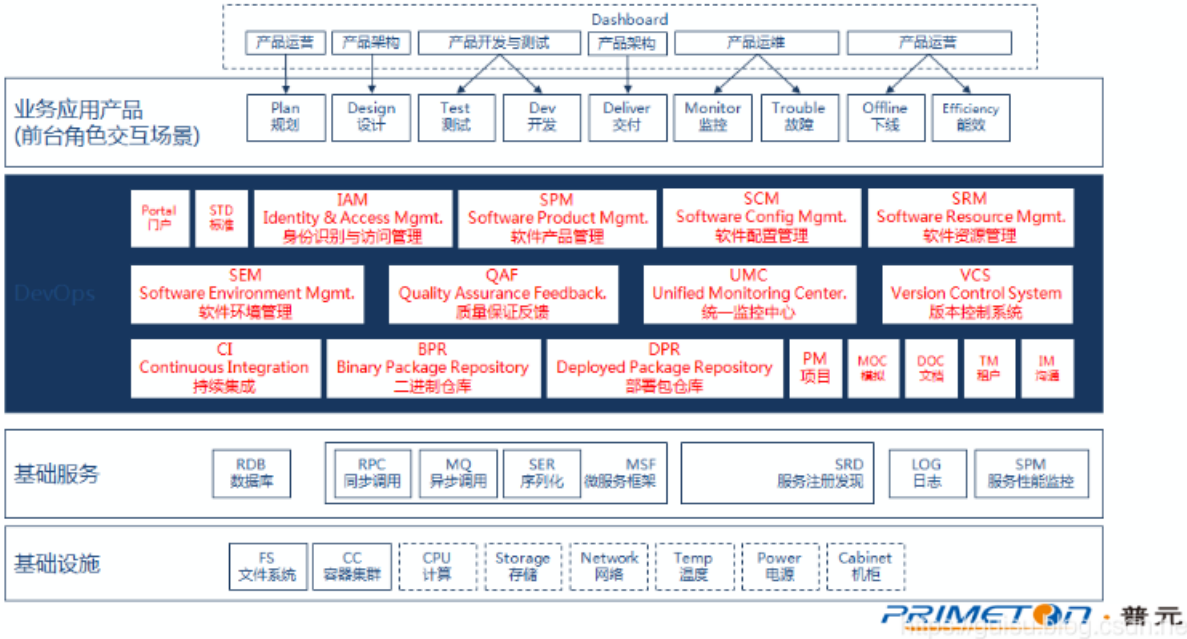
以产品为核心，将代码、配置、环境进行严格分离，同时覆盖产品全生命周期。

这里面概念看似简单，其实很多：比如：部署包=介质包+配置，这和传统的CI和CD体系就有点不一样；

再比如：环境分开发、测试、预发、生产，我们觉得即使公有云上，也应该给客户将这些做物理或逻辑隔离，因为大家的配额需求不一样，容器replication需求也可能不一样；

再比如：运营反馈，既然要做DevOps，那整个过程导出都应该可以有检查点插入，为运营提供有效数据，我们把检查点至少分成了四类，包括过程的、安全的、性能的、业务的。

DevOps架构支撑



基于领域模型梳理DevOps平台业务架构，目前共建设18个领域系统来支撑，比如：软件产品的管理、软件各阶段环境的管理、质量的管理、部署包、二进制包的管理、资源管理、监控中心、认证中心等。

每个领域系统严格按照AKF扩展立方体的Y轴进行拆分，采用微服务架构模式进行平台建设。

“DevOps业务架构”，是我们基于对企业IT管理的理解，所进行的平台化设想。从图里还可以看到，红色字部分，是我们对现有DevOps的落地实现。

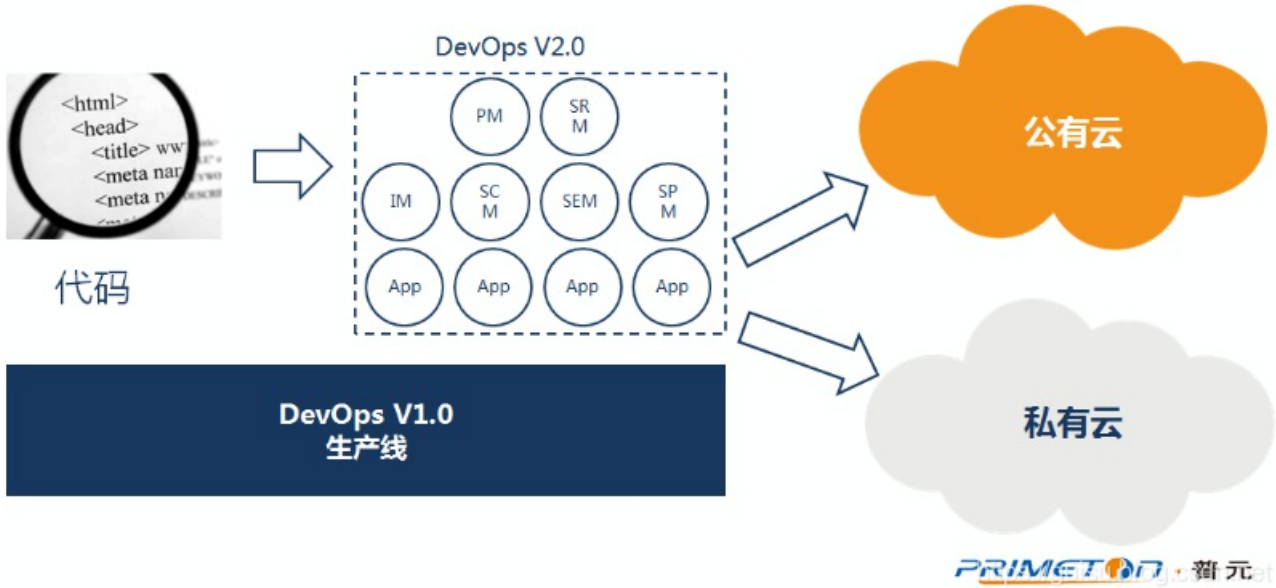
Portal（DevOps门户），自研，提供给用户使用的统一操作门户，包括用户管理、产品看板、产品全生命周期（设计、开发、测试、预发、生产、监控、故障处理）管理等；
IAM（身份识别与访问管理），自研，提供用户身份识别和访问控制的能力，包括用户管理、Token管理和用户授权等功能；
SPM（软件产品管理），自研，提供产品、组件的基准定义和管理能力，包括产品类型、产品管理、组件管理、依赖产品管理及产品投放市场等功能；
SCM（软件配置管理），自研，提供产品、组件配置管理能力，包括配置项的定义和在各个不同环境下的配置信息的管理维护能力；
SRM（软件资源管理），自研，提供产品和组件自动编译、打包和部署的能力，提供部署模板管理，支持编译和部署流程编排，编译和部署进度跟踪以及日志查看；
SEM（软件环境管理），自研，提供租户和产品环境资源配额、负载均衡，以及运行容器的管理能力，包括租户可用资源的配额，以及基于租户资源的产品和组件在各种环境下的资源配额（如开发环境、测试环境、生产环境等等）和负载均衡；同时，还提供运行容器的创建、销毁、调度、复制以及持久化卷管理等能力；
QAF（质量保证反馈），自研，提供产品的质量管理和监控能力，包括测试用例管理、缺陷管理、质量监控等；
UMC（统一运维中心），开源集成、借鉴自研相结合，提供统一的监控、预警、故障处理等能力，包括系统日志和业务日志的监控，产品的资源使用情况和运行情况监控，故障定位等。
VCS（版本控制系统），开源集成，主要以GitLab为核心，不直接提供GitLab的原生界面，所有功能在统一的DevOps上提供；提供源代码库管理的能力，包括代码库的创建、维护，分支的管理和用户权限控制等；
CI（持续集成），主要以Jenkins为核心，使之成为以API为主要使用方式的服务，提供持续集成任务调度和执行的能力，包括集成任务管理、编译、打包等；
BPR（二进制介质仓库），开源集成，主要以nexus为核心；提供二进制包仓库的管理能力，包括二进制包、文档等编译产物的上传、下载和存储访问等；
DPR（可部署介质仓库），自研，主要存储可部署的介质，其主要区别是注入了与环境相关的配置（这种部署模型是很适合没有上Docker或者容器，以虚机为主的IT基础设施或者物理机）；
PM（项目管理），自研，可以与常见的PM管理工具对接与集成，提供产品的开发过程的管理和协作的能力，主要包括：任务计划、人员分工和过程跟踪、看板等；
MOC（API模拟），开源集成，为REST API调用提供模拟能力，以便产品或组件在开发调试期间可以脱离依赖、减少阻塞、单独运行，支持根据Swagger和Mock数据发布Mock Rest Service，支持用户私有的MOCK数据；
DOC（API文档），开源集成，提供REST API/SPI文档的自动生成能力；
TM（租户管理），自研，提供租户管理的能力，包括租户管理、邀请码管理和租户配额等功能；
IM（即时沟通），开源集成，提供产品设计、开发、测试、运维等相关人员间的协作沟通能力，支持群组聊天、离线消息推送、聊天记录查询和导出；

逻辑架构整个DevOps平台分为三层：

基础设施层：包括IaaS，CaaS，我们分别是基于OpenStack和Kubernetes、Docker的，上层有一层不同环境的适配；

基础服务层：包括服务管理与调度的基础能力，如注册中心，编排，伸缩漂移；还有一堆具体的企业级或互联网式的云服务；

DevOps层：更多的是工作流程（需求、设计、开发、测试、发布等）的串接，看板等文化的体现；



在整个平台研发过程中，采用了是自己开发自己的模式，即使用上一个发布的平台作为生产线，支撑下一个版本的产品研发工作。自己交付自己可以带来两点好处：

1. 平台交付客户前，自己先把可能的坑趟掉；

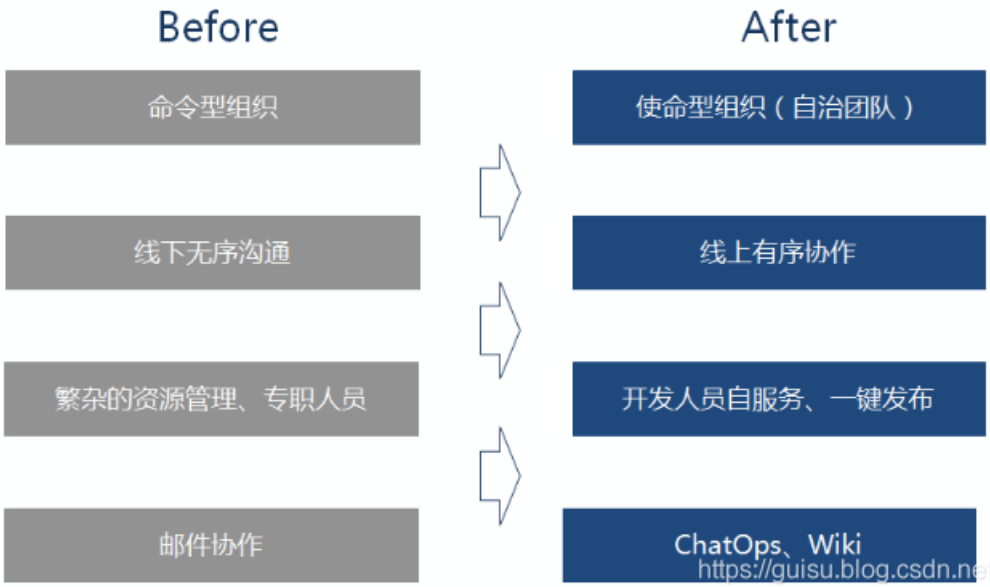
2. 当前生产线所有不能满足的功能，视作下一版本的需求（实际操作过程中，我们仅允许使用wiki作为辅助工具来支撑生产线未满足的需求）；

所以可以拿一些数字估算一下当前的规模。在研发过程中，把DevOps视为一套业务平台，目前规划的领域有18个，如果每个领域中再有多个以微服务架构落地的系统进行支撑，预计总共支撑DevOps的系统，就会超过50个。同时提供Mock、开发、测试、预发、生产5类环境（每类环境中可能还会有多套，比如集成测试、性能测试、全链路测试）。

当前版本的DevOps，整体的部署规模将超过200个集群，部署的进程实例总数也会轻松超过500个。需要注意的是，500这个数字，还没包含技术平台中的一些分布式中间件，比如缓存、消息队列等等集群。

不过，500映射到企业内IT人员自己用的平台，这个数字，对于不同的企业，可能是个天文数字，也可能只是九牛一毛。

实施DevOps价值



在部门实施DevOps之后，我们团队有显著改变：

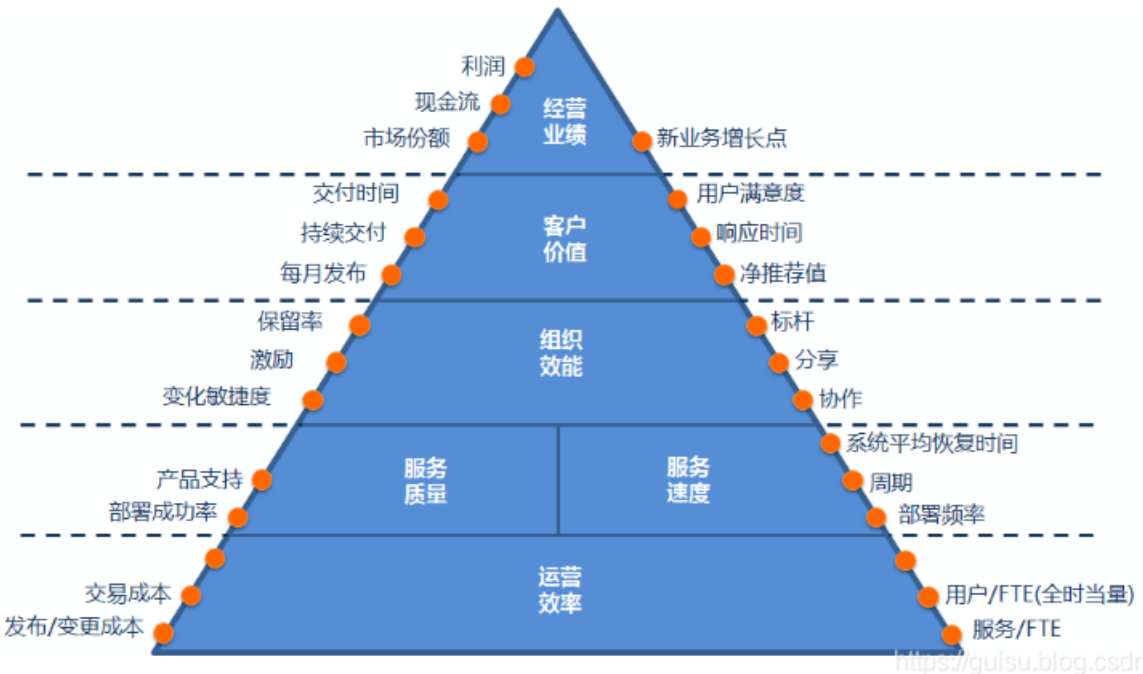
? 在团队组织上，每个团队小而自治且是全栈团队，沟通、技能互补，每个团队负责独立的领域系统，目标感非常明确，团队在走向使命型组织；

? 项目的从原先线下协作、沟通，统一到统一的DevOps平台上协作、沟通；团队成员可以随时了解项目进展全貌，利用平台可以做到各种过程数据的实时收集（举例，比如需求变更、任务延期等）；

? 资源管理由原来专职人员，过渡到开发人员实现自助化服务，可以按需实现各类环境申请与开通，基础设施即服务提供来技术的支撑；

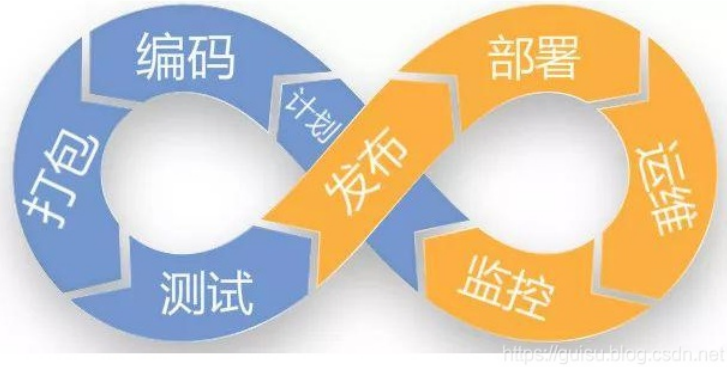
? 从原来的邮件文化，到DevOps平台统一沟通，同时DevOps打通多个工具链路端，任务分发、沟通、提醒可以实时推送；

最后给大家奉上DevOps成熟度评判指标，在践行DevOps时，可以从运营效率、IT服务水平、组织效能、客户价值、经营业绩五个维度进行评判，持续优化与改进。



八、DevOps生命周期内的工具

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：代码开发和审阅，版本控制工具、代码合并工具

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：通过测试和结果确定绩效的工具

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：变更管理、发布审批、发布自动化

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。

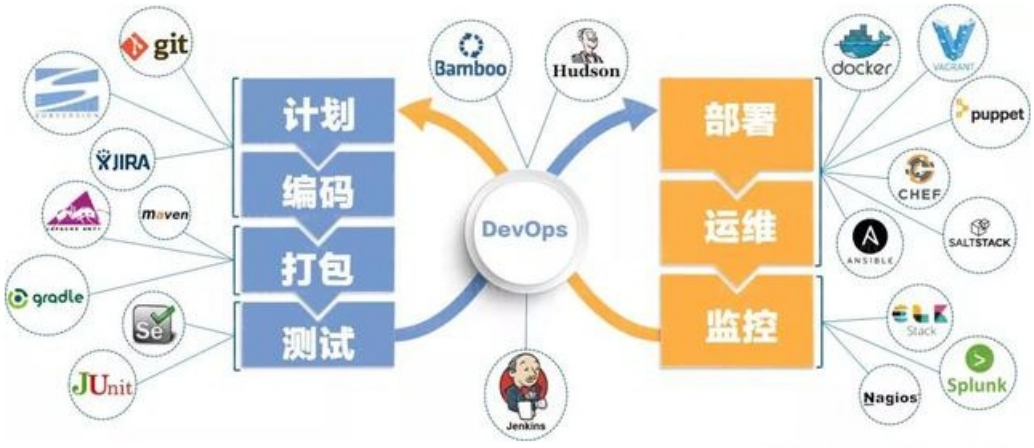
容器化工具在部署阶段也发挥着重要作用。**Docker和Vagra**nt是流行的工具，有助于在开发、测试、登台和生产环境中实现一致性。除此之外，它们还有助于轻松扩展和缩小实例。

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、zabbix。**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。

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）

九、DevOps与虚拟化、容器、微服务

这几年云计算技术突飞猛进，大家应该对虚拟化、容器、微服务这些概念并不陌生。当我们提到这些概念的时候，也会偶尔提及DevOps。

它们之间有什么联系呢？其实很简单。

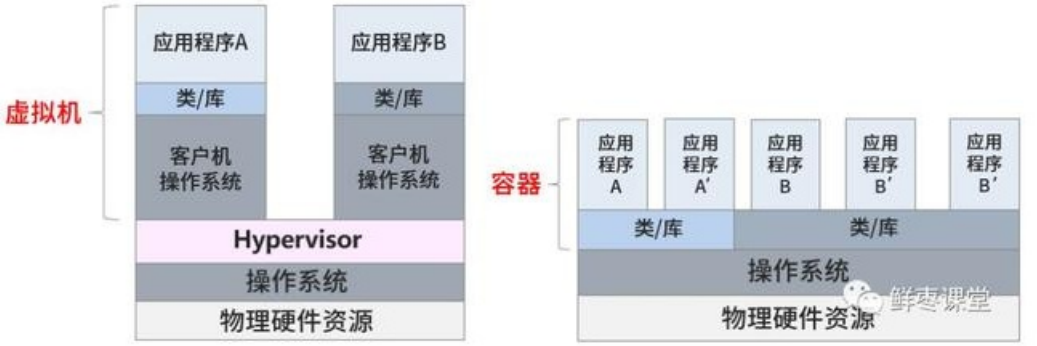
大家可以设想一下，如果要对一项工作进行精细化分工，我们是对一个大铁疙瘩进行加工方便？还是拆成一块一块进行加工更加方便？显然是拆分之后会更加方便。

所谓“微服务”，就是将原来黑盒化的一个整体产品进行拆分（解耦），从一个提供多种服务的整体，拆成各自提供不同服务的多个个体。如下图所示：

单体式架构（Monolithic）→微服务架构（Microservices）

微服务架构下，不同的工程师可以对各自负责的模块进行处理，例如开发、测试、部署、迭代。而虚拟化，其实就是一种敏捷的云计算服务。它从硬件上，将一个系统“划分”为多个系统，系统之间相互隔离，为微服务提供便利。

容器就更彻底了，不是划分为不同的操作系统，而是在操作系统上划分为不同的“运行环境”（Container），占用资源更少，部署速度更快。



虚拟化和容器其实为DevOps提供了很好的前提条件。开发环境和部署环境都可以更好地隔离了，减小了相互之间的影响。