redhat.

电子书

教大象跳舞

跨团队、流程和应用的主动变革

Burr Sutter，*开发人员体验总监*
Deon Ballard，*内容营销员*

应用业务已转移到 IT 部门之外。如今已既成事实的是每家公司其实都是软件公司，快速向客户提供新服务和新功能是公司的关键竞争优势之一。IT 敏捷性是有些初创公司能战胜巨头的基石（就像圣经故事中年轻的大卫击败非利士巨人歌利亚一样）

曾几何时，几代人以前（技术时代），IT 部门作为一个内部的部门，其着眼点是维护公司内部的基础架构和服务。部分公司可能有一些面向外部的服务，尤其是 Web 服务，但通常也仅限于较为狭窄且受限制的领域。IT 部门不属于创收或战略部门，只是一个支持环境而被视为成本中心。

对于以基础架构为中心的环境而言，其结果之一就是开发人员对自己代码的用途失去了感知力。产品的发布周期较长，而更改速度较慢。在开发人员完成相关工作后，代码将进入测试或运转流程，然后要在数月之后才能发布。由于前期准备时间长，工程师失去了作为开发人员的乐趣，也就是做出创造性工作并见证它在现实生活中发挥作用的乐趣。

数字化转型以及相关的文化和技术变革（例如 DevOps）所引发的巨变之一，就是重新带来了创建代码的乐趣。开发人员可以创建代码并实际看到运行。这是一种巨大的转变。使得代码创建的即时性得到重现。由于能实时查看应用，从而为开发人员提供了一个反馈回路，藉此可以让他们重新设计和改进代码，并实现项目的快速推进。

如今，您的企业的境地就跟大象一样。传统环境与由微服务和 DevOps 支持的现代环境之间存在着诸多变化。一些企业有资本从头开始进行部署，但对于许多的企业而言，挑战在于如何教会笨拙的大象可以像灵活的芭蕾舞演员那样跳舞。

这种变化意味着什么？

数字化转型是企业的一种战略性变革。促使公司在竞争压力发生变化或新法规出台时能够调整核心服务，在发现漏洞后可以立即推出更新。

但是，除了“改变现状”之外，还没有数字化转型的通用定义。数字化转型一词有时用于表示新的架构（如微服务）、新的流程（如 DevOps）或新的技术（如容器和应用编程接口（API））。当某个概念能够表示任何事情时，实际上是毫无意义的。数字化转型并不是某个能够实现的具体目标。这是一种每个企业都必须为自己量身定制的东西。

没有任何一个架构模式或技术平台可以完美地应用于各种环境。成功实现数字化转型的企业是能清楚地了解自己目标和文化的企业，而且每个企业看起来也各有不同。例如：

- 沃尔玛在“黑色星期五”部署了代码，而那个时段有 2 亿人在线。[1]
- 亚马逊每秒都会在数百个应用和数百万个云实例中部署更新的代码（每年 5000 万个）。[2]
- Etsy 每天对一个单体式应用进行 60 次部署。[3]
- Netflix 每天在一个复杂的分布式架构上完成数百次部署，目的只是将一段代码在 16 分钟内从检入更改为生产。[4]

这些企业都采用了非常不同的团队结构、底层技术、代码库和架构。关键点不在于他们专注于某种模式或某项技术，并且让一切平稳运转。重要的是，他们都是从团队、当前技术债务及业务策略的评估入手，然后专心致志地按照既定方向将企业向前推进。最终，他们得到了想要的结果。

这就是教大象跳舞的过程。借助明确的目标和策略，无论您的企业现在处于什么状况，都可以不断克服困难（技术债务、设计缺陷等等），实现所设想的进化。这意味着您必须清楚地了解自己所要完成的任务以及与现实之间的差距。

注意到房间内的“大象”

评估当前的技术前景并完善业务策略绝非易事。往往很难得到这种视角。有一个众所周知的寓言是“盲人摸象”：六个盲人遇到了一头大象，每个人分别去摸不同的部位，试图全面了解这种新的动物。一个人摸到了象鼻，认为它像一支矛；另一个人摸到了象身，认为它像一堵墙；还有一个人摸到了耳朵，认为它像一把蒲扇。有趣的是，根据来源不同，这个寓言也有多个不同的结尾。有的说，这些人无法接受其他人的描述，因此闹翻了。还有的说，有个明眼人出来给大家描述了大象的整个外观，从而统一了他们的看法。

不同的结尾可能正是这个寓言最现实的部分。故事的重点在于每个人都有不同的观点、有限的信息，以及基于上述观点的假设。这些不同观点所引发的结果说明了团队内部的内在沟通和关系——有些可以跳出自己的观点局限，有些则分崩离析。我们所看到的取决于我们是谁、我们在哪里、我们要寻找什么、我们知道什么，以及我们不知道什么。

问题是，现在大多数企业拥有的不止是一只大象，这让情况变得更加复杂。Netflix 是一家独角兽企业，当然这不仅仅是因为其先进的微服务架构以及基于容器的测试和部署环境。它有能力根据业务策略建立团队流程和技术体系，而不会带来技术负担。这是一个初创公司。

任何采用传统应用（和传统团队）的企业，其房间内不止有一只“大象”，其中包括：

- 当前的团队结构和沟通模式。
- 开发、测试、构建和发布流程。
- 技术债务和现有的商用应用。
- 部门之间的目标或战略愿景不同。

---

1 *Cian O'Maidin：“为什么 Node.js 成为企业的首选技术”，NearForm，2014 年 3 月 10 日，www.nearform.com/blog/node-js-becoming-go-technology-enterprise/。访问时间：2017 年 9 月 1 日。*

2 *Joe McKendrick：“亚马逊如何做到每秒处理一次新的软件部署”，ZDNet，2015 年 3 月 24 日，www.zdnet.com/article/how-amazon-handles-a-new-software-deployment-every-second/。*

3 *“卓越的微服务与单体式应用 Twitter Melee”，High Scalability（高可扩展性），2014 年 7 月 28 日，http://highscalability.com/blog/2014/7/28/the-great-microservices-vs-monolithic-apps-twitter-melee.html。*

4 *Ed Bukoski 等，“我们在 Netflix 是如何构建代码的”，Netflix 技术博客，2016 年 3 月 9 日，https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15。*

这些都是大象，而每个利益相关者都可以从不同的角度看待这些大象。

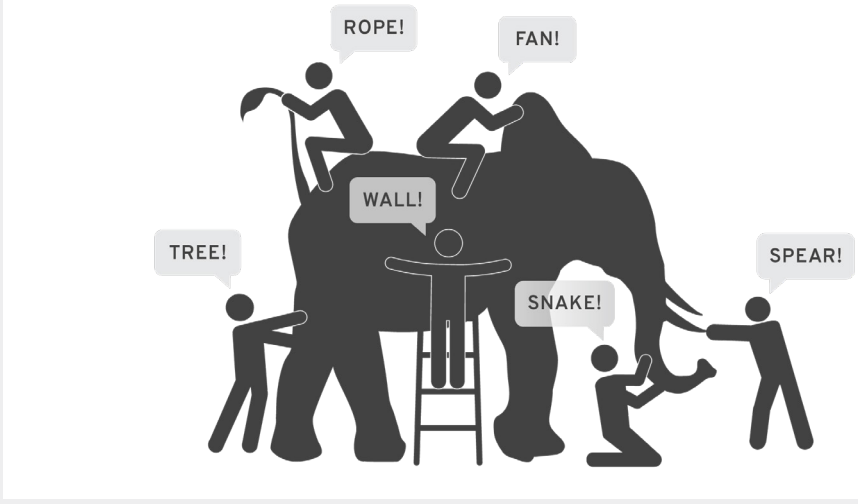

图 1. 盲人摸象

数字化转型的达尔文进化论观点

有一种趋势是将 IT 或数字化转型计划视为二选一：要么做某件事，要么做另一件事。但由于以下几种原因，这种观点可能站不住脚：首先，有时您可以选择做多件事或实施混合解决方案。其次，您经常面对的不是改变单个因素——不同类型的改变可能需要不同的基础性变革，有时可能还取决于文化和流程的改变。

罗马不是一日建成的

如果将数字化转型视为一个逐层推进、环环相扣的连续过程，可能会更具建设性。

图 2. “数字进化论主义”的各个阶段

微服务是一种必需的状态吗？

鉴于维护的复杂性，数字化演进的最后阶段是微服务。但是，企业演进的最后阶段一定要是微服务吗？

未必

如果演进的其他方面都到位，那么单体式架构仍可以每周发布一次，并使用高级部署技术、CI/CD 和分布式可扩展基础架构。

只有在团队规模较大且需要比按周发布更快的速度，或需要按不同的时间表进行发布时，才应考虑微服务。小型团队或小型代码库不需要分解为微服务代码库。

DevOps 与流程改变

数字进化的基础是 DevOps。和业务策略一样，应用也体现了创建过程中团队及沟通的情况。DevOps（或类似的流程改变）让更多的利益相关者参与到开发讨论中，并提供了有关运营团队如何维护软件和基础架构（以及客户和合作伙伴如何实际使用这些应用）的深入见解。DevOps 在团队之间建立了更紧密的反馈回路，并且需要开诚布公的交流。这种开诚布公的交流是其他演进阶段的基础。

自助式基础架构

该阶段是一种以技术为中心的变革，旨在引入通常与现代化技术平台相关联的高效率。容器和自助服务目录使开发人员、测试人员和运营团队可以快速启动一致的环境。在有些企业中，新实例的交付时间已从几天缩短到几分钟。为什么要让技术人员等待几天才能获得计算资源？

构建自动化和编排

构建自动化涉及两方面的改变。从技术角度看，可以采用红帽® Ansible 或 Puppet 等高级部署引擎，但也需要流程的改变。许多企业都围绕变更和风险管理制定了严格的流程。如果不对这些流程采用更敏捷的方法，就无法充分利用新的技术。

持续集成/持续交付 (CI/CD) 管道

持续交付就是承诺以快速、迭代的方式交付软件变更。管道的思路是：同时采用流程和技术手段，减少不良代码（或损坏的代码）从制作到部署的风险。它反映了先前步骤（DevOps 以及团队之间的开放式交流、围绕测试和构建的流程，以及自动测试与部署）的成熟度。待所有这些阶段都确定之后，就有可能快速推送代码。这就是管道。

高级部署路径

一旦用于快速部署的流程和基础架构就位，便可以使用部署系统来减轻更新所带来的任何风险，评估功能的有效性，并为新想法提供实实在在的测试基础。这其中可以包括在部署过程中（蓝绿部署）采用独立的环境并在它们之间进行负载平衡，使用两个不同的环境来测试用户交互（A/B 测试），或向少数用户推出更新并安全地增加用户数量 （金丝雀发布）。

微服务（或分布式系统）

微服务是一种执行离散、功能单一的小型应用。整个应用架构可能需要执行数十个或数百个不同的功能，而且每一项功能都要在微服务中定义和编排。微服务架构（或任何分布式计算架构）既复杂又简单。单个服务更为简单，更易于维护、添加和停用，但总体架构却更为复杂。如果做法得当，“基于微服务的设计就会成为您所学优秀应用设计的最终体现”。[5] 这种高度分散型架构可以更轻松地进行扩展，更容易引入新的服务或更新，并且降低了系统范围故障的风险。正是由于这种架构内的弹性，使得微服务与一些颠覆性创新公司（如 Netflix 和 Google）存在广泛关联。

5 Ben Cotton：“从单体式应用到微服务”，2017 年 1 月 3 日，https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/。

“CI/CD 可提高交付速度和频率。为交付内容提供支持的是 DevOps。” [9]

- BURR SUTTER

“您无法实现持续交付：” 康威定律，DEVOPS 与文化

在 2016 年 DevNation 大会上，Rachel Laycock 就曾表示“软件的‘实施’非常难”。[6] 她分享了一个关于未能在一家大型金融服务公司实施持续交付的故事，那家公司正遭受数周才能推出更新（甚至是关键更新）的困扰。公司认为，解决办法就是转向持续交付。经过六个月反复尝试更改流程之后，她终于归来并告诉一位副总裁：不，他们无法实现持续交付。

部分问题出在技术和架构上。这家金融服务公司拥有一个含有 7,000 多万行代码的代码库，并采用了深受怀疑的“大泥球”（Ball of Mud）架构。但是，软件却被放在次要位置。这是一个显而易见的问题。房间里真正的“大象”体现在企业对改变各个部门行为的无能为力。同时阻碍了公司为改变所付出的努力。

文化先行

值得注意的关键点是达尔文（软件）进化过程不只是技术的改变。流程和人员的改变以及基础架构的改变交替发生，而文化的革新更是首当其冲。正如 Laycock 在其会议发言中所说的那样：[7]

“你可以创建所需要的最漂亮的架构图。一旦涉及到诸如人员、流程等因素，就必须打造一个支持持续交付和架构规范的 [文化] 环境，因为流程或结构的改变实际上并没有带来持久的改变。人们并不遵守规则。因此，房间里真正的‘大象’是康威定律。那又意味着什么呢？传统企业结构会一次次摧毁您的精美的架构。”

要想将软件或技术视为一种进化性的变革，关键是要认同演变是环境的自发功能。对于企业而言，则体现在文化上。管理层可以支持进化性变革所需要的改变，但这些改变不能由管理层来拍板决定。人们需要有改变的主观愿望。这体现的是自由意志，而不是强制。

Gartner 实际上有这方面的数据：“有 90% 尝试使用 DevOps 、但没有针对性地解决自身文化基础的企业会走向失败。”[8]

更改基础架构或应用架构很容易。为了有效地改变自己的工作状况，您需要首先改变自己的文化。

康威定律指出：“任何采用系统设计的企业，其设计结构最终都会不可避免地与该企业的沟通结构相一致。”针对这方面，有两个彼此相关的解释：

- 除非也改变沟通结构，否则单纯改变架构或基础架构不会带来任何改变。
- 无论基础架构如何，改变沟通结构都会带来流程和基础架构的改进。

康威定律指出：“任何采用系统设计的企业，其设计结构最终都会不可避免地与该企业的沟通结构相一致。”[10]

DevOps 是万里长征迈出的第一步

敏捷方法是一种软件设计方法，它试图将所有参建方（包括 QA、产品管理、开发人员，甚至文档编写）都纳入一个更具凝聚力的团队。其思路是：通过精简整合，专注于以用户目标（称为案例）表示的具体任务，以此来明确目标。这样就打破了传统的瀑布式软件开发方法，即一种类似分遣队的开发方法，任务将由一个团队移交至另一个团队。

6 *Rachel Laycock：“持续交付”，下午的会议。红帽峰会 - DevNation 2016，2016 年 7 月 1 日，美国加利福尼亚州旧金山。https://www.youtube.com/watch?v=y87SUSOfgTY*

7 *Rachel Laycock：“持续交付”，下午的会议。红帽峰会 - DevNation 2016，2016 年 7 月 1 日，美国加利福尼亚州旧金山。https://www.youtube.com/watch?v=y87SUSOfgTY*

8 *“Gartner 强调实现敏捷 I&O 文化的五个关键步骤”，2015 年 4 月 20 日，www.gartner.com/newsroom/id/3032517。*

9 *DevNation Federal，2017 年 6 月 8 日，美国华盛顿特区，https://www.youtube.com/watch?v=tQOo2qaUc6w&t=1s*

10 *Melvin E. Conway（1968 年 4 月），“How do Committees Invent?”，Datamation*

“团队设计是您架构的初稿。”

– MICHAEL NYGARD，RELEASE IT!

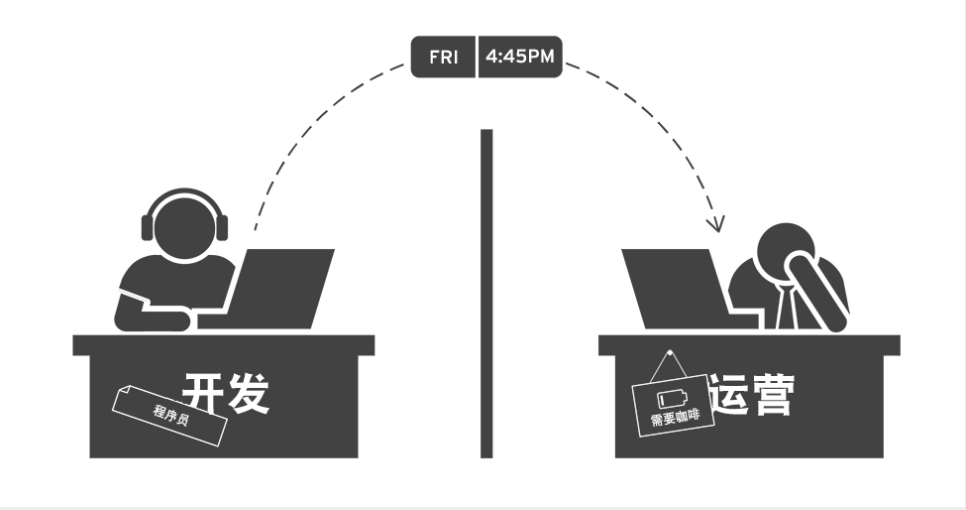

图 3. 部门壁垒 - DevOps 之前

但是，这仍然只解决了应用实际生命周期的半数问题。一旦某个应用完成开发，就会将其交给运维团队来部署和维护（通常在周末的维护期内）。

问题在于，运维团队并不总是知道应用的具体用途是什么，这意味着他们可能无法进行有效的部署。开发人员可能对实际的操作环境一无所知，并可能会创建无法在生产环境中执行的应用。再者，为了降低变更的风险，许多企业都制定了繁琐的变更管理流程，以试图解释并证明任何变更的合理性。

DevOps 是一种文化的转变，努力打破开发人员、运维团队和业务利益相关者之间的隔阂。这种隔阂是真实存在的，但也是人为造成的。一个团队中可以不止包含工作职能相同的人员。DevOps 试图重新定义团队，将应用生命周期中涉及的每个人员都纳入其中，并在这些团队之间实现有效沟通。

相比 DevOps、敏捷开发或其他方法，文化的改变具有更加深远的意义。这是一种将每个人都融入到同一团队的承诺。如果您更改了沟通方式，结果也会随之而变。

一些重大更改有时是从非常简单的步骤开始的。文化上的改变是所有技术和流程改变的基础。如果您在努力建立 DevOps 文化，不妨尝试以下两种方法：

- 让您的开发人员与运维团队共度周末，观看产品发布过程并了解他们的工作。
- 了解开发人员为请求新的虚拟系统所需经历的步骤或服务单。

在原地观察其他团队的工作情况可以有效地鼓励团队改变其流程或进行开诚布公的交流。

PUPPET的DEVOPS现状报告亮点

- 前期准备时间缩短 2,555 倍
- 部署次数提高了 200 倍
- 故障恢复速度加快了 24 倍
- 变更失败率为原来三分之一
- 返工时间减少了 22%

图 4. 跨团队沟通路径的全景图

Puppet 的 DevOps 现状报告揭示了改变团队结构（和沟通）的有效性。[11] 其研究表明，DevOps 团队在以下方面发生了明显的改观：

- 前期准备时间缩短了 2,555 倍。
- 部署次数提高了 200 倍。
- 故障恢复速度加快了 24 倍。
- 变更失败率只有原来的三分之一。
- 返工时间减少了 22%。

速度是 DevOps 的主要优势之一。从整个发布过程看，所有这些不同的团队都要参与进来，从而将应用推向市场。同时，取决于流程、基础架构和代码库的干净程度，发布过程可能还会花费更多的精力。

11 Gene Kim 等，“DevOps 现状报告”，Puppet，2016 年，https://puppet.com/resources/whitepaper/2016-state-of-devops-report。

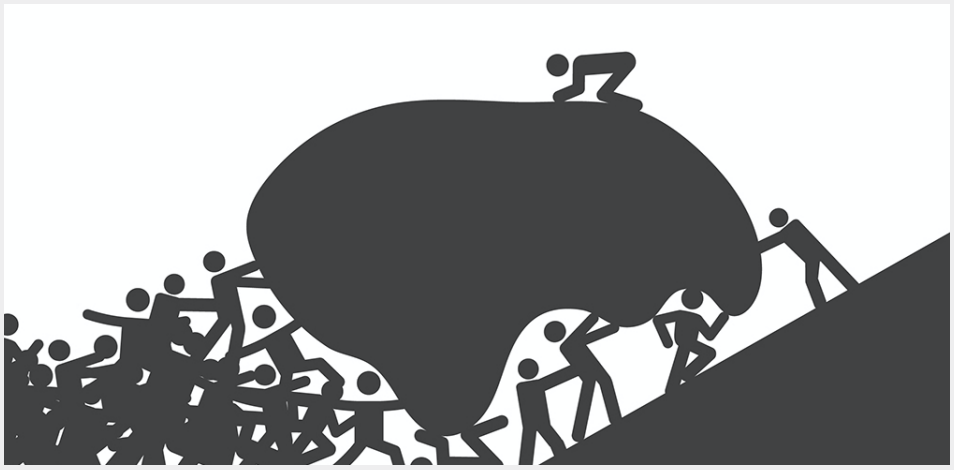

图 5. 单个应用发布的团队合作

如果发布频率相对不高，这种痛苦还是可控的。但是，一旦软件成为业务的推动要素，整个发布过程就必须每年进行多次。（在像亚马逊这样 具有高度创新性的公司中，发布过程每天可能要进行数百次。）在这种情况下，由多个团队通力将应用投入生产的现象就会反复发生。

图 6. 发布单个版本后，还会发布更多版本

DevOps 将紧急、全力以赴式的工作方法转变为在团队规划、开发、测试和部署中实现更平稳、更可持续的迭代。这就是 DevOps 团队在效率方面实现天文数字式增长的原因所在，因为整个生命周期都是可重复的。

无论您的最终应用架构是更完善的单体式应用还是分布式微服务，都需要改变团队结构和沟通方式。无论是在规划应用前还是在部署应用后，都必须确保沟通的顺畅。除非建立并启用支持开放式沟通和反馈的文化，否则按照不同的服务来划分团队很容易造成公司分化成一个个孤岛。[12]

应用架构设计：关注微服务

新的应用架构是数字化演进的最后阶段，但它通常是房间内中最引人注目，也是最容易识别的“大象”之一。即使首当其冲的是改变平台和流程，架构也是值得关注的重要因素。

设计持久度、技术债务与策略

不知道该如何处理当前的应用，是企业变革的一大阻碍。这就是许多数字化转型计划考虑采用“淘汰和替换”方法的原因之一；有些时候，重新开始似乎更容易。

但是，问题在于技术债务是设计的结果。在不清楚要替换什么的情况下拿掉一个应用，意味着同一个不被接受的架构最终还会再次出现。

正如 Rachel Laycock 所说：“希望不是一种设计方法。您必须要非常用心地去做。”[13]

您所开发的一切都是您的技术债务。对于非常新的应用，团队经常会在没有明确设计的情况下开始开发工作，这不一定是坏事。Martin Fowler 对设计持久力和技术债务的描述中指出：如果开始时没有明确的设计，通常反而能够更快地实现初始创新。[14] 不过，也有观点认为，好的设计造就稳定发展，而没有设计则平淡无奇。

12 Ben Cotton，“从单体式应用到微服务”，2017 年 1 月 3 日，https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/。

13 Rachel Laycock：“持续交付”，下午的会议。红帽峰会 - DevNation 2016，2016 年 7 月 1 日，美国加利福尼亚州旧金山。https://www.youtube.com/watch?v=y87SUSOfgTY

14 Martin Fowler：“设计持久力假设”，2007 年 7 月 20 日，https://martinfowler.com/bliki/DesignStaminaHypothesis.html。

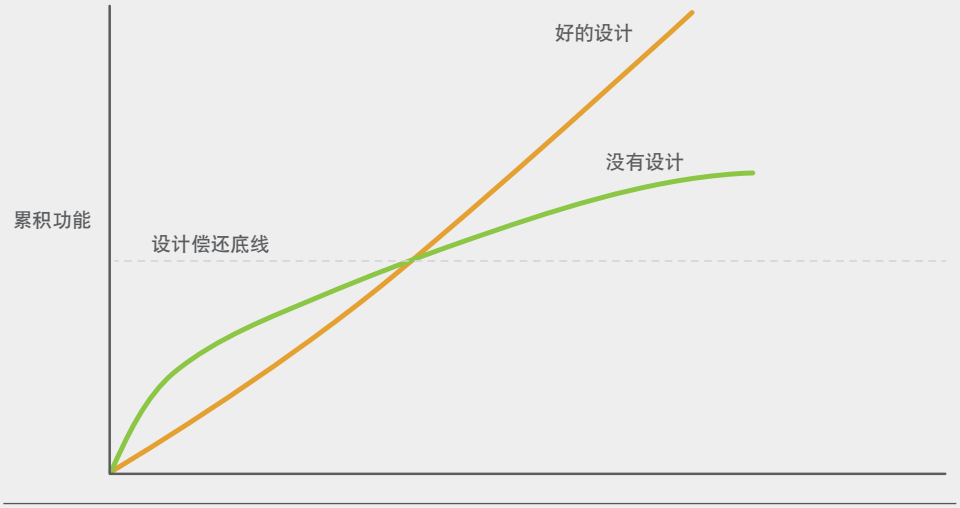

图 7. 设计持久力理论的概念图

在开始规划架构之前，您需要对战略重点和目标有非常清楚的了解。Rob Zuber 曾在 Information Week（信息周刊）上撰文称：[15]

“如果您对自己的业务、产品或定位没有清楚的认识，那么在不知道如何使用它们的情况下将它们过早地分解成若干项服务不会带来好的结果……现在是到了思考架构的时候了，但采用渐进的方式同样也很重要，它可以让你测试和验证自己的想法，而不是幻想着一劳永逸。那样做注定会失败。一个为期 9 个月的项目，其最终的结局却是告诉工程副总裁：尽管你学到了很多东西，但团队最终还是决定不予交付。这绝不是你想看到的。你希望通过详尽的规划和周密的流程为客户和企业带来价值和影响力。”

在最近 IDC 发表的观点中，Stephen Elliot 写道：定义架构之前，您必须确定客户或最终用户是谁，客户看重的是什么，期望的结果是什么，以及如何衡量成功。[16]

换句话说，不要一开始就想着如何完成某个任务或要使用哪种架构。要从战略目标开始，然后设计支持该目标的架构。在这种情况下，应用便被推到了首要位置。

微服务和单体式应用定义

根据服务间的关系，现在主要有三种应用架构：单体式（紧密耦合）、微服务（非耦合）以及（不太受欢迎的）面向服务的架构（松散耦合）。

15 Rob Zuber：“迁移到微服务：两个单体式应用的故事”，InformationWeek，2017 年 5 月 25 日，https://www.informationweek.com/devops/transitioning-to-microservices-the-story-of-two-monoliths-/a/d-id/1328972。

16 Stephen Elliot：“借助现代应用架构实现 DevOps”，IDC 视角，2016 年 12 月。

主动规划

- 谁是您的客户或用户？
- 他们想要做什么？
- 您使用什么基础架构？
- 该基础架构生命周期有多长？
- 单个工作流有哪些服务或功能需求？
- 该工作流的生命周期有多长？
- 您的部署路径是什么？需要多久部署一次？
- 这会影响哪些业务功能？

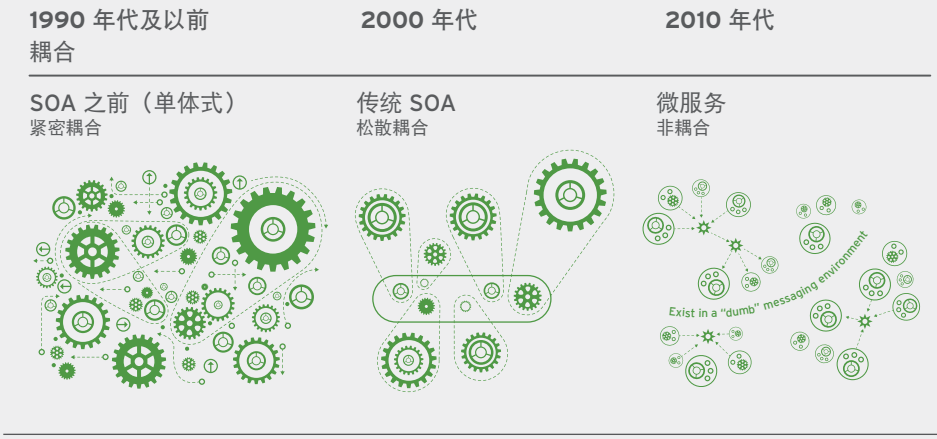

图 8. 单体式架构、面向服务的架构及微服务架构中的耦合

简单来说，单体式应用就是在单个应用中包含所有功能的应用堆栈。无论是服务之间的交互还是开发与交付方式，都采用紧密耦合的形式。更新或扩展单体式应用的某个方面也会对整个应用及其基础架构产生影响。

单体式应用存在的潜在问题是动态扩展和故障转移。这些问题通常可通过简单的扩展性设计来加以解决，例如横向扩展（在集群中复制该功能）或纵向扩展（镜像实例和扩展硬件）。开发和运维团队也很少考虑可扩展性问题。如果需要一个完整的 50 人团队每 6 到 9 个月发布一个单体式应用，那么通过让五个独立的团队提供五个较小的应用并每隔几周发布一次更新，即可提高可扩展性。

考虑到最初的简单性以及服务与依赖项之间的清晰关系，单体式架构可能称得上是最古老的应用架构。该架构也更能反映有限的、基于商品的 IT 基础架构以及更严格的开发和发布流程。

由于单体式是一种较为陈旧的架构形式，因此它们常常与传统应用相关联。相比而言，更为现代化的架构则尝试按功能或业务能力来拆分服务，以带来更好的灵活性。这在面向客户的界面（如 API、移动应用或 Web 应用）中尤为常见。这些界面通常较小，需要更加频繁的更新才能满足客户的期望。

分布式架构的最新定义之一是微服务。与其他模块化设计，像面向服务的架构（SOA）有一些相似之处，但微服务将服务之间的松散耦合转变为服务独立性。通常，单个服务的定义较为明确，可以轻松地在较大的架构中添加、升级或删除服务。这对于动态可扩展性和容错能力都有一定的好处：可以在不占用大量基础架构的情况下按需扩展单个服务，或者可以在不影响其他服务的情况下进行故障转移。正如 Ben Cotton 所说：“基于微服务的设计是您所学到的所有优秀应用设计的终极体现。”[17]

17 Ben Cotton：“从单体式应用到微服务。”2017 年 1 月 3 日，https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/。

微服务架构的流动性意味着它与诸如容器和云之类的动态技术紧密相关，这使得单个实例可以轻松投入使用和销毁，甚至可通过编程方式实现。

分布式计算的即时性对于企业和团队而言都有直接的好处，他们可以随时了解自己的工作成果，包括：

- 改进的容错能力及最短的服务中断。
- JSON/REST 和 HTTP/OAuth 等简单协议，用于简化集成。
- 多语言服务，旨在为开发人员提供相应的灵活性。
- 加快应用和功能的上市时间。
- 简化数据检索和服务间共享，无须再使用较为繁重的消息总线或转换。[18]

Coding the Architecture（架构编码）一书中指出，当应用本身的运行时为单体式、静态时，单体式架构与微服务架构便形成了鲜明的对比。[19] 许多应用都具有大量的软件包或模块，它们在构建或部署方式上可能彼此独立，但应用本身是作为单个实体进行交互的。

潜在的问题是：哪种架构模式最为合适。下意识的反应是认为新的总是最好的，但重要的是应退后一步，评估什么最适合您想要的业务成果。Etsy 和 Netflix 的工程师就微服务持续部署和开发人员自主权的必要性在 Twitter 上展开了一场友好的辩论。Etsy 的工程师指出，他们拥有小规模、基于功能的敏捷开发团队，可在运行单体式应用的情况下仍实现超快部署（每天约 60 次）。他们找到了一种适合自己和企业文化的系统。

架构和技术会不断变化与发展。“昨天还是最佳实践，到了明天就可能变成阻碍，”Laycock 在其 DevNation 会议发言中说道。“微服务是一种选择。它们代表了一种答案，而不是 解决方案；它们只是潜在的解决方案。”[20]

对于企业而言，问题不在于我们能否用微服务来取代单体式架构，而在于我们的战略目标是什么，为了实现这一目标我们需要做些什么。了解沟通结构和文化、业务功能以及所需的工作流程之后，就会影响服务的耦合方式、服务的生命周期，以及最终的应用架构。

18 *Natalie Lambert：“微服务：分解软件单体式架构”，NetworkWorld，2017 年 11 月 22 日，https://www.networkworld.com/article/3143971/application-development/micro-services-breaking-down-software-monoliths.html。*

19 *Robert Annett：“什么是单体式架构？”，Coding the Architecture（架构编码），2014 年 11 月 19 日，http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html。*

20 *Rachel Laycock：“持续交付”，下午的会议。红帽峰会 - DevNation 2016，2016 年 7 月 1 日，美国加利福尼亚州旧金山。https://www.youtube.com/watch?v=y87SUSOfgTY*

表 1. 单体式和微服务架构的比较

	单体式架构	微服务
开发	• 初始开发速度更快 • 难以更改或添加功能	• 早期设计至关重要 • 添加或更改服务较为容易
应用工作流	• 易于将应用纳入工作流 • 可在单个位置实施功能（如身份验证或监控）	• 在工作流中分配服务变得更加复杂 • 服务间的依赖关系或要求可能不明确
培训和维护	• 架构简单 • 对环境和语言有着严格的开发要求	• 架构灵活，设计复杂度得到提高 • 采用标准 API 或消息连接的多语言服务
灵活扩展	• 难以扩展；依赖于硬件基础架构 • 为应对一个服务的高峰而需扩展整个应用	• 易于扩展各个服务，而不会影响整体架构 • 使用软件定义的基础架构（容器、云）实现动态响应
更新、故障转移、停机	• 所有服务均紧密耦合 • 服务必须一起更新；版本耦合 • 如果单个服务出现故障，则存在系统故障的风险	• 服务非耦合 • 可以独立添加或更新服务 • 故障风险仅限于一小部分服务
自动化	• 很大程度上无须自动化	• 需要自动化和编排

有关分布式计算的错误见解

一位工程师打趣说，微服务将每次中断都变成了谋杀疑案。[21] 这是对分布式计算所存在的最大问题的一种形象总结：分散的复杂性。

增加了成本和交易费用

对于真正的分布式计算，所需的基础架构更改可能需要大量的费用。此外，还要涉及重新培训、学习新技能、调整团队结构以及迁移系统等间接成本。当然，这部分费用将来会改善一些指标（如上市时间和停机时间的缩短（如果有效实施了新架构）），但这些好处并非立竿见影。

增加了复杂性

不像单体式架构那样只有单个（灾难性）故障点，微服务架构可能有数百个不同的潜在故障点。与单体式架构类似，这些故障往往很难跟踪，因为根本原因可能并不那么明显。不过，微服务会增加额外的复杂性，因为服务之间的依赖性更不明显。

甚至更快的发布周期和更敏捷的团队结构也会增加复杂性。有效的沟通对于微服务而言至关重要。每个软件架构都要平衡内在的复杂性。这种复杂性可能隐藏在应用本身中（单体式架构），也可能融于团队沟通结构内（微服务）。

系统化思维与设计

要设计有效的微服务架构，需要缜密的 系统化思维。您必须了解服务、业务功能与用户体验之间的相互影响。这种系统化思维还必须扩展到企业文化中的沟通结构和流程层面。Gartner 指出，在不了解整个系统的情况下，微服务架构的性能与刻板的单体式架构非常像：“如果不采用统筹

21 @HonestStatusPage。Twitter，2015 年 10 月 7 日，https://twitter.com/honest_update/status/651897353889259520?lang=en。

考虑软件架构、开发基础架构和开发流程的整体化方法，则无法提供最佳结果，会继续遭受单体式软件系统的诸多弊端所带来的切肤之痛。”[22]

资源限制

鉴于在扩展方面存在的问题，单体式架构的资源限制是显而易见的。系统要么有足够的硬件容量来处理繁重服务上的峰值负载，从而导致不必要的容量冗余，要么就会面对没有足够的容量来处理高峰期的风险。

借助微服务，架构将变得灵活自如，并且由于各个服务的规模都非常小，因此很容易在轻量级的临时资源（例如容器或云实例）上扩展新服务。

因为给定服务的单个资源需求相对较少，所以有时可以忽略整个架构的独特资源需求，或将其压缩到最小。当然，这是基于以下假设：

- 网络始终可靠。
- 无延迟。
- 无限带宽。
- 网络保持安全。
- 基础架构拓扑不会改变。
- 有一个管理员。
- 传输成本为零。
- 网络同构。

还有另一种资源限制，不过它只影响小部分计算用例。超高性能的应用可能对微服务架构有着过高的要求，例如气候模拟或 DNA 定位。

必须要重视有关微服务的错误见解，原因很简单：没有任何系统是完美的。这么做的目的不是要找到解决问题的完美技术（或架构）。相反，专注于文化和沟通并不断完善流程将有助于创建一个成熟、有效的组织体系。在这个基础上，企业才有能力设计满足特定目标的有效架构。

重新思考有意义的应用

很多有关数字化转型的讨论都是集中在基础架构和文化上。鉴于它们属于基本要素，这样做是有道理的。但是，基础架构和文化毕竟是“方法”，它对应的“目的”是创建一个对用户有用、与企业相关的应用。

有意义的应用具有以下特征：

- 可以响应用户
- 反映核心业务功能或商业目的
- 对环境的动态变化具有适应性或应变性
- 跨环境互联互通
- 轻便、灵活，可进行功能的添加和维护

当某个应用满足以上这些特征时，它就是一个有意义的应用。

22 Kirk Knoernschild：“重构单体式软件，以最大程度提高架构和交付的敏捷性”，Gartner 关键洞察，2017 年 5 月 18 日

单体式和微服务架构都可以反映这些特征。架构是一种设计选择。即使在单体式架构中，要想创建一个现代、灵活的应用，适时调整对不同要素的看法也同样至关重要。Etsy 可以实现持续交付、每天数十次更新、高用户负载，甚至支持移动应用，原因就是它采用了庞大的单体式架构，这是一种有明确意图且精心构建的应用架构。

这种意图必须体现在单体式或微服务架构的几个关键方面，其中包括：

- 集成度。
- 数据管理和一致性。
- 消息传递和服务沟通。
- 一致的流程或工作流模式。

总的来说，传统的单体式环境倾向于将这些问题作为需要解决的问题或作为应用的不透明方面来处理。例如，集成有时被视为合并不同应用或数据源的替代方法。这就如同用绷带将环境的不同部分拼凑在一起。甚至像流程之类的东西也被视为可以通过人工干预和工作流程自动化来提高生产率的一种方式，但这不能总是作为核心设计决策。

在现代应用中，架构的某些方面（例如集成和流程）并不是处于次要地位，而是对于应用的性能至关重要。这一点在微服务架构非常明显（对于庞大的单体式架构而言也是如此）。

集成和消息传递

对于微服务而言，矛盾在于如何实现这些服务的耦合，从而在保持这些服务的自主性的同时，仍允许进行自由通信。这是一个集成和消息传递方面的问题。集成定义了这些单独的服务之间如何相互通信。这是一项重要的设计选择。从基础架构的角度看，微服务有时会被拆分为“容器 + API”，以满足耦合这些服务的核心需求，其中容器是服务，而 API 则起到纽带作用。（类似的方法是“容器 + 消息传递”，也就是任何一种提供服务间数据传输方法的技术。）

这与面向服务的架构有所不同，主要体现在两点：一个是消息传递和集成并非集中在总线内，另一个是数据不会在服务之间转换。

流程

现代应用都可以响应其用户。这在以消费者为中心、简单即时的移动应用中最为明显。当客户发起一笔交易时（例如检查银行余额或浏览机票），必须立即启动工作流程。该工作流程必须灵活适应用户的下一个决定，同时还要符合企业的协议。传统的业务流程管理（BPM）主要是一种自动执行任务以提高效率的途径，但在现代应用架构中却发生反转，BPM 成为了交付功能并改善用户体验的主要要素。

数据

某些时候，所有数据都是有状态的。数据必须进行存储并受到架构内服务的访问（无论是分布式还是单体式），因此必须了解数据在服务间的移动方式以及所生成数据的类型。[23] 开发人员 Christian Posta 表示，微服务中最困难的部分是数据管理，需要定义服务间的自然边界和事务限制。[24] 这里的矛盾在于协调数据一致性、可访问性和自主性之间的关系。定义自然域，然后通知数据模型和存储结构。

---

23 Kyle Brown：“重构为微服务，第 2 部分：移动数据时要考虑的因素”，IBM developerWorks，2016 年 5 月 4 日，https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-2/index.html。

24 Christian Posta：“微服务中最困难的部分：您的数据”，2016 年 7 月 14 日，http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/。

“有人试图复制 Netflix，但他们只能复制表面上的东西。他们复制的结果，而非过程。” [27]

– ADRIAN COCKCROFT，
前 NETFLIX 首席云架构师

快速可靠地驱动

数字化转型的主要目标之一是加快应用发布速度。但是，速度只是效率的提高。要想实现转型，速度还需要有一定的目的性——实现快速创新，交付新功能，以及检验新创意。

Ron Kohavi 是微软人工智能实验团队的杰出工程师兼总经理，他在 2013 年明尼苏达大学计算机科学与工程技术系开放日的致辞中指出，只有不到三分之一的创意达到了旨在改善的指标。[25]

评估一个好创意的方法是进行全面、有效的测试，其中不仅包括代码质量，还包括用户体验和偏好。实践出真知。正如 Kohavi 所说，“数据胜于直觉”。

这正是持续交付和高级部署技术的目的。CI/CD 是一个快速部署平台；部署技术则是进行实验和完善的工具。

这两个阶段背后则是文化的改变，即鼓励创新并支持失败和冒险。创新不是目的或目标，而是一个实验过程。勇于在创新之路上承受失败风险需要有一种谦逊文化。

自助服务、自动化和 CI/CD

在数字化转型中，最初的重构举措之一就是转向采用小型动态团队并倡导相互沟通的 DevOps 文化。接下来是技术方面，旨在提供一个支持快速开发周期的基础架构。

这里有两个密切相关的技术阶段：

- 弹性、自助服务式基础结构 — 也就是可以根据确切的规范近乎实时地请求和接收实例。
- 自动化或编排 — 也就是可以在整个环境中自动创建和管理多个实例。

这些技术具有互补性：没有弹性环境，也就无法实现自动化；没有工具来确保一致性和可重复性，也就难以管理数百个实例。

在转型的这一阶段，改进技术可以切实提高生产率。有一位红帽客户引入了自助服务目录，其开发人员可以快速请求虚拟系统，从而将系统请求时间从五天缩短到了大约 15 分钟，并将流程从手动更改为自动。[26] 这种变化有效地释放了运维团队的资源并提高了开发人员的效率（和士气）。

尽管技术是互补的，但并非都是规定性的。弹性基础架构既可以是云（公共云或私有云）、虚拟机，也可以是容器。自动化既可以是基础架构系统中的组件（例如用于 OpenStack® 编排的 Heat 项目），也可以是外部工具（例如带有基于 Docker 的容器的红帽 CloudForms 或 Kubernetes）。企业的技能和现有的基础架构各异，技术之路也各有不同。

容器（例如 Docker 或红帽 OpenShift）之所以与 CI/CD 如此密切相关，原因之一就是它们都可以提供严格且可重复的系统环境，这也意味着在完全不同的企业环境之间迁移时问题会更少。（当代码从开发进入生产阶段时，还在“对笔记本电脑也同样适用”方面争论不休实在是太可怕了。）虚拟机或云实例可以在不同的底层操作系统和程序包版本之间有所不同；容器必须具有相同的操作系统设置，而所选的容器映像在不同的部署之间也是相同的。

这种一致性为 CI/CD 的第一部分（持续集成）奠定了良好的基础。借助持续集成，可以在每次检入时不断编译和构建开发中的变更，因而更容易暴露问题。这项工作通常与自动化测试套件结合在一起，用于验证稳定性或功能性。检入、构建和测试所构成的连续过程可确保代码的更高质量。

---

25 “在线控制实验：简介、洞察、扩展和统计数据”，明尼苏达大学 SOBACO，2013 年 10 月 18 日，https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics。

26 “红帽虚拟化有效提升业务效率和成本效益”，Forrester 总体经济影响研究，2017 年 1 月 26 日，www.redhat.com/zh/resources/virtualization-tei-forrester-analyst-paper。

27 https://twitter.com/kelseyhightower/status/641886057391345664

“对 Docker 最恰当的描述是‘有观点的容器’，因为它的工具集鼓励并建立在不可变基础架构的原则上，依照该原则，开发人员将固定应用配置，并以最少的附加配置变更将其部署到生产环境。” [28]

- IDC 调查报告

一旦持续集成投入运行，企业即可实现持续部署，从而更快地将变更转化为生产。这种速度的提升既有利于开发人员，也有利于运维团队。开发人员和业务主管当然乐于看到新产品更快地进入市场。运维团队则可以在更短的时间内推出修补程序，甚至是重要的通用漏洞披露（CVE），从而提高系统的安全性和性能。

取决于您的发展速度和业务需求，持续的含义可能略有不同。对于庞大的单体式架构（采用完善的流程和技术，以及更传统的应用架构），可以每周发布一次，且进行一次性整体更新，此时对敏捷冲刺流程的要求构成了唯一的约束条件。[29] 对于微服务，由于任何服务都可以进行更新，而且冲刺周期可以重叠，因此可以每天对整个架构进行更新。

高级部署和创新

实现 CI/CD 的阶段涉及到了应用交付的速度和质量。但是，正如 Kohavi 所说，大多数想法并没有实现既定的目标。为了举例说明这一点，他在主题演讲中提供了一系列 Bing 搜索引擎的设计图，并要求大家凭直觉猜测用户实际上更喜欢哪个版本。他依次提出了四个问题，但在数百名与会者中，只有一个人坚持己见。[30]

他的观点是“数据胜过直觉”。Bing 的设计是通过大量用户测试最终确定的，他们发现只有大约三分之一的想法改善了用户体验（同样有大约三分之一的想法则有损于用户体验）。

用于实验的应用基础架构

2006 年（在“微服务”一词进入开发理念约十年前），开发人员 Neal Ford 在他的 Meme Agora 博客上定义了一个称为多语言编程的术语。[31] 由此，确立了一种编程环境的转变。

在企业应用的初期，应用采用的是真正的单一语言形式。随着应用开始转向服务器-客户端或基于 Web 的架构形式，出现了一些专门针对特定操作的语言（例如用于 Web 用户界面的 JavaScript 或用于数据库的 SQL），但核心应用仍然是用单一核心语言编写的。

Ford 的多语言编程思路是不再采用一种核心编程语言，而是在整个应用架构中用完全不同的语言来编写各个服务或功能，从而最好地满足特定服务的需求。

多语言编程代表了敏捷开发环境的核心需求之一：开发人员必须能够尝试新的、应用核心设计之外的内容。对于有效的单体式应用和微服务架构来说，都是如此。

创建实验用平台时，必须考虑开发环境的灵活性和有关的选项。实验用环境必须支持：

- 多种语言。
- 多个运行时。
- 灵活的部署环境，例如云环境、物理环境或混合环境。
- 灵活或可变的应用架构；这样，应用即可在整个生命周期中适应环境的变化。
- 开放标准，或实现迭代标准化的能力。

---

28 IDC《行业发展与模式》系列中的“微服务的出现，成为一种构建新软件系统的新架构方法”，作者：Al Hilwa，2015 年 6 月。

29 Raffaele Spazzoli：“快速发展的单体式架构：我们如何加快交付速度 - 从每三个月到每周”，Red Hat Developer（红帽开发人员），2016 年 10 月 27 日，https://developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/。

30 “在线控制实验：简介、洞察、扩展和统计数据”，明尼苏达大学 SOBACO，2013 年 10 月 18 日，https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics。

31 Neal Ford：“多语言编程”，Meme Agora，2006 年 12 月 5 日，http://memeagora.blogspot.com/2006/12/polyglot-programming.html。

“数据胜过直觉”。

RON KOHAVI，
微软人工智能实验团队总经理
2013 年明尼苏达大学计算机科学
与工程技术系开放日 [23]

容器就是这种支持的一个范例，尽管通过其他平台也可以实现相同的结果。一个容器本质上是有关其中库、语言和版本的规范。但是，一个容器目录可以有成百上千个不同的镜像，支持不同的语言和运行时。这种平台允许开发人员根据需要找到执行服务的确切运行时和语言，并允许运维团队有效地部署这些服务。

创新的部署模式

先进的部署技术能在结构和透明度方面实现创新。成熟的部署方法能够构建一个真正实现实验、反馈和分析的环境。更好的实验有助于实现更好的创新。

这些是常见的部署模式。任何一种方法或全部方法都可能适用，具体取决于您的应用和用户环境的性质。

蓝绿环境

蓝绿环境是缓解因推广更改而带来的风险的一种方法。新的构建版本会穿过 CI/CD 管道中的所有环境。对于生产而言，有两个相同的环境（蓝和绿），但只有一个处于活动状态。所做的更改会被推广至生产中的空闲环境。然后，待环境完成验证后，将会切换路由并将流量移至已更新的环境。

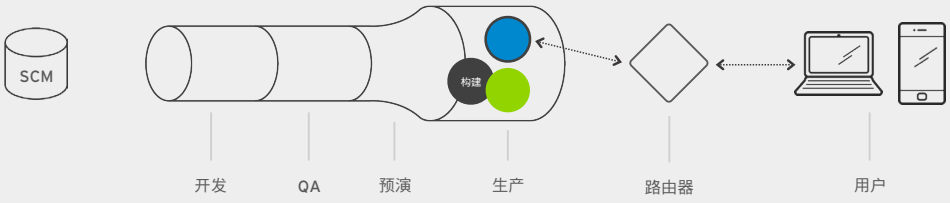

图 9. 蓝绿部署管道

金丝雀发布（Canary releases）

金丝雀发布与蓝绿部署类似，不同之处在于其初始发布只针对环境中的部分用户（对有害气体敏感度超过人体的金丝雀，被带进矿井以预示危险）。随着从用户那里不断收集反馈，可以逐渐增加人数，直到最终切换至所有用户为止。

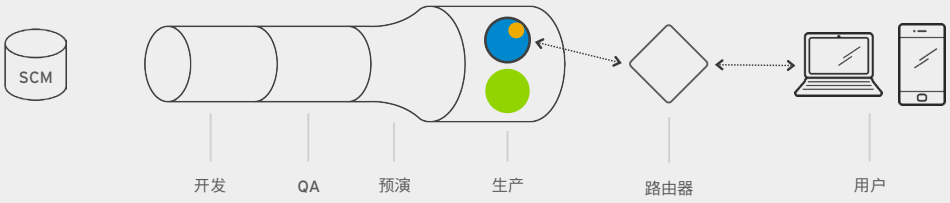

图 10. 金丝雀部署管道

这可以用作测试技术的一部分，以针对当前生产环境中的一小部分用户，借助实际的流量和使用模式来评估应用的不同功能或设计。

A/B 测试

A/B 测试将向用户展示两种不同的设计，然后根据所需的指标来评估哪种设计效果更好。既可以直接让用户对自己的体验进行评分或提供反馈，也可以用更加不易察觉的方式完成。例如，将 A/B 测试与金丝雀发布结合在一起，可以对比两种潜在的设计甚至隐藏的功能，然后以当前环境为基准评估用户对不同设计的反应。

举例来说，在图 11 中，移动应用对用户而言看起来没有什么不同，但 A/B 环境使用不同的算法来测试产品推荐。

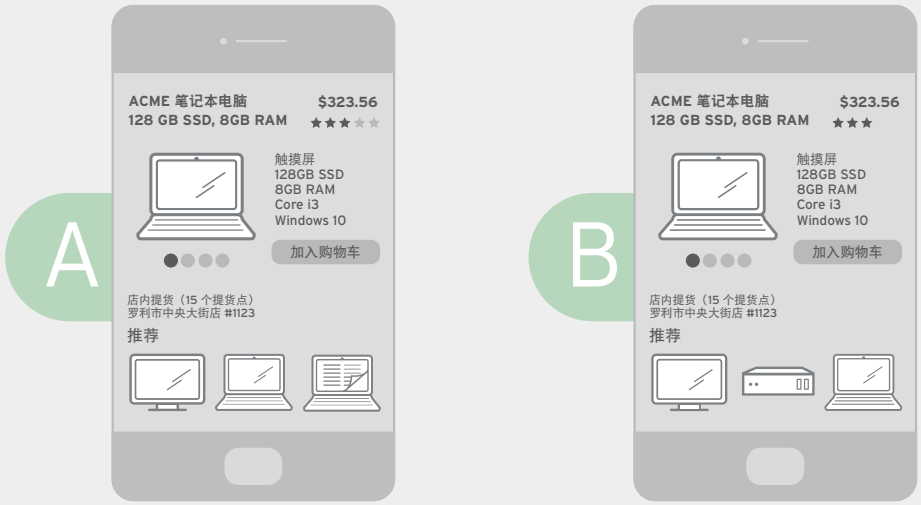

图 11. A/B 界面测试

如同金丝雀发布一样，一旦给出的设计取得成功，就可以将它发布到更大的环境中。

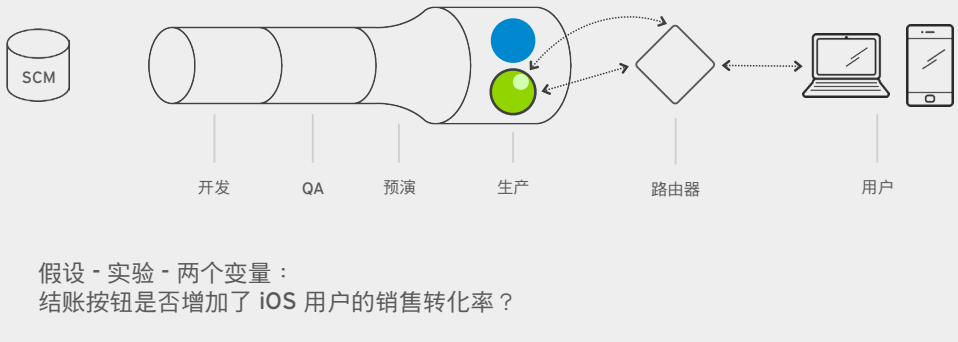

图 12. A/B 测试发布管道

如果一切进展顺利，还可以将生产环境转变为实验环境，这样团队在设计中就会更具创新性和关联性。

这种测试正是“数据胜过直觉”的核心所在。

如何教大象跳舞

选择现有阶段

在更传统的瀑布式环境与完全分布式的微服务之间，有很多阶段。Laycock 认为所有软件架构都是“耦合与内聚之间矛盾”的产物。[32] 在您开始规划数字化转型策略时，这种矛盾就体现在当前的基础架构和文化中。

确定您的企业实际可以走向哪里。当然，这并不意味着“轻松”，因为数字化转型的目标是让文化、流程、架构和技术发生显著改变。这就意味着您要了解这种变更所要实现的目标，然后清楚地评估实现该目标所需的条件。请思考几个问题：

- 您当前有哪些团队或小组？
- 这些小组之间采用什么沟通方式？
- 规划周期内目前涉及哪些人？
- 从功能角度看，当前应用架构与目标应用架构之间有多大差距？
- 您的企业对风险或失败的容忍度有多高？
- 对自己材料和信息流的了解程度如何？（这是企业价值的体现。）
- 您需要多久发布一次更新，以满足客户或运营需求？
- 业务目标或开发需求需要哪些新功能？

制定操作规则

文化的改变是企业针对数字化转型所做的所有流程、技术或架构改变的基础。

尽管道理浅显易懂，但制定一系列得到管理层和不同团队支持的核心规则确实有助于强化数字化转型计划并统一队伍。

这些规则可能比较简单，但保持清晰明了的态度和行为却大有用处，而这些对于文化的改变而言最为重要，尤其是在其中一些态度或行为（例如鼓励冒险和实验）与当前文化背道而驰的情况下。例如：

- 失败不可避免。
- 要敢于实验。
- 企业（人员）优先。
- 追求持续改进。
- 终身学习。
- 始终负责。
- 保持透明。

32 Rachel Laycock：“持续交付”，午后会议发言。红帽峰会 - DevNation 2016，2016 年 7 月 1 日，美国加利福尼亚州旧金山。https://www.youtube.com/watch?v=y87SUSOfgTY

“强大的应用架构、成熟的 DevOps、敏捷的流程以及专注的数据管理团队，共同打造了有效的微服务环境。该环境可将开发准备时间缩短75%。”

– GARTNER[33]

图 13. 更加顺畅地完成精心编排的发布

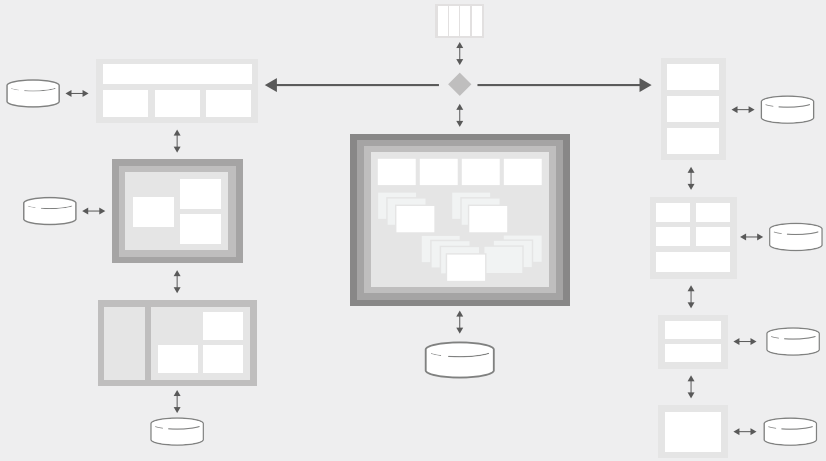

图 14. 拆分各个服务，以“扼杀”单体式应用

告别庞然大物

有些时候，数字化转型会影响您当前的应用和架构。如果您现在采用的是单体式应用，则有两种相关的方法可以开始解决该技术债务：

- 在需要更新或替换时，中断现有服务（strangling）。
- 在另外的独立服务中创建新功能（starving）。

33 “微服务的创新见解”，Anne Thomas 与 Aashish Gupta，2017 年 1 月 27 日

这些都不需要全面采用微服务。Key Bank（红帽的一位客户）需要将发布频率从每季度一次提高到每周一次，并且要在保持单体式应用的情况下做到了这一点。[34] 核心应用逻辑可以保留单体式架构，但有些逻辑域可能存在服务分离的情况，最常见的是用户界面。例如，为 API 系统、移动前端及其他用户界面等服务创建单独的层，可以让面向客户或面向外部的服务比核心应用具有更快的迭代速度或更加独立的生命周期，从而以更低的业务风险实现更多的创新。

Gartner 建议将这种渐进式方法应用于分布式架构，因为它“迭代的同时专注于价值领域”。[35]

无论企业数字化演进的最后阶段是什么，所选择的方法都应涵盖三个基本方面：

- 架构设计的敏捷性。
- 实验性。
- 自动化。

实现未来敏捷性的设计架构

无论您的侧重点是简化流程以改进单体式架构的性能还是创建微服务，架构基础都必须具有敏捷性。通常，敏捷就意味着混合。Gartner 建议新项目以单体式架构开始，待成熟后再推出微服务。[36] Gartner 指出：“您对此的最初反应可能是：'会不会浪费开发工作？’ 简而言之，我们的研究得出了相反的结论：先采用单体式架构的方法可以降低风险、提高初始生产率，并确保将应用分离和分解为正确的微服务集。”[37]

请牢记设计持久力理论，创建一个侧重于清晰性和简易性的开发流程。代码应做到易于理解。功能和用途要明确。随着应用不断成熟，就可以将其发展成更具分布特性的架构形式。拥有良好的开发和部署流程会让这一途径保持敏捷。

预留试验用的时间和预算

必须要留出试验新技术和应用功能的预算空间。例如，IDC 建议仅将 2% 的 IT 预算预留给试验容器技术。[38]

Gartner 指出，技术本身并不是目标。从这个意义上讲，试图进行“在云端部署”或“实施微服务”等改变注定会失败，因为这些改变所要实现的目的不明确。[39]

但是，数字化战略目标经常会得到技术变革的支持。缩短服务上市时间可能需要移至容器（举例而言），而 JavaTM EE 应用可以在容器化平台（例如红帽 JBoss® 企业应用平台（EAP））中运行。

还要预留资源，让开发人员和运维团队可以识别有用的技术并培养相关的技能，以支持最终部署的任何基础架构。

---

34 Raffaele Spazzoli：“快速发展的单体式架构：我们如何加快交付速度 - 从每三个月到每周”，Red Hat Developers（红帽开发人员），2016 年 10 月 27 日，developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/。

35 Gary Olliffe：“如何为敏捷架构设计微服务”，Gartner 关键洞察，2017 年 1 月 30 日。

36 同上。

37 同上。

38 Stephanie Elliot 等，“IDC 技术简报：容器”，IDC 调查报告，2017 年 1 月。

39 Kirk Knoernschild：“重构单体式软件，以最大程度提高架构和交付的敏捷性”，Gartner 关键洞察，2017 年 5 月 18 日。

一切皆自动化

自动化主要有两个明显的优势：通过消除手动步骤来提高效率，以及内在的一致性和可重复性。实现每个开发（开始时）和部署（随着流程不断成熟）步骤的自动化，也就是可就每个步骤的改变以及在利益相关者从开发向运营再向客户转换时为您的团队提供反馈回路。这种方法 提高了整体代码质量。

第一步，为企业的当前状态设置一条基线，以将其纳入自动化策略。接下来的步骤包括：

- 定义相关指标。
- 可视化或图解当前的工作流。
- 确定不同步骤的主要参与者。

结论

长期以来，企业应用往往会演变成死板僵化的庞然大物——不够透明、更新起来繁琐，且迟迟无法纳入新的功能。然而，这些企业应用同时也提供了开展业务和实现创收的核心功能。这就像是房间里的大象。

只要对最终状态有明确的认识，就可以将那头大象训练得敏捷并且具备转变能力。这是数字化转型的进化过程。没有一种所谓理想的结果，每条演变路径都反映了企业本身的独特意图和个性。

评估数字化演进的每个阶段：DevOps、自助服务或弹性环境、自动化、CI/CD 管道、高级部署和微服务。围绕最能满足您业务需求的进化阶段，构建自己的数字化转型策略。

聚焦于打造文化，并在技术变更与相应的流程变更之间取得平衡，以确保您的技术得到团队的全力支持。

随着流程不断成熟，开始评估您的应用和架构。必要时，隔离或开发独立的服务，并创建可以随着业务优先级变化或出现而调整的敏捷架构。

最后，培养创新能力。这意味着对风险和失败要有一定的容忍度（在业务目标和客户需求的范围内）。这就要求架构规范在时间、成本和基础架构方面预留出一定资源。试验是创新的根本，也为数字化转型的成功提供了更好的机会。此外，试验也让人们重拾最初的喜悦，正是这些喜悦吸引着众多开发人员和运维人员热爱技术， 从事创造性工作并见证创新的产生。

销售及技术支持

800 810 2100
400 890 2100

红帽北京办公地址

北京市朝阳区东大桥路 9 号侨福芳草地大厦 A 座 8 层　邮编: 100020
8610 6533 9300

红帽官方微博　红帽官方微信

cn.redhat.com
#8980_0917