ThoughtWorks®

DIGITAL PLATFORM STRATEGY

数字平台战略

构建数字平台 助力企业创新

thoughtworks.com

目录

概述

什么是数字平台战略?

传统企业正在面临IT新技术的挑战——单从“传统企业”这个居高临下的称谓，你就能读出“非传统企业”（也就是IT企业、互联网企业）满满的优越感。每天在各种新媒体平台上看着BAT们又掌握了什么黑科技、又颠覆了哪个行业，“云大物移”已经成了高频出现的热词，传统企业们愈发清晰地感受到IT的重要性与挑战。数字化浪潮躲不过，和BAT拼技术又拼不过，传统企业的出路在哪里?

目光投向大洋彼岸，最传统的传统企业、年收入数千亿美元的沃尔玛在过去几年中的数字化历程颇有可玩味之处。直到2011年，沃尔玛还不是出色的数字化玩家，只能算一个有电商网站的线下零售商而已。正因为如此，当沃尔玛的电商收入在2011年至2014年的三年间增长150%、从年销量49亿美元增长到122亿美元、超过史泰博（Staples）成为亚马逊和苹果之后的美国第三大在线零售商时，这一变化才更令人惊叹。像沃尔玛一样的数字化转型先行者，能给我们带来哪些启示?

提升IT效能

为产品和技术团队赋能，更快更好地为客户交付产品

构建行业生态

构建基于API的IT生态系统，使得新业务线、新产品线和新功能的创新与落地能够充分利用服务化后的企业核心能力和资源

促进业务创新

充分利用核心资产进行高效、快速的创新实验，保持企业竞争力

数字化企业的三个关键字

首先，传统企业们需要清楚一件事："传统"不应该是贬义词，它同时意味着数十年积累的宝贵资产，包括客户关系、数据、品牌形象、供应链、渠道等等。传统企业要在互联网时代的竞争环境中占得一席之地，靠的不是突破最高精尖的技术领域，而是以数字化的形式激活自己多年累积的核心资产，将核心资产转变为可以在互联网上使用的服务，使其焕发新的价值。

对众多成功的数字化企业进行的调研显示，这些企业有着一些引人注目的共性。在"激活核心资产"的过程中，他们对三个关键字的重视特别值得我们关注：IT效能、生态系统、创新实验。

首先，这些成功的数字化企业重视提升IT团队的效能。正如ThoughtWorks在第16期技术雷达中所指出的，技术人员的工作体验正在成为科技企业的差异化竞争优势。这里所说的"体验"不止是给程序员舒适的座椅和人体工学键盘，更重要的是消除IT团队在工作中遇到的阻力和摩擦，尤其是充分利用云计算的弹性能力大量简化和自动化与实现业务功能无关的基础设施性工作，让IT团队将注意力集中在真正与业务相关的工作上。这里涉及的一些技术和实践（例如技术栈管理）乍听起来可能困难重重，但为提升IT团队效能付出的成本终将物有所值。

随后，这些成功的数字化企业把他们的核心商业能力与资产以服务的形式在互联网上提供出来，构建本行业的数字化生态系统，使新的服务和产品能够在这些服务的基础上被创造出来。同样是在技术雷达中，我们看到了"平台的崛起"：几年前只有亚马逊这样的巨头企业能在互联网上提供各种云服务；而现在有更多原本不太有"互联网基因"的企业围绕自己的核心资产建立起了数字化平台，不仅对内、而且对外提供服务。在国内，我们看到华为把软件开发能力变成了云服务、海航建立了自己的云生态。我们相信，更多的企业也能从核心资产的服务化中受益良多。

最后但绝非最不重要的，这些成功的数字化企业养成了创新实验的习惯。在互联网中弄潮的经验让他们承认，自己不能预先掌握所有需求、做好所有设计。因此他们转而打造组织的响应力，致力于缩短精益创业的"构建-度量-学习"周期。他们知道成千上万的用户不会明明白白地说自己想要什么功能，于是他们监控用户行为、用A/B测试等方法进行受控实验，用"假说-实验"代替了"需求-实现"，在不断的反馈中完善自己的产品和服务。

数字平台战略的五大支柱

以提升IT效能、构建行业生态、促进业务创新为目标，有志于迈出数字化步伐的企业应该立即开始制订自己的数字平台战略蓝图。不要被“平台”和“战略”这样的大词欺骗：这个以增强企业响应力为目标的平台战略不应该是在漫长的规划之后建设出一个庞然大物，而应该是迭代的、精益的、价值驱动的。更多的时候，我们谈论的“数字平台”更像是一系列IT技术与实践的落地结合。这些技术与实践有机构成的五个支柱，让数字化的企业能快速交付IT系统、围绕核心资产构建云上生态系统、从线上系统和用户行为中获得洞察、开展受控实验、并为顾客创造全渠道统一的用户体验。在成功的数字化转型案例（例如沃尔玛的案例）中，我们就能看到这五个支柱的投影。

交付基础设施	API和架构治理	数据自服务	创新实验基础设施和监控体系	客户触点技术
快速交付	构建生态系统	获取洞见	有序地创新	一致的体验
弹性基础设施	开发者体验（DX）	数据流水线设计	数据采集	统一客户视角
持续交付流水线	服务边界	实时架构和API	金丝雀发布	内容战略
部署运行时	事件驱动架构	数据湖设计	Toggle架构	个性化
监控	公共网关	数据即产品	路由技术	移动应用持续交付
安全	微服务SOA拓扑	细粒度授权	埋点和可视化	跨渠道引流

第一个支柱是支持云和敏捷的交付基础设施。为了让IT团队快速交付，他们使用的基础设施应该具有弹性，开发、测试、运维等不同角色应该可以随需动态获得完整的应用环境，从而统一环境、标准化研发实践、规范化研发能力。他们开发的应用程序应该用持续交付实践打通开发、构建、验证和部署流程，使软件随时处于可发布状态。他们的交付流程中应该内建对安全的考量，而不是依赖最后的整体安全检查。生产系统所使用的运行时环境应该前向拉通到验证和研发环节，保障运行时环境的一致性。需要对系统的IT运维和业务运营进行全面的监控，聚合起来了解系统整体状况。

第二个支柱是以微服务为核心的API和架构治理。为了鼓励不仅企业内、还包括企业外的开发者在平台上发挥创造力，平台架构和API的设计应该注重开发者体验。在API的背后，应该从业务功能的角度出发划分合理的限界上下文和服务边界，对外提供高内聚低耦合的服务。在服务边界之间，应该考虑使用异步的事件机制实现服务之间的通信，来客观地描述运行时间比较长、甚至本质上不可能立即完成的操作（例如涉及人工操作）。为了方便使用者，应该提供API网关作为所有服务使用者的单一入口点，在API网关背后去处理众多内部IT系统的复杂性。整个API架构应该以微服务的风格呈现，避免典型SOA架构中普遍存在的、过于复杂的ESB编排逻辑。

第三个支柱是允许开发团队数据自服务。为了让业务和研发团队获得关于生产环境、关于线上业务、关于顾客的洞见，他们需要首先定义数据流水线，使数据能够顺畅地流过收集、转换、存储、探索/预测、可视化等阶段，产生业务价值。他们需要用实时的架构和API在短时间内处理大量、非结构化的数据，从中获得洞见，并“实时”影响决策。为了提高应变能力，系统中的数据不做ETL预处理，而是以“生数据”的形式首先存入数据湖，等有了具体的问题要回答时，再去组织和筛选数据，从中找出答案。IT团队会更进一步把数据包装成能供外人使用的产品，让第三方从数据中获得新的洞见与价值。为了支持数据产品的运营，他们需要实现细粒度的身份认证，针对不同的用户身份，授权访问不同范围的数据。

第四个支柱是创新实验基础设施和监控体系。为了让创新真正基于数据（而非拍脑袋）来开展，IT团队需要从多种来源采集关于系统、关于顾客的数据。需要根据业务目标在系统中埋设监控点，并及时把监控结果可视化呈现给业务用户。为了降低实验试错的风险，在把新版本发布给全部用户之前，应该以“金丝雀发布”的形式首先发布给一小部分用户，确保新版本不造成重大损害。系统需要支持功能切换开关（toggle），允许团队在不修改代码的前提下改变系统的行为，再加上用路由技术支持蓝-绿部署和A/B测试，方可高效地开展受控实验。

第五个支柱是支持全渠道的用户触点技术。为了通过多样化的触点技术向顾客提供随时随地、连贯一致的用户体验，整个企业需要建立对其顾客和目标顾客的唯一、连贯、准确、整体的视图，从而更好地了解和服务顾客。他们需要结合顾客的特征和不同数字渠道的特征建立连贯的内容策略，在多种渠道（例如电脑、智能手机、门店等）之间引导顾客的消费旅程，与顾客产生正确时间、正确地点、正确方式的交互。基于从各种渠道获得的顾客本人及其行为的数据分析，他们可以向顾客提供定制化的内容、服务和产品推荐。作为必要的技术保障，所有数字渠道的软件应用（尤其是原生的Android和iOS应用）都应该实践持续交付，这样才能实现全渠道的快速响应。

小结

在数字化的浪潮面前，传统企业不必恐惧于互联网企业的技术优势。只要抓住交付基础设施、API和架构治理、数据自服务、创新实验基础设施和监控体系、用户触点技术这五个支柱，逐步建设自己的数字平台，不断提升IT效能、构建本行业的数字化生态系统、养成创新实验的习惯，传统企业同样可以用数字技术激活自己多年积累的核心资产，在新的竞争环境中找到自己的一席之地。

第一支柱

交付基础设施

传统企业在逐步建设自己的数字平台过程中，需要抓住交付基础设施、API和架构治理、数据自服务、创新实验基础设施和监控体系、用户触点技术这五个支柱。那么，当我们谈“交付基础设施”时，我们究竟在谈什么？怎样的交付基础设施能加速数字化项目的交付？

什么是交付基础设施

云时代的研发环境应该以原生支持云计算的方式来提供、管理和维护。在提供基础的弹性计算能力的IaaS平台之上，交付基础设施负责为交付团队提供便利的、最好是自助式的工作环境，让交付团队专注于交付软件的功能性需求，而不必操心软件功能之外的“脚手架”工作。按照ThoughtWorks数字平台战略的定义，这些脚手架包括：

- 弹性基础设施：即交付团队使用底层云计算平台的方式，既包括各种虚拟机和镜像的管理，也包括生产环境的水平伸缩能力。
- 持续交付流水线：交付团队编写的代码需要通过这条流水线最终变成可以上线运行的软件。
- 部署运行时：软件在开发、测试、试运行、用户验收、培训、生产等各种环境需要部署的环境。
- 监控：为交付团队提供生产环境（及其他环境）的可观测性，方便他们发现和解决问题。
- 安全：把安全内建在软件的研发过程中，尽量避免因为人为失误造成安全隐患。

从前这些交付基础设施脚手架通常是由每个交付团队的技术领导者（Tech Lead）来负责搭建和维护的。并且由于软硬件资源的稀缺和不灵活，团队经常需要微调自己的实践来适应不同的环境。所以，即使在同一家公司，各支团队所使用的交付基础设施也可能大相径庭。交付基础设施不一致、不规范的情况会迫使团队花费额外的精力去操心脚手架工作，并且使最佳实践不易推广普及。走上数字化道路的企业必定有大量的软件项目，尤其是微服务架构风格的引入会使企业拥有数量更多、单体规模更小的软件应用，此时交付基础设施不一致、不规范的情况就会给企业的数字化进程带来更大的阻力。

云计算带来的弹性和灵活性让组织级的交付基础设施标准化、规范化成为可能。一个跨越项目团队的、组织级的交付基础设施团队现在可以在IaaS的基础上封装标准的脚手架，甚至把脚手架本身以PaaS的形式提供给交付团队。通过把整个企业优秀技术领导者的知识与经验内嵌在交付基础设施脚手架中，就降低了对单个交付团队的技术要求，帮助企业缓解优秀技术领导者难以获得的人才挑战。从这个意义上，以PaaS形式提供的交付基础设施本质上是技术领导者作为服务（Tech Lead as a Service）的云计算应用形式，它解决的是优秀技术人才的弹性和灵活性问题，让企业能够以一种创新的方式使用这些人才。

架构师写代码吗?

关于“架构师是否应该写代码”这个问题，业界有各种不同的声音。在敏捷的社区里，意见倾向于认为架构师需要写代码，因为这是他们获得关于技术决策的反馈和建立技术领导力的重要方式。将交付基础设施明确提出来，就给了架构师又一个清晰的编程目标——他们需要用代码的形式描述软件交付中的基础设施和最佳实践。除了培训、开会、代码评审等我们已经知道效率并不太高的方式以外，架构师对交付团队的指导和监管现在可以用实实在在的代码来承载。当交付团队不理解架构师说的某件事应该怎么做，现在他们更有理由要求架构师“show me the code”。

交付基础设施解读

下面我们来看看，在“交付基础设施”这顶帽子下面，架构师/技术领导者们究竟应该关心哪些问题，又有哪些最佳实践应该被纳入他们的视线。

1 弹性基础设施

允许交付随需获得计算能力。在微服务语境下，这种弹性有两层常见的含义：在生产环境下，服务可以随负载动态获得和释放计算资源，从而更高效地使用计算资源，更自动化地应对负载变化；在研发环境下，开发、测试、运维等不同角色可以随需动态获得完整的环境，从而统一环境、标准化研发实践、规范化研发能力，并且给研发提供体验更好的开发环境。

为了实现弹性基础设施，一方面基础设施需要支持弹性，例如使用支持弹性计算的公有/私有云，并且有对生产环境的监控和自动化手段；另一方面应用本身需要有可扩展性，例如服务能分别独立部署、无状态化、容器化、有透明的前端负载均衡机制。有状态服务（比如数据库服务）的弹性伸缩问题是特别需要考虑的重要挑战。

2 持续交付流水线

用持续交付实践打通微服务的开发、构建、验证和部署流程。在数字化、服务化的背景下，众多互相依赖的微服务形成的系统架构，对构建、验证和部署造成更大的压力：各个服务有独立的代码库和构建流程，又需要随时能组合成可用的软件；构建产物需要有统一的存储管理；完整的运行时环境应该能按需获得；配置和部署应该能快速准确地完成。

为了应对这些挑战，交付基础设施中应该包含完整的持续交付概念：流水线、环境管理、构建产物管理等。应该鼓励对服务虚拟化，最好是每个主机运行一个微服务，而不共享使用主机。应该包含配置自动化工具，例如Chef、Puppet等。在服务化的背景下，持续交付流水线需要体现服务间的依赖关系和团队间的协作关系，设计一个运转良好的流水线不是容易的任务。

3 部署运行时

交付基础设施应该包含生产系统所使用的运行时环境，并把生产环境前向拉通到验证和研发环节。为了在研发流程的出口得到服务化友好的交付物，最好是在整个开发过程中一直使用与生产环境近似的环境。例如开发人员应该使用全套环境随时验证，自动化测试和手工测试都基于全套环境开展。在这种情况下，环境的设置、管理、更新不可能由每个开发人员和测试人员自己进行，所以环境的管理更新必定是集中进行的，环境的设置必定是自动化的。

在《技术栈管理：云时代的研发环境》一文中，我们已经介绍过“一个平台、两个PaaS服务、三个运行时环境”的技术栈管理理念。特别需要注意的是，如何将生产数据拉通到验证和研发环节。

4 监控

在微服务架构中，系统由多个小服务组成，且广泛使用异步通信，使问题和故障更难定位。因此交付基础设施需要提供全面可靠的监控机制，帮助交付团队了解系统的整体状况。

监控的实现涉及日志、服务指标跟踪、业务语义综合监控等方式。在云环境下如何划分和管理监控的层级，监控系统如何无侵入的在各个微服务体系中收集故障和信息，如何有效管理监控的反馈环，如何在前后端分离和移动应用情况下收集和监控客户端日志，都是常见的挑战。

5 安全

当数字化、服务化IT系统的数量剧增，安全的设置会变得更加复杂。在微服务架构下，系统的安全性需要有一个整体的考虑。例如单点登录、服务间的身份验证和授权、各种防御措施等安全考量不应该下放到交付团队，而应该被涵盖在交付基础设施中统一提供、统一管理、统一更新。

交付基础设施还应该鼓励安全实践内建（Build Security In），例如团队应该熟悉OWASP安全列表和测试框架、需求分析中应该包含安全需求和恶意用户需求、测试过程中应该包含安全性测试、应该进行自动化安全性测试并纳入持续交付流水线。这些流程与工作方法虽然不能完全以软件代码的形式承载，但它们同样是交付基础设施的重要组成部分。

小结

数字化、服务化的IT大背景会让企业开发和拥有的IT系统数量剧增。当企业IT交付更多地以“两个pizza团队”的形式组织，依赖于每个交付团队的技术领导者来搭建和维护一套完整高效的交付基础设施脚手架，这种期望即使不是完全不现实，也会对企业的人才积累提出非常高的要求。因此，企业应该集中优秀的技术人才（包括架构师们），打造一套标准的交付基础设施，充分考虑生产环境与研发环境的弹性、持续交付、部署运行时的统一、监控、安全等因素，并借助云计算的弹性和灵活性将其提供给交付团队。用便利的脚手架赋能一支能快速交付的团队，这是企业的数字化旅程的第一步。

第二支柱

API和架构治理

企业资源服务化

从1990年代起，企业资源计划（ERP）一直是企业信息化的核心议题。植根于供应链管理，ERP通过对企业内部财务会计、制造、进销存等信息流的整合，提升企业的计划能力与控制能力。然而近年来，在互联网的冲击下，传统企业开始面临全新的挑战。尤其是在互联网的去中介化效应影响下，原本在供应链上下游各安其位的企业突然间都被压缩到了“生产-流通-消费”这个极度精简的价值链中。药品购销两票制就是这个极简价值模型的直观呈现。在这个模型中，掌握技术优势和消费者入口的互联网企业有可能形成一家独大的超级垄断，挤死传统的流通企业，把生产企业变成自己的OEM厂商，这是传统企业对来自互联网竞争者恐惧的根源。

为了对抗互联网企业的竞争，传统企业最好的办法不是硬拼互联网上的技术和流量，而是在自己擅长的领域开战：把自己多年积累的线下资源变成线上服务，构建起本行业的线上生态系统，不仅支撑本企业的线上经营，而且为上下游周边企业提供线上经营的平台，从而把线下优势转化为线上优势，以资源优势对抗技术优势。

为了支撑企业资源的服务化，在设计在线服务的API和架构时需要考虑以下问题：

- 平台架构和API的设计应该注重开发者体验。
- 在API的背后，应该从业务功能的角度出发划分合理的限界上下文和服务边界，对外提供高内聚低耦合的服务。
- 在服务边界之间，应该考虑使用异步的事件机制实现服务之间的通信，来解耦领域模型，客观地描述运行时间比较长、甚至本质上不可能立即完成的操作。
- 为了方便使用，应该提供API网关作为所有服务使用者的单一入口，在API网关背后去处理众多内部IT系统的复杂性。
- 整个API架构应该以微服务的风格呈现，避免典型SOA架构中普遍存在的过于复杂的ESB编排逻辑。

ERP之后是什么?

进入2010年代以来，“后ERP时代”这个说法不断被提出。在谈到ERP的发展方向时，通常都会涉及业务与技术两个角度。例如一种观点认为，ERP需要从以流程为中心转变为以客户为中心，并且需要用好云计算、社交网络、大数据和移动化等新技术。

ThoughtWorks认为，ERP在互联网时代的发展方向将是企业资源服务化（Enterprise Resource Servicification，ERS），通过数字平台的技术能力，把一家企业的资源融入一个行业的互联网生态，为企业铺下明确的数字化道路。

API和架构治理解读

下面我们来近距离看看，在“API和架构治理”这顶帽子下面，有哪些具体的问题需要被考虑到。

交付基础设施	API和架构治理	数据自服务	创新实验基础设施和监控体系	客户触点技术
				
快速交付	构建生态系统	获取洞见	有序地创新	一致的体验
弹性基础设施	开发者体验 (DX)	数据流水线设计	数据采集	统一客户视角
持续交付流水线	服务边界	实时架构和API	金丝雀发布	内容战略
部署运行时	事件驱动架构	数据湖设计	Toggle架构	个性化
监控	公共网关	数据即产品	路由技术	移动应用持续交付
安全	微服务SOA拓扑	细粒度授权	埋点和可视化	跨渠道引流

1 开发者体验

当企业资源以服务的形式对外提供，也就意味着不可能——像传统的IT系统建设那样——强迫别人使用这些服务。尤其是要把这些服务提供给第三方开发者、希望他们开发出形形色色的应用程序，那么服务的API是否易用就会很大程度上影响它能吸引到多少第三方开发者。ThoughtWorks第16期技术雷达还专门把开发者体验作为一个重要的技术主题。

在讨论开发者体验时，可以从开发工具和开发环境的安装、配置、管理、使用、维护等角度来考量。具体而言，开发环境和测试环境是否能弹性地随需获得，开发/测试基础设施和持续交付流水线是否以源代码的形式提供并完全自动化，是否提供对主流开源软件的支持，是否提供可编程的、命令行友好的（而不仅仅是图形化的）工具界面，安全、数据访问权限等企业规章是否严重影响开发者的效率和感受，这些都是影响开发者体验的要素。

2 服务边界

和所有的面向对象设计一样，服务的设计应该是高内聚低耦合的：与一个业务相关的修改只在一个服务内部进行，并且一个服务的修改/部署不需要影响其他服务。和一个代码库内部的对象设计不同，每个服务通常有专属的代码库，并且由专人负责维护（而不是所有人拥有所有代码），因此服务边界的改变会带来更大的变更成本。所以，服务边界的划分需要投入精力认真对待。

从设计原则上来说，服务的边界应该体现业务的边界，而不是单纯从技术角度出发划分服务边界。从业务功能的角度出发划分合理的限界上下文，以领域模型和领域事件的聚合为出发点来划分服务，更可能得出与业务边界一致的服务边界。随后再以业务目标驱动建设全功能一体化团队，就能做到业务、技术、团队三者对齐（康威定律再次起作用）。四色建模、事件风暴等方法都能有效地实现领域驱动设计，从而建立起良好的领域模型及服务边界。

3 事件驱动架构

使用异步的事件机制实现服务之间的通信。对于运行时间比较长、甚至本质上不可能立即完成的操作（例如涉及人工操作），使用异步通信是合理的选择。即便不考虑响应的实时性，事件驱动的架构还表达了领域模型之间的松散耦合关系：跨领域的协作以事件而非方法调用的形式来表达，系统追求最终一致性而非强一致性。这一结构准确地映射了真实世界中多支相关但独立的团队之间的协作关系，避免了过度依赖其他服务的响应速度或可靠性等服务质量指标，使服务真正具有技术上的独立性。

在设计系统时，借助事件风暴方法，可以通过领域事件识别出聚合根，进而划分微服务的限界上下文。当出现跨多个聚合根的事件时，可以很自然地将其实现为异步的领域事件，从而获得与领域设计高度吻合的实现。关于如何设计和实现领域事件，可以参阅ThoughtWorks咨询师滕云的文章。

在实现事件驱动的架构时，当然可以沿用传统的SOA架构中的消息中间件。但由于微服务架构中，业务逻辑都存在于各个服务内部，没有庞大臃肿的ESB（稍后我们还会详谈这个问题），因此消息机制也不需要强大的服务编排（orchestration）能力。RabbitMQ这样标准的消息代理当然很好，也有很多系统（例如Bahmni）采用更简单的做法：领域事件发生时，以ATOM格式发布；关心特定领域事件的其他领域模型则订阅特定的ATOM feed主题。这种基于HTTP的事件传播方式最大的优势就是简单，几乎不需要增加新的软件就可以实现。不过这个方案在处理低延迟的场景时表现不佳。

4 公共网关

微服务提供的API粒度与客户端的需求不同，所以客户端一个请求经常需要多个服务；服务端和客户端之间可能需要通信协议转换；不同的客户端对数据的需求不同，例如浏览器客户端需要的信息可能多于移动客户端；服务的终端信息（主机+端口）可能变化；不同数据片可能由不同的服务终端来提供——以上这些因素都指出：有必要对服务做一层门面封装，提供API网关作为所有服务使用者的单一入口点。

API网关处理请求的方式有两种：一种是直接代理/路由给合适的服务；另一种是由一个请求扇出/分发给多个服务。API网关可能针对不同客户端提供不同的API，可能包含针对客户端的适配代码。横切需求（例如安全）也可能在API网关实现。

当服务数量变多、API网关变大以后，维护一个通用的API网关会增加API网关层的复杂度，导致一个独立的“API团队”出现，协调和沟通的工作量加大。这时可以考虑引入公共网关的一个变体：为特定前端设计的后端（Backend For Frontend，BFF），即为每个前端应用提供一个单独的API网关，使对齐业务的一体化团队能够拉通前后端开发、而不必等待“API团队”完成他们的backlog。

API网关可以实现为一个独立的服务端应用，其代价则是增加一层复杂度（和出错的可能性）。为了降低这一代价，可以考虑用Zuul等工具来实现API网关。

5 微服务SOA拓扑

与传统的SOA架构相比，所谓“微服务”最大的特点可能就在于没有一个重量级的ESB。重量级的ESB有其历史原因。在2000年代业界刚开始采用SOA时，很多企业尽管把业务系统包装成了web服务，但IT团队的组织结构并没有发生改变，仍然是由一组人集中式地掌管整个业务流程——只不过系统集成的方式不再是直接的方法调用，而是服务编排（orchestration）。原本存在于集成代码中的复杂逻辑，现在被转移到了ESB中。而这个“ESB团队”成了IT交付的瓶颈：不论发布事件的服务还是消费事件的服务、或是编排逻辑本身的改变，与事件相关的变更都需要通过ESB团队。这个团队的backlog堆积起来，使得每个服务、每个应用都无法提供快速响应。

微服务架构更重视服务与业务的对齐。贝索斯所说的“两个pizza的团队”不仅负责一个IT系统的交付，而且要负责用这个IT系统来支撑一个业务的成功。为了做到单个服务能够独立开发、独立部署、独立运行，这支团队应该能够在很大程度上掌控自己的进度，而不依赖于一个集中式技术团队的进度。因此微服务应该通过服务注册与发现机制获得自己需要的依赖服务、自己判断是否要直接调用或订阅依赖服务的事件，每个服务包含与其业务对应的复杂度，而不是把整个系统的复杂度集中在ESB和编排逻辑上。整个系统的架构（以及团队的架构）应该呈现为若干个端到端拉通的、与业务对齐的纵切服务，而不是一个横切的大块（ESB）覆盖所有业务。

小结

为了激活企业线下资源、打造行业线上生态，IT需要一套有效的服务API和架构治理方法。首先从领域驱动设计入手，划分出合理的限界上下文和服务边界，然后用异步消息机制来描述领域事件。设计好的服务通过API网关或BFF暴露给前端应用，把依赖关系和集成逻辑约束在与业务对齐的一体化团队内部。在整个服务架构的设计中，需要保持对开发者体验的关注。顺畅地将企业资源服务化，这是企业数字化旅程的第二步。

第三支柱

数据自服务

什么是数据自服务

数据在企业中的处理过程，能清晰地映射出康威定律对IT系统的影响。在各个部门分别建设IT系统、组织内部大量存在信息筒仓（silo）的年代，数据的操作由OLTP应用系统的开发团队同步开发，那时几乎每个政府信息化、企业信息化系统都会有一块“报表需求”。随后众多组织认识到筒仓系统导致信息在组织内不能拉通，不能产生对整体业务流程的洞察，于是开始建设以数据仓库为代表的OLAP系统。

这些系统在支撑更高级、更复杂的数据分析的同时，也对应地在组织中造就了一支专业的“数据团队”。这些人使用非常专业的技术和工具对数据进行提取、转换、装载、建立数据立方、多维钻取、生成报表。这些专业的技术和工具，普通的软件开发人员并没有掌握，因此对数据处理、分析和呈现的变更都必须归集到这个数据团队来完成。结果是，数据团队的backlog里累积了来自各个部门的需求，需求的响应能力下降，IT系统从上线到获得市场洞察的周期变长。

微服务架构鼓励小型的、全功能的团队拥有一个完整的服务（及其对应的业务）。这样的全功能团队不光要开发和运维IT系统，还要能从数据中获得洞察——而且要快，不然就会跟不上市场变化，甚至使一些重要的业务场景无法得到支撑。因此他们不能坐等一支集中式的、缓慢的数据团队来响应他们的需求，他们需要数据自服务能力。

要赋能数据自服务，企业的数字化平台要考虑“两个披萨团队”的下列诉求：

- 需要定义数据流水线，使数据能够顺畅地流过收集、转换、存储、探索/预测、可视化等阶段，产生业务价值。
- 需要用实时的架构和API在短时间内处理大量、非结构化的数据，从中获得洞见，并实时影响决策。
- 为了提高应变能力，系统中的数据不做ETL预处理，而是以“生数据”的形式首先存入数据湖，等有了具体的问题要回答时，再去组织和筛选数据，从中找出答案。
- 更进一步把数据包装成能供外人使用的数据产品，让第三方从数据中获得新的洞见与价值。
- 为了支持数据产品的运营，需要实现细粒度授权，针对不同的用户身份，授权访问不同范围的数据。

数据自服务解读

下面是ThoughtWorks的数字平台战略第三个支柱“数据自服务”中所蕴涵的具体内容。

交付基础设施	API和架构治理	**数据自服务**	创新实验基础设施和监控体系	客户触点技术
快速交付	构建生态系统	**获取洞见**	有序地创新	一致的体验
弹性基础设施	开发者体验（DX）	数据流水线设计	数据采集	统一客户视角
持续交付流水线	服务边界	实时架构和API	金丝雀发布	内容战略
部署运行时	事件驱动架构	数据湖设计	Toggle架构	个性化
监控	公共网关	数据即产品	路由技术	移动应用持续交付
安全	微服务SOA拓扑	细粒度授权	埋点和可视化	跨渠道引流

1 数据流水线设计

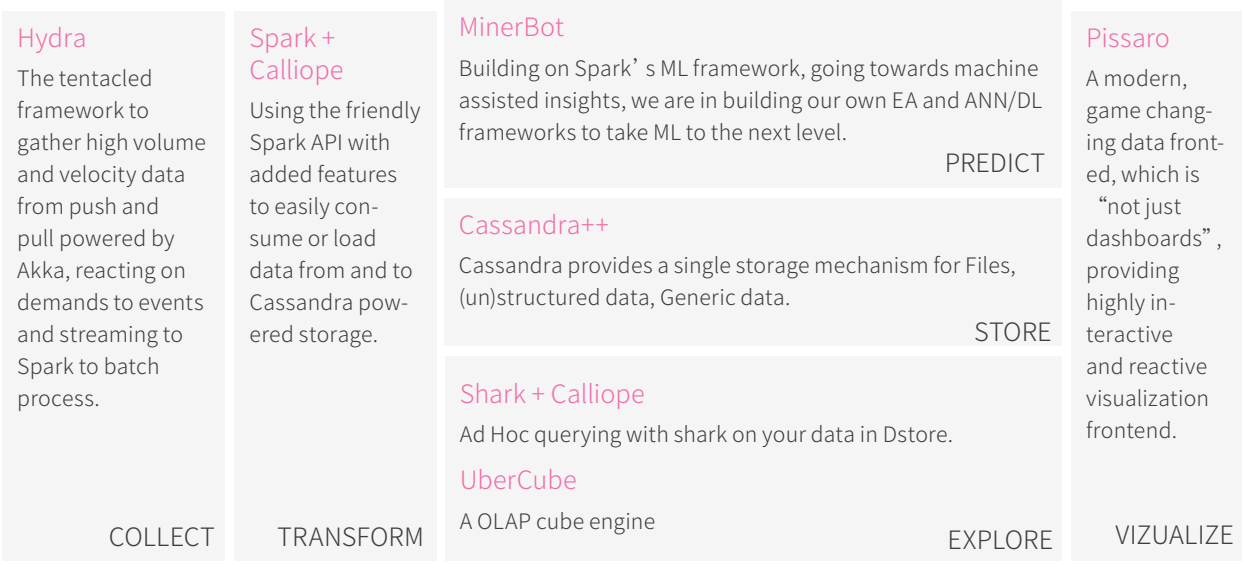
所谓流水线，是指用大数据创造价值的整个数据流。流水线从数据采集开始，随后是数据的清洗或过滤，再然后是将数据结构化到存储仓库中以便访问和查询，这之后就可以通过探索或预测的方式从数据中找到业务问题的答案，并可视化呈现出来。

（图片来自：tuplejump Inc.）

一条运转良好的数据流水线，能有效处理移动/物联网等新技术制造出的极其大量的数据，缩短数据从获取到产生洞见的反馈周期，并以开发者友好的方式完成数据各个环节的处理，赋能一体化团队。

数据流水线的实现有两种可能的方式。一种方式是在各个环节采用各种特定的工具，例如前面介绍的数据流水线，各个环节都可以用开源的工具来实现。当然，选择这种方式也并非没有挑战：组织必须自己编写和维护“胶水代码”，把各种专用工具组合成一个内聚的整体。对组织的技术能力有较高的要求。

除了基于开源软件实现自己的数据流水线，也可以考虑采用云上的数据流水线PaaS服务，例如Databricks、AWS Data Pipeline、Azure Data Factory等。这个方式的优点是对技术能力要求较低，缺点则是造成对特定云平台/PaaS提供商的依赖。



（图片来自：tuplejump Inc.）

2 实时架构和API

实时的数据架构和API支持短时间内处理大量、非结构化的数据从中获得洞见，并“实时”影响决策。正如Mike Barlow所说：“这是关于在正确时间做出更好决定并采取行动的能力，例如在顾客刷卡的时候识别信用卡欺诈，或者当顾客在排队结账的时候给个优惠，或者当用户在阅读某篇文章的时候推送某个广告。”

在Cloudera的一篇文章中介绍了实时数据处理的4个架构模式，整个流水线架构在Flume/Kafka基础上：

- 数据流吸收：低延迟将事件持久化到HDFS、HBase、Solr等存储机制
- 近实时（100毫秒以下）的事件处理：数据到达时立即采取警告、标记、转换、过滤等初步行动
- 近实时的事件分片处理：与前一个模式类似，但是先对数据分片
- 复杂而灵活的聚合或机器学习拓扑，使用Spark

3 数据湖设计

数据湖概念最初提出是在2014年Forbes的一篇文章中。它的概念是：不对数据做提前的“优化”处理，而是直接把生数据存储在容易获得的、便宜的存储环境中；等有了具体的问题要回答时，再去组织和筛选数据，从中找出答案。按照ThoughtWorks技术雷达的定义，数据湖中的数据应该是不可修改（immutable）的。

数据湖试图解决数据仓库几方面的问题：

- 预先的ETL处理终归会损失信息，如果事后才发现需要生数据中的某些信息、但是这些信息又没有通过ETL进入数据仓库，那么信息就无法寻回了。
- ETL的编写相当麻烦。数据仓库的schema发生改变，ETL也要跟着改变；应用程序的schema发生改变，ETL也要跟着改变。因此数据仓库通常由一个单独的团队负责，于是形成一个功能团队，响应速度慢。
- 数据仓库的分析需要专门的技能，大部分应用程序开发者不掌握，再度强化了数据仓库专门团队；而数据仓库团队其实离业务很远，并不能快速准确地响应业务对数据分析的需求。
- 在数据湖概念背后是康威法则的体现：数据能力与业务需求对齐。它要解决的核心问题是专门的数据仓库团队成为响应力瓶颈。当IT能力与业务需求组合形成一体化团队以后，数据的产生方不再假设未来要解决什么问题，因此也不对数据做预处理，只是直接存储生数据；数据的使用方以通用编程语言（例如Java或Python）来操作数据，从而无需依赖专门的、集中式的数据团队。

数据湖实施的第一步是把生数据存储在廉价的存储介质（可能是HDFS，也可能是S3，或者FTP等）。对于每份生数据，应该有一份元数据描述其来源、用途、和哪些数据相关等等。元数据允许整个组织查看和搜索，让每个一体化团队能够自助式寻找自己需要的数据。任何团队都可以在生数据的基础上开发自己的微服务，微服务处理之后的数据可以作为另一份生数据回到数据湖。维护数据湖的团队只做很少的基础设施工作，生数据的输入和使用都由与业务强关联的开发团队来进行。传统数据仓库的多维分析、报表等功能同样可以作为一个服务接入数据湖。

在实施数据湖的时候，有一种常见的反模式：企业有了一个名义上的数据湖（例如一个非常大的HDFS），但是数据只进不出，成了“数据泥沼”（或数据墓地）。在这种情况下，尽管数据湖的存储做得很棒，但是组织并没有很好地消化这些数据（可能是因为数据科学家不具备分析生数据的技术能力，而是更习惯于传统的、基于数据仓库的分析方式），从而不能很好地兑现数据湖的价值。

4 数据即产品

数据产品是指将企业已经拥有或能够采集的数据资产，转变成能帮助用户解决具体问题的产品。Forbes列举了几类值得关注的数据产品：

- 用于benchmark的数据
- 用于推荐系统的数据
- 用于预测的数据

数据产品是数据资产变现的快速途径。因为数据产品有几个优势：开发快，不需要开发出完整的模型，只要做好数据整理就可以对外提供；顾客面宽，一份数据可以产生多种用途；数据可以再度加工。数据产品给企业创造的收益既可以是直接的（用户想要访问数据或分析时收费）也可以是间接的（提升顾客忠诚度、节省成本、或增加渠道转化率）。

在实现数据产品的时候，不仅要把数据打包，更重要的是提供数据之间的关联。数据产品的供应者需要提出洞见、指导用户做决策，而不仅仅是提供数据点。数据产品需要考虑用户的场景和体验，并在使用过程中不断演进。

5 细粒度授权

当数据以产品或服务的形式对外提供，企业可能需要针对不同的用户身份，授权访问不同范围的数据，对应不同的服务水平和不同的安全级别。一些典型的细粒度授权的场景可能包括：企业内部和外部用户能够访问的数据范围不同；供应链上不同环节的合作伙伴能够访问的数据范围不同；付费与免费的用户能够访问的数据范围不同；不同会员级别能够访问的数据范围不同等等。

允许访问的数据范围属于数据产品/服务自身的业务规则。《微服务设计》的第9章建议，“[服务]网关可以提供相当有效的粗粒度的身份验证……不过，比允许（或禁止）的特定资源或端点更细粒度的访问控制，可以留给微服务本身来处理”。

小结

为了加速“构建-度量-学习”的精益创业循环，业务与IT共同组成的一体化团队不能依赖于集中式的数据团队来获得对业务的洞察。他们需要规划适宜自己的数据流水线，在必要时引入实时数据架构和API，用数据湖来支撑自服务的数据操作，从而更快、更准确地从数据中获得洞察，影响业务决策。更进一步，数据本身也可以作为产品对内部用户乃至外部用户提供服务，并通过细粒度授权体现服务的差异化和安全性需求。通过建设“数据自服务”这个支柱，企业将真正能够盘活数据资产，使其在创新的数字化业务中发挥更大的价值，这是企业数字化旅程的第三步。

第四支柱

创新实验基础设施

什么是创新实验基础设施

作为数字化企业的代表，亚马逊是众多怀揣数字化梦想的企业学习的榜样。今天的亚马逊，在零售、广告、消费电子终端、应用商店、云服务等多个领域与各领域的领先企业竞争。更可怕的是，除了丰富的业务线，亚马逊还有Dash Button、Echo、Prime Air、AWS等大量创新。最可怕的是，据AWS的CEO说，除了这些大家知道的、获得了一定成功的创新项目，还有更多创新项目失败了——而亚马逊认为完全OK。亚马逊强大的创新能力，背后体现的是更为强大的快速实验、快速学习、快速调整的能力。缺少这种能力，就算把亚马逊的产品和模式摆在面前照抄，也无法跟上它不断创新的步伐。

为了支撑数字系统的快速实验、快速学习、快速调整，需要在快速交付基础设施与数据自服务的基础上再考虑下列问题：

- 需要从多种来源采集关于系统、关于顾客的数据。
- 需要根据业务目标在系统中埋设监控点，并及时把监控结果可视化呈现给业务用户。
- 为了降低实验试错的风险，在把新版本发布给全部用户之前，应该以金丝雀发布的形式首先发布给一小部分用户，确保新版本不造成重大损害。
- 系统需要支持功能切换开关（toggle），允许团队在不修改代码的前提下改变系统的行为。
- 用路由技术支持蓝-绿部署和A/B测试，方可高效地开展受控实验。

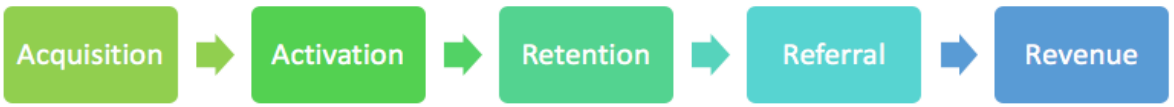
实验基础设施详解

数字平台中的实验基础设施由以下特性共同支撑。

1 数据采集

精益创业的核心逻辑是缩短“Build-Measure-Learn”的周期。为了从实验中学习，需要全面采集实验数据。交付基础设施通常会包含技术性的监控和数据采集（例如基于ELK的日志监控体系），提供性能、资源、系统告警等角度的数据。

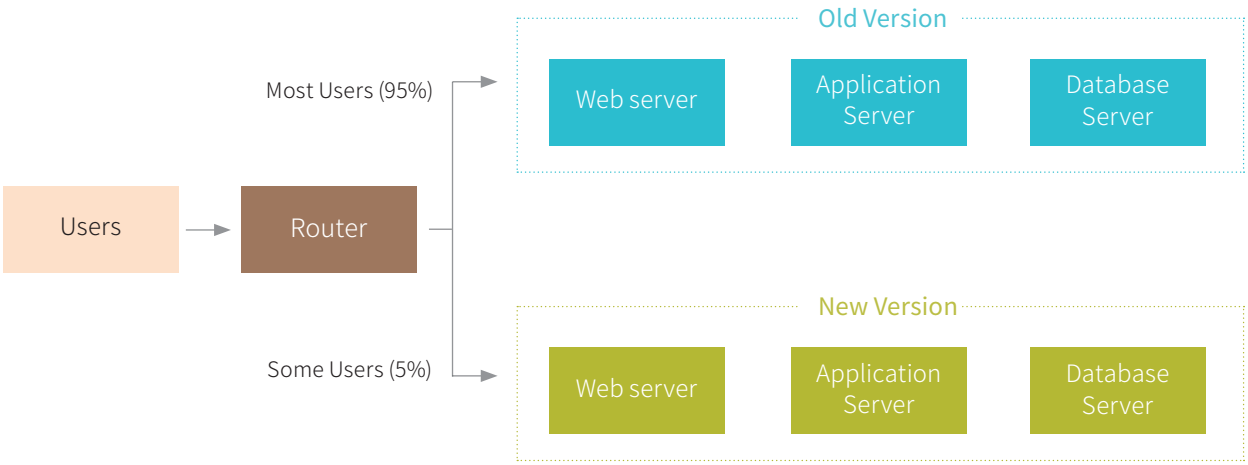
单纯技术性的数据不足以对业务实验提供反馈，需要贯串用户体验，获取对业务有指导意义的数据。一个可供参考的框架是“海盗度量”：聚焦关注创新业务的获客、活跃、保留、推荐、创收（AARRR）这5个环节，从这个5个点上提出假设，然后用数据来证实或证伪假设。

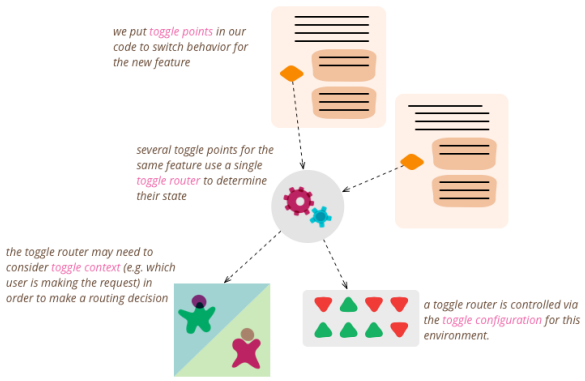


2 金丝雀发布

金丝雀发布是一种控制软件发布风险的方式：在把新版本发布给全部用户之前，首先发布给一小部分用户，确保功能完好可用。金丝雀发布的主要目的是降低风险。新的软件可以先在与用户隔离的环境中接受UAT测试；如果新的软件有问题，受到影响的只是一小部分用户，不至于立即造成巨大的损失；如果新的软件有问题，可以立即回滚到旧版本。

金丝雀发布最基本的形式，就是在前端反向代理上用路由技术把一定比例的用户导向“金丝雀”版本（例如Nginx可以支持多种筛选用户的方式）。在路由技术的背后，应该以凤凰服务器和不可变服务器来实现每个服务，服务的创建和回收应该是完全自动化的。同时还需要端到端的综合监控，根据有业务语义的目标（例如转化率）是否发生突变来判断金丝雀的效果。





3 Toggle架构

Feature Toggle的目标是，通过架构设计允许团队在不修改代码的前提下改变系统的行为。常见的需要Toggle的场合包括：避免多个交付版本的代码branching/forking；避免未完成功能的代码branching/forking；运行时动态改变系统行为，以实现一些特定能力，例如：线上受控实验、针对不同用户权限提供不同服务、回路熔断和服务降级等。

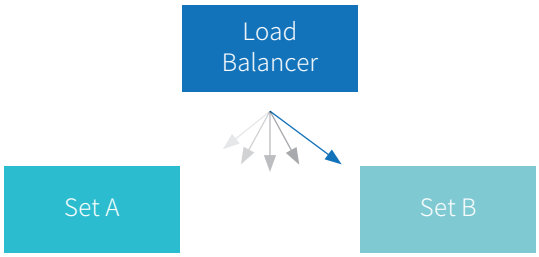
常见的Toggle可以分为4类：

- Release Toggle：某些功能已经存在，但暂时不向用户发布。主要目的是为了基于trunk开发、避免开发分支。静态，生存周期短。可以用Togglz之类工具。
- Ops Toggle：回路熔断，高负载或发生故障时自动降级服务。较动态，生存周期长。工具如Hystrix等。
- Experiment Toggles：用于支持线上实验（例如Canary Release、A/B Test等）。动态，生存周期较短。采用路由技术实现。
- Permission Toggles：用于给不同权限的用户提供不同的服务。引入统一的toggle router和toggle configuration，避免在代码中写条件。动态，生存周期长。

Feature Toggle也应该以服务API的形式暴露出来，并且鼓励用结构化的、人类可读的配置文件管理Toggle。

4 路由技术

通过路由切换的方式，让用户在不同的时间、不同的场合访问到不同的服务实例（可能是不同的版本）。路由技术可以用来支撑多种实验性部署技巧，包括蓝-绿部署（零宕机部署）、A/B测试、金丝雀发布等。这篇文章介绍了这些部署技巧直接的关系。



路由技术的实现与下层的弹性基础设施有很大关系，以AWS为例，有几种比较简单的实现蓝绿部署的方式：

- 对于单个EC2实例，可以修改它的Elastic IP
- 对于EC2集群，可以切换ELB背后的Auto Scaling Group
- 可以用Route53修改DNS重定向
- 可以用BeanStalk切换整个应用环境（如果应用部署在BeanStalk上）

Cloud Foundry也有一组实现蓝绿部署的最佳实践。

5 可视化和埋点

通过埋点获得系统运行时的信息，收集之后显示出来，从而把运维环境中的信息及时反馈回开发团队，缩短反馈周期。

常见的埋点方式有：

- 代码中埋点（例如New Relic、AppDynamics、Dynatrace）
- 监控进程（例如StatsD、collectd、fluentd）
- 日志（例如Splunk、ELK）

数据需要用一体化的、直观可视的仪表板展示出来，从而快速指导业务调整。Grafana和Kibana等工具提供了很好的仪表板功能，不过还是需要针对自己的情况加以定制。

小结

很大程度上，大部分组织的IT建设都谈不上“科学”。科学的基础建立在假说和实验之上，而在很多组织里，“有可能失败的项目”恐怕根本无法立项——更不用说“很有可能失败的项目”。降低做实验和犯错误的成本、从经验中尽可能多的学习，是企业面对未知世界的唯一出路。然而快速的受控实验背后隐藏的是基础设施、软件架构、数据等多方面的技术支撑，把实验基础设施作为企业数字化旅程的阶段性目标，拉动各方面基础能力的建设，是建设数字平台的合理路径。

第五支柱

客户触点技术


2:09 PM
Save images
Popular
CHEAT SHEET: THE WAISTCOAT LIST
STONE
CUSTOM BUILT

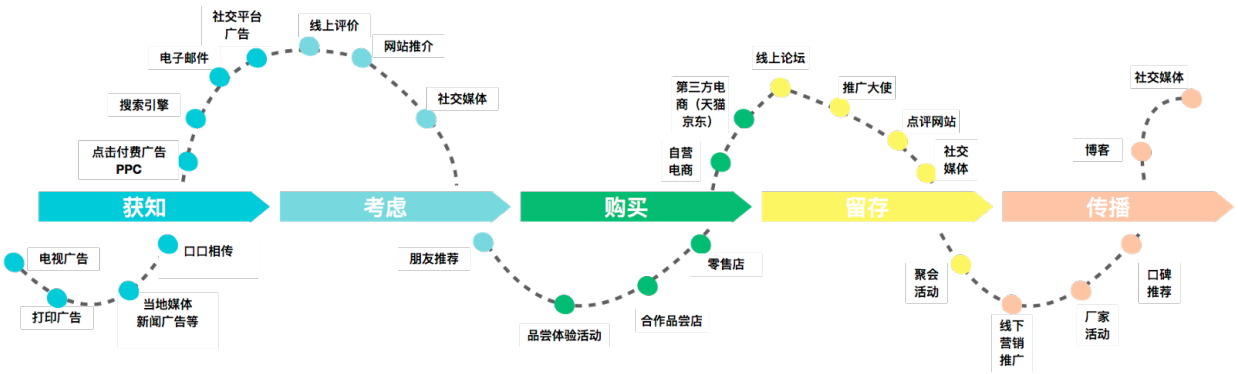


图1 企业的线上线下多样化触点

什么是客户触点技术

随着科技的发展，客户与企业的互动过程中产生了线上线下非常多样化的触点。图1展示了一个啤酒企业在客户生命周期的获知、考虑、购买、留存、传播不同阶段的线上线下触点。不仅仅是啤酒，家电、汽车企业，甚至金融也都类似。全渠道成为新常态，企业需要通过多样化的触点技术向顾客提供随时随地、连贯一致的用户体验。

以亚马逊书店为例，线下也能提供与线上一致的体验，如“一进门的推荐货架”，“每本实体书配有评论卡(Review Card)，可以看到读者评论”，“相似图书的推荐 (If you like…, you' ll Love) ”，“以及与线上的同一价格”。而做到线上线下书店拥有几乎完全一致体验的前提是整个企业需要:

- 建立对其顾客和目标顾客的唯一、连贯、准确、整体的视图，从而更好地了解和服务顾客;
- 结合顾客的特征和不同数字渠道的特征建立连贯的内容策略;
- 在多种渠道之间引导顾客的消费旅程，与顾客产生正确时间、正确地点、正确方式的交互;
- 基于从各种渠道获得的顾客本人及其行为的数据分析，向顾客提供定制化的内容、服务和产品推荐;
- 作为必要的技术保障，所有数字渠道的软件应用（尤其是原生的Android和iOS应用）都应该实践持续交付，提供全渠道地快速响应。

交付基础设施	API和架构治理	数据自服务	创新实验基础设施和监控体系	客户触点技术
快速交付	构建生态系统	获取洞见	有序地创新	一致的体验
弹性基础设施	开发者体验（DX）	数据流水线设计	数据采集	统一客户视角
持续交付流水线	服务边界	实时架构和API	金丝雀发布	内容战略
部署运行时	事件驱动架构	数据湖设计	Toggle架构	个性化
监控	公共网关	数据即产品	路由技术	移动应用持续交付
安全	微服务SOA拓扑	细粒度授权	埋点和可视化	跨渠道引流

1 客户触点技术解读

单一客户视图和个性化营销

单一客户视图（SCV）是组织对其顾客和目标顾客绘制的唯一、连贯、准确、整体的视图。客户在不同的生命周期中，在不同触点产生不同类型、不同结构的各类数据：人口/家庭特征及联系数据、社交媒体数据、市场活动互动数据、交易数据、用户行为数据，以及其他非结构化的各种数据，如社交媒体上的评价，各类服务请求等。只有当这些异源异构的数据有机的组合在一起，形成“单一客户视图”，才能准确衡量客户的客户终身价值(CLV)、在各个渠道上提供一致的用户体验、更有效地进行交叉销售(Cross-sell)和追加销售（upsell），进而留存客户。

一个典型的建立单一客户视图并实现个性化营销的方案，包括：

- 采用数据流如Kafka、Flume、Flink技术来采集数据进入数据湖；
- 利用Spark Streaming进行实时数据分析；
- 数据的清洗、过滤、整合、身份关联，建立单一客户视图；
- 同时，将相关的产品、销售、订单、渠道触点等信息也通过数据集市展示出来；
- BI及Analytics分析系统建立智能分析模型如“客户终身价值”、“下一步最佳行动”等；
- 营销活动系统发起“客户留存”、“交叉销售”、“追加销售”、“最佳渠道”、“潜力客户转化（Most Look Alike）”等营销活动行程进行智能个性化营销。

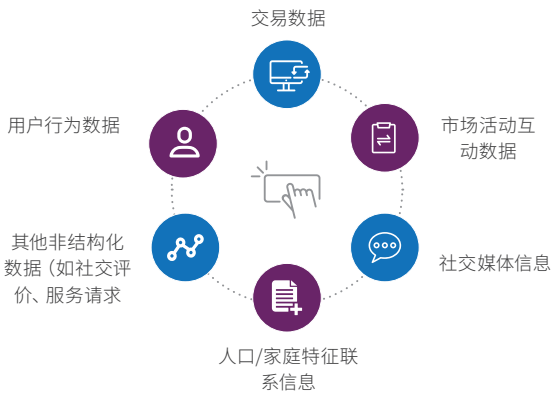


图2 异源异构的客户数据有机组装成“单一客户视图”

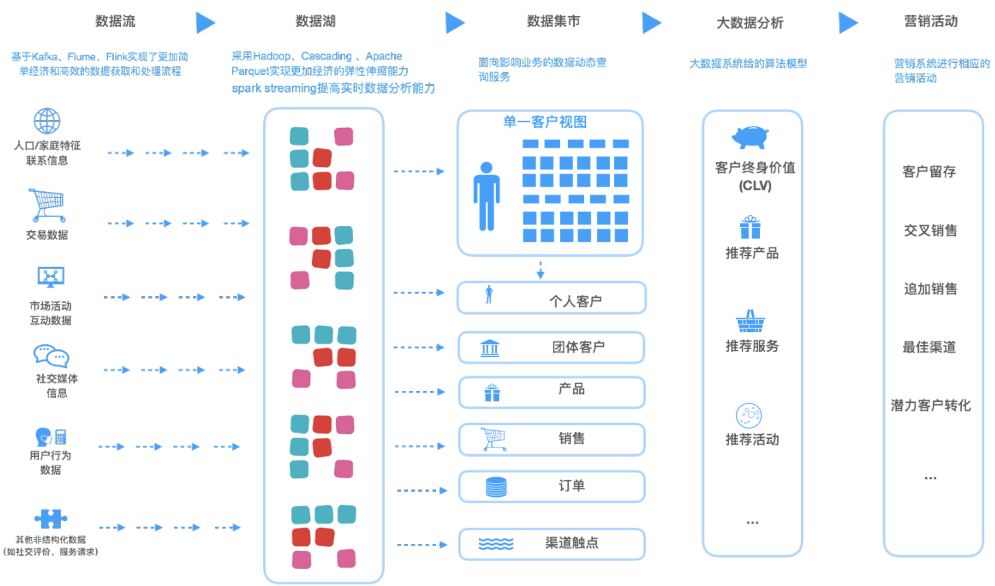


图3 基于数据湖的单一客户视图和个性化营销的架构方案

2 内容策略

当多样化的触点形成以后，内容的推送和服务也要相应地在正确的时间、在正确的渠道、推送给准确的目标群体。图4展示了这个逻辑流程：

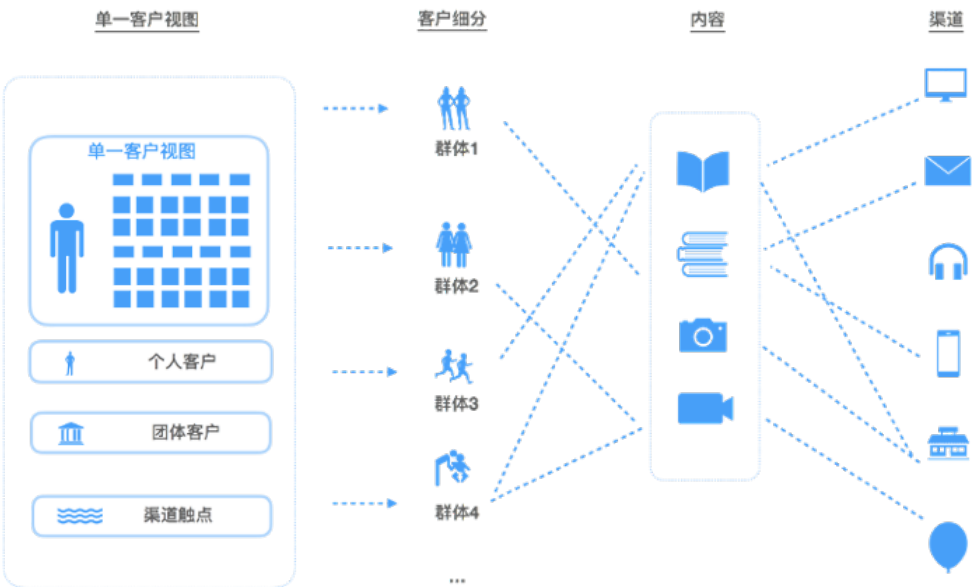


图4 基于单一客户视图，建立细分客户群体，设计不同内容，在合适的渠道进行推送

比如一个在线购物平台，针对孕妈妈做内容策略：数据显示82%的孕妈妈每周做一次线上咨询，内容类型是可以互动的、图文结合展示不同孕期内的信息资讯，通过电脑、手机、售货亭、社交网站推送；有67%的孕妈妈则是订阅每周的邮件，内容则是针对性的最新孕期知识；78%的孕妈妈们会经常浏览孕期及产前产后的博客，推送的内容则是不同年龄段孕妈妈写的各种博客。基于单一客户视图，对客户群的细分管理可以采用Adobe Audience Manager；在内容端用Adobe Experience Manager来管理；用Adobe Target和数字化营销系统完成内容定向推送。

3 跨渠道引流

全渠道时代，用户可以在任一环节便利地切换到最舒服的渠道和触点来继续服务。用户在整个服务过程中得到的信息是透明的、一致的。

图5 一个零售服装企业的跨渠道引流、决策、购买、交付、留存和传播

在图6展示的汽车企业的例子中，通过移动端营销活动和展示，用户可以继续在移动端、经销商、直营店对钟爱的车型，通过VR看车对车进行深入了解，然后在4S店预约试驾，完成购买。其中，二维码扫描、拍照购物（图片搜索）、虚拟现实/混合现实、在线个性定制等技术是跨渠道引流的支点。

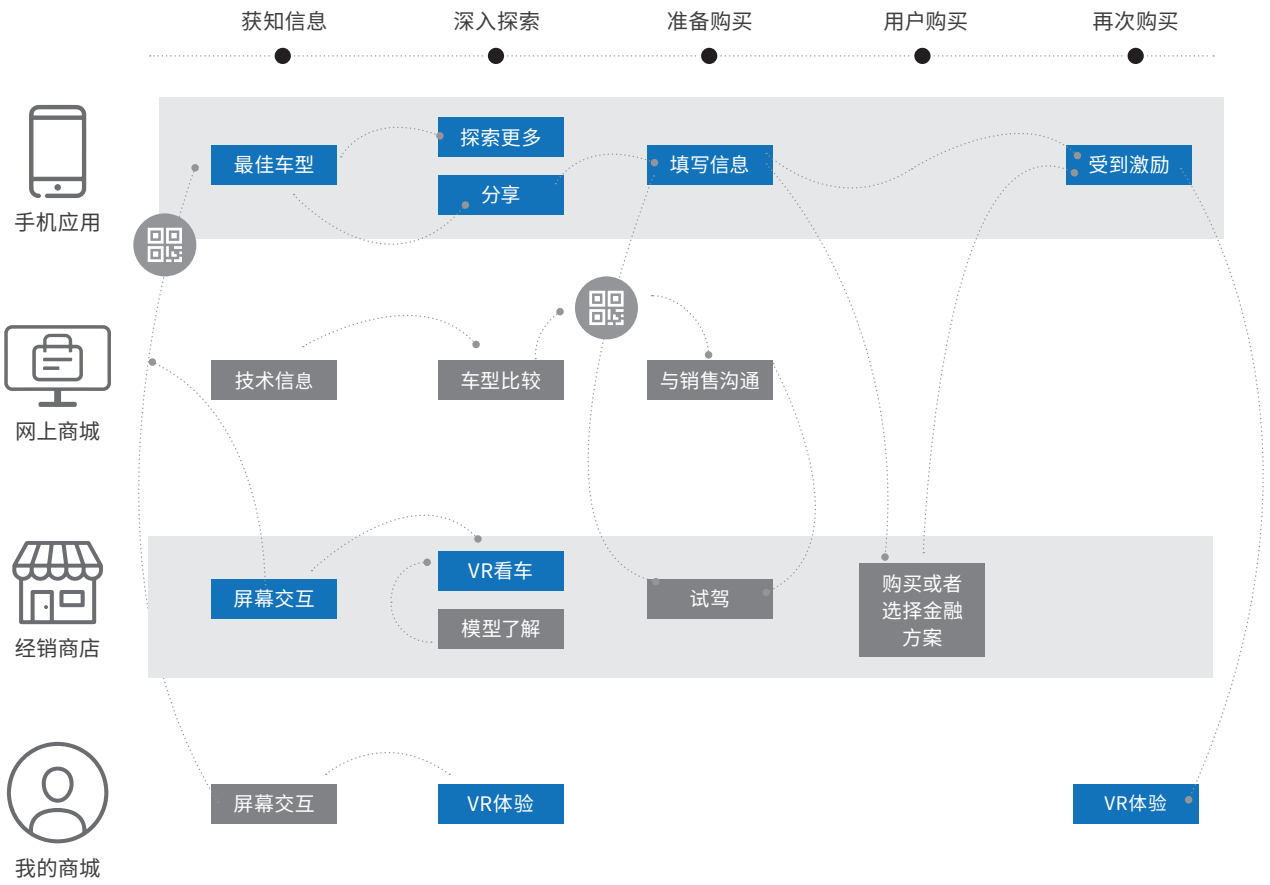


图6 一个汽车企业的跨渠道引流、决策、购买、交付、留存和传播

4 移动应用持续交付

移动应用仍是当前全渠道中的一个核心触点。如果移动应用不能加快交付周期，与Web端做到同步持续交付，就会导致用户在Web端和移动端体验不一致，进而导致客户流失。

事实上，现在应用商店的审核速度已经有了非常大的提升。作为开发者，移动应用可以通过持续集成和自动化测试加快提交审核的速度；当然，我们也有一些技术能够绕过应用商店，直接更新，但是这样有app被下架的风险。

相关的技术方案有：

- Fastlane是常用的iOS及Android自动构建工具集，功能覆盖了移动应用从创建、证书管理、构建、运行测试、打包到发布整个流程，配置简单、功能完善；
- 单元测试框架成熟；
- 使用截图测试来保证我们的UI正确性；
- 用Appium / Calabash编写运行移动验收测试。

小结

全渠道已是新常态。获取便利舒服的体验、个性化的服务，是消费者的一贯需求。然而现实中所谓的“个性化推送”往往变成了“垃圾信息轰炸”。麦肯锡报告显示，98%的受访社交媒体用户都在社交媒体上收到过广告，但只有18%的人认为收到的推荐“投其所好”。对于线下购物体验，只有10%的消费者表示在店铺得到了个性化的服务或建议。究其原因，重点还是在于客户数据的完整性、连贯性、准确性和统一性。在英国，只有16%的企业拥有有效的单一客户视图。扎实做好单一客户视图这个数据基础，应用合适的内容策略，通过创新的服务设计支持用户在不同触点之间便利流转，移动端持续交付提供透明一致的信息和服务，才能把触点技术发挥到极致，塑造真正的数字化平台消费者体验。

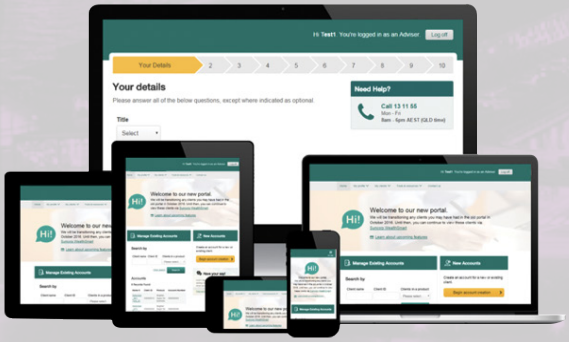
DPS案例

澳洲某大型金融公司

客户背景

我们的客户是澳大利亚最大的金融服务集团公司之一，其资产总值接近1000亿澳币，年收入达到170亿澳币，员工超过1万人。该客户业务主要涵盖个人险、商业险、寿险、银行及针对新西兰市场的服务，旗下包含二十多个品牌和众多产品线，其服务的900万(澳洲总人口约2400万)客户主要来自澳大利亚和新西兰。

ThoughtWorks和该公司的合作超过十年，分别在中国和印度设有离岸交付中心，ThoughtWorks一直以来都是该公司非常重要的战略性合作伙伴。从2007年开始，我们在3年内帮助该客户的IT部门完成了组织敏捷转型，不但在集团内部建立起了敏捷文化和敏捷学院，更重要是的帮助它在澳洲市场树立起了敏捷领导者的品牌形象。并在历时5年的遗留系统简化战略中，帮助客户每年节省超过2亿澳币的维护成本。



Online conversion rate increased (Home Insurance)	81% ↑
Online conversion rate increased (Car Insurance)	509% ↑
Budget Saved (Personal Insurance)	55% ↓
Delivery Time Shortened (Personal Insurance)	59% ↓
Average Idea-to-market time Shortened (Personal Insurance)	70% ↓

机遇与挑战

澳洲人口仅2400万，在这个竞争激烈的市场上，对于已经形成稳定客户群的成熟公司来讲，发展新客户的成本是非常高的。为了寻找到新的价值增长点以支持未来3-5年的业务增长，我们的客户将主要关注点转移到了现有的900万客户身上，并在2016年底启动了市集平台战略。

该平台战略 (Platform Strategy) 主要希望借助零售业的经验，为其客户打造一个丰富的产品市集平台。平台内，不但能够集成其当前已有的丰富的保险类产品以及银行相关服务，同时，还有望整合其上下游的合作服务供应商，形成横向和纵向的服务交叉网，从而为其现有客户提供全面的、一致的、无缝的用户体验和全方位、多样化的金融服务，从现有客户身上找到新的价值增长空间。

产品市集平台需要集成多个后台系统，并在前端给用户提供全面、一致、无缝的体验。为了达成这一愿景，IT团队面临几方面的挑战：

- 前端后端多个团队配合集成，如何让各个团队高效工作，缩短整个系统交付的周期？
- 后端的保险和银行核心系统都是多年积累的遗留系统，如何划分隔离边界，为前端提供高内聚、低耦合、优雅易用的服务接口？
- 系统内部环境复杂，对外又是直接面对消费者的门面，如何实现零宕机部署，如何及时发现系统故障，如何从故障中快速恢复？
- 如何透过IT系统形成对用户的深入理解，为用户提供个性化的增值服务？

DPS如何支撑业务愿景

市集平台提供4大模块功能：

1. 金融产品推荐。根据用户信息，定制推送适合的投资、理财、保险等金融产品，实现跨品牌、跨产品交叉销售。
2. 保单自助管理。该集团下属十余个保险品牌、三大类保险产品（财险、寿险、医疗险），用户都可以在平台上自助查询和管理。
3. 保险自助理赔。用户可以在平台上申请理赔、查询理赔进度。
4. 银行自助服务。集团下属的银行业务，也可以通过市集平台便利访问。

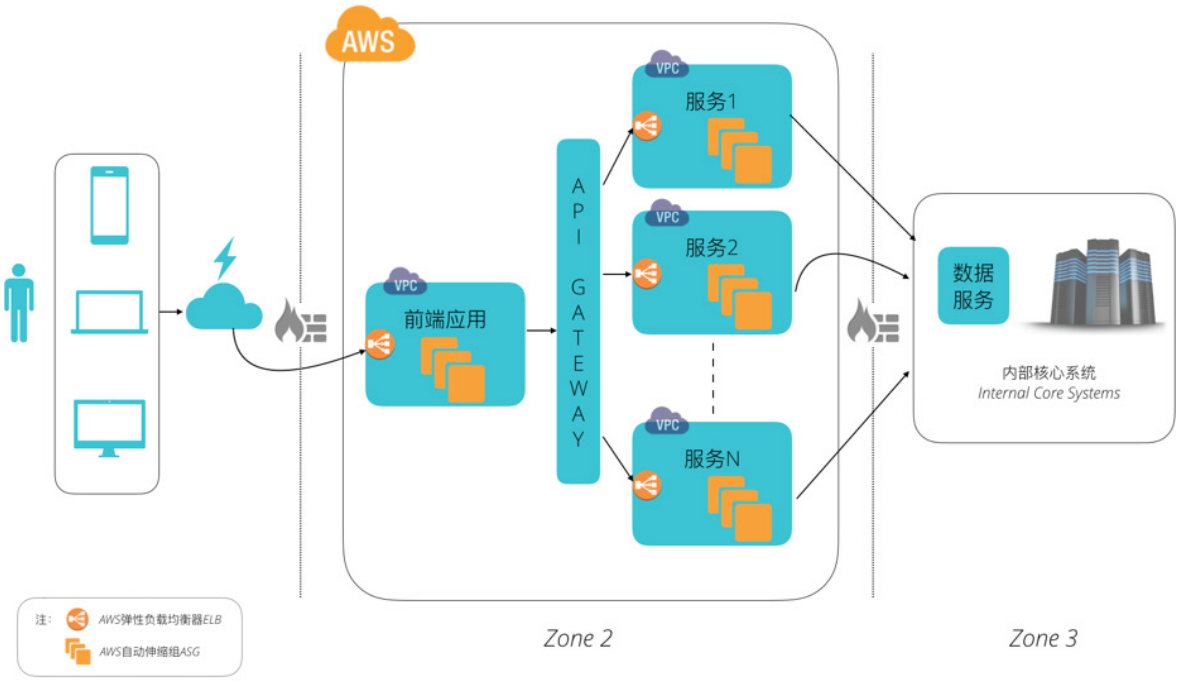
交付基础设施支撑快速交付

市集平台系统在AWS虚拟私有云的基础上构建了云原生（Cloud Native）的弹性交付基础设施。

基于Jenkins 2的持续交付流水线，对每次代码提交执行完整的构建过程（包括静态检查、单元测试、覆盖率检查、契约测试等环节）。构建成功的产物打包进入Nexus，供后续多个环境选择部署。

构建成功的发布候选版本会流经系统测试、用户验收测试、最终产品验证等环境，然后部署上线。各个环境都采用AWS提供的ASG（自动伸缩组）管理，环境的开通和配置完全自动化，各个环境与生产环境高度一致，确保验证可靠性。

交付团队采用Splunk监控各个环境的日志，并汇总形成可视化dashboard，不仅可以帮助开发人员了解系统运行内部情况，而且在系统故障发生时及时提供告警和诊断信息。



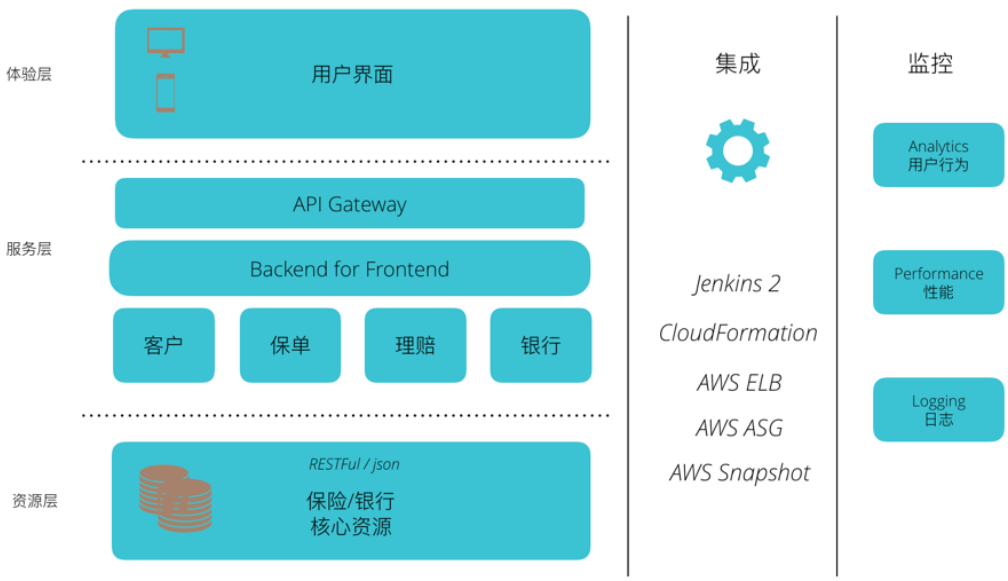
服务化架构激活核心资产

市集平台系统采用服务化架构。整个系统架构分为3层：

1. 资源层。在现有核心保险和核心银行系统之上封装一层RESTful API，以JSON格式的服务接口对其他系统暴露集团的核心资产：保险和银行业务相关的数据与能力。
2. 服务层。对资源层提供的核心保险和核心银行数据与能力加以整合适配，提供顾客信息、保单信息、理赔信息、医疗保险处理能力（由第三方合作伙伴提供）、银行账户信息等业务服务，提供简洁易用的服务接口，并在业务服务之上用Axway提供服务网关。
3. 体验层。提供Web和Mobile两种前端渠道，基于业务服务实现前端一站式用户体验。

为了支撑服务化架构，市集平台提供了完善的微服务基建体系：

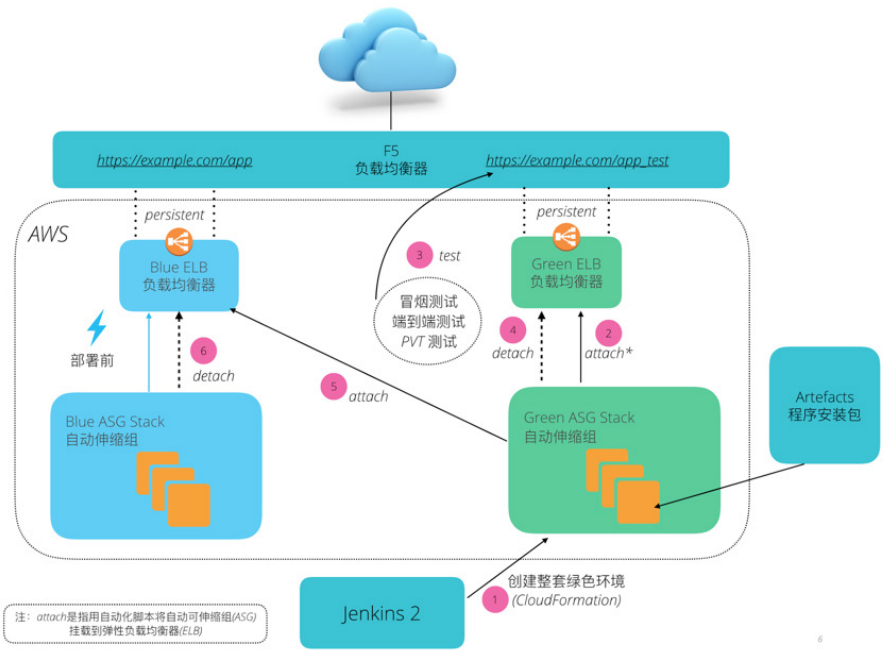
- 使用Eureka提供服务注册发现能力，方便服务组装。
- 使用Hystrix提供回路熔断能力，在负载过大或服务发生故障时优雅降级。
- 使用RxJava实现异步请求处理能力，提升前端应用响应速度。
- 采用Spring Boot开发微服务，复用业界成熟的微服务工程经验。



部署和监控技术赋能创新实验

市集平台采用AWS提供的ASG和ELB（弹性负载均衡）等能力实现了蓝-绿部署。只需点击一个按钮，即可自动完成ASG创建、应用部署、ELB切换等步骤，零宕机实现系统部署。如果新版本发生异常，Splunk监控系统会提醒运维人员，同样可以一键式回滚到前一个可靠的版本。

市集平台采用Ensighten监控用户在前端应用上的行为，并用Splunk监控用户与后端系统的交互。将这些信息汇总，就能勾画出完整的用户旅程，随时了解系统关键节点的运营指标。



个性化内容强化顾客黏性

市集平台获得的用户行为数据会流入集团的客户数据仓库，和集团其他系统中已有的客户数据一起用于建立对客户的全面理解。客户可能购买了一个品牌的房屋险，在另一个品牌购买了寿险，在第三个品牌的银行里有存款账户…… 这些信息都会被汇总成为对这个客户的统一视图。

基于对客户的统一视图，市集平台系统会给客户推送个性化的内容与产品，将他们引流到最适合的品牌和渠道上，协助他们购买自己需要的金融产品，把他们培养成更忠诚的客户。

小结

市集平台系统为了应对前后端多团队配合、多个遗留系统需要整合、需要频繁部署上线、需要从现有客户身上挖掘价值增长点等诉求，从DPS的交付基础设施、API和架构治理、实验基础设施、客户触点技术这4个支柱入手，建立起了一个高效的数字平台，使IT交付团队能够基于企业多年积累的核心资产，为客户创造全面、一致、无缝的体验。

该系统的数字平台战略地图如下:

交付基础设施	API和架构治理	数据自服务	实验基础设施	客户触点技术
弹性基础设施	**开发者体验**	数据流水线	**数据采集**	**统一顾客视角**
持续交付流水线	**服务边界**	实时架构 和API	金丝雀发布	内容战略
部署运行时	**事件驱动架构**	数据湖设计	Toggle架构	**个性化**
监控	**公共网关**	数据即产品	**路由技术**	移动持续交付
安全	**微服务** **SOA拓扑**	细粒度授权	**埋点和可视化**	**跨渠道引流**

DPS案例

全球房产网络

客户背景

我们的客户是澳大利亚知名的在线房地产信息提供商，市值近百亿澳元，年度营业收入超过6亿澳元，全球有1500名员工，为56个国家的消费者提供购房、租房、房屋装修、商业地产等房地产相关信息服务。

ThoughtWorks与该公司有长期的战略合作关系。从2011年起，ThoughtWorks在西安为该公司设立了离岸交付中心。2012年西安办公室成立了CASA和RCA团队，在随后的五年间，团队数量也随业务的发展而持续增加，至2017年已有15个开发团队，总人数接近100，覆盖客户七大业务线。

机遇与挑战

在经营澳大利亚地产信息的过程中该公司发现，来自外国买家的销量占到整个新房产销量的10.4%。为了发掘全球化、跨地域房产消费的巨大潜力，该公司将自己十余年来在线经营澳大利亚房地产信息服务的经验复制到北美、欧洲、亚洲等多个其他地区，迅速将业务扩展到12个网站，覆盖56个国家的房源信息。这个全球化的房产网络触角，每月能触及超过2亿客户，为各种顾客提供新的价值增长点：

- 购房者通过本地化的网站快速获得全球56个国家的房产信息，为就业、移民、全球化资本配置等购房需求提供一站式信息服务。
- 房地产开发商通过全球房产网络让其他地域的消费者了解自己开发的楼盘，创造新的销售机会。
- 房产销售代理通过全球房产网络在不同地域展开促销活动，结合本地特色建立全球销售渠道。

另一方面，全球房产网络的业务愿景给IT团队提出了新的挑战：

- 数据来源复杂。房产数据从不同的地区、不同的数据源获取，面临网络延迟、数据格式不统一、信息不完整或信息重复等问题，如何快速、准确地把房产信息导入到后台核心系统？
- 前端系统多样。房产信息数据源以及12个前端网站分布在不同的区域、使用不同的技术栈，如何给不同渠道的终端应用提供一致、统一的数据服务？
- 用户体验。顾客从不同的渠道（移动终端、个人电脑等）、不同的区域（美国、意大利、法国、中国、香港、新加坡、马来西亚等）检索房产信息，如何提供既有本地特色、又体现公司整体品牌一致性的优质用户体验？
- 响应变化。数据采集、处理、本地化、分级存储、提供给前端网站使用，流程长、环节多，如何快速响应业务变化，迅速调整后端系统提供顾客需要的数据？

DPS如何支撑业务愿景

全球房产网络的系统架构分为几个主要的层次。来自三个大洲的房地产数据经过整理和全球化处理后，存入分布式的数据湖；在数据湖之上，该系统以微服务架构对前端应用提供服务接口；前端应用针对本地特色、基于公司整体品牌设计用户体验。

云原生交付赋能高效团队

全球房产网络采用AWS虚拟私有云搭建交付基础设施。S3弹性存储和Lambda弹性任务使系统能良好应对数据量的起伏。

采用Bamboo　Server实现了持续交付流水线，因为它能够很好地集成Git分支工作流，并且无缝地与JIRA缺陷管理、任务追踪和项目管理工具进行集成。在持续交付流水线的基础上，团队实现了多层面的测试覆盖网：单元测试选用JUnit + Mockito，API测试使用Jasmine + RSpec，UI自动化测试用Selenium WebDriver + Cucumber实现。

实际部署有开发、模拟和生产三类环境，而且生产环境分布在东京和美国两个区域，项目采用业内广泛使用的Ansible自动化部署工具来应对复杂的部署环境以及如此高的自动化要求。

系统所有的服务都是部署在AWS上，计算资源监控主要使用Cloudwatch；由于数据处理流水线是系统的核心依赖，数据处理状态的监控采用了Grafana图形化监控工具；程序的日志监控采用Splunk。

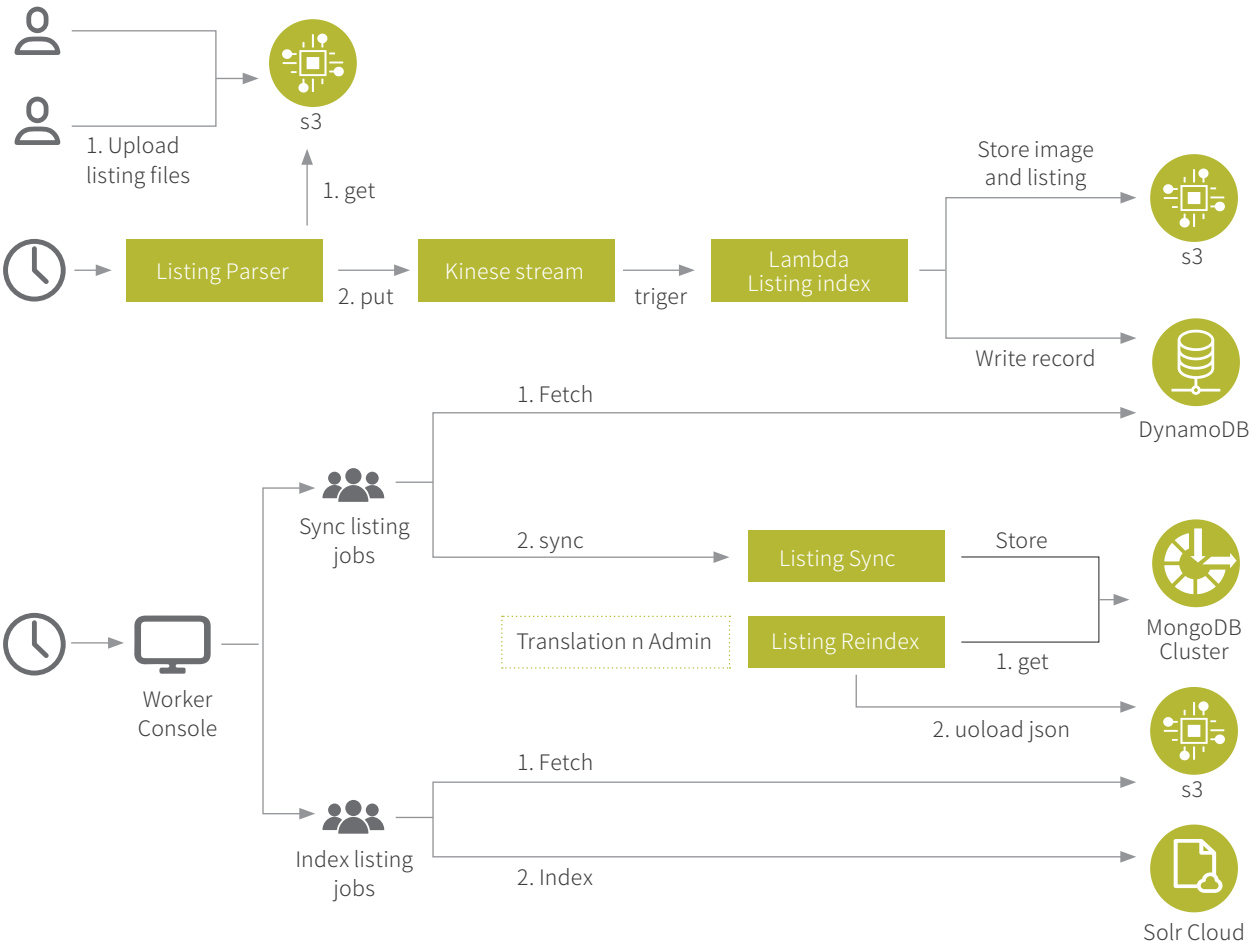
服务化架构支撑丰富前端

系统采用微服务架构，各个服务采用Scala、Java、Ruby等不同语言、不同技术栈开发。针对每个前端应用提供BFF (Backend For Frontend) 服务，统一的服务网关提供身份认证、鉴权功能，负载均衡使用AWS Load Balancer。

微服务架构的核心是服务的自治性和相对独立性，不同服务之间的依赖必须是松耦合的，依据这个基本原则，系统使用 AWS SQS 和 AWS Kinesis 来解耦服务之间的依赖关系，并且服务之间交换消息采取异步方式。

自服务数据提速业务响应

全球房产网络系统基于S3云存储和Lambda弹性任务建立了数据流水线，自动完成数据采集、整理、分级存储全过程。来自三大洲不同数据源的房产数据在24小时内即可提供给全球用户检索查看。



房产数据被爬虫抓取后，暂存于DynamoDB中，基于Lambda的弹性任务对原始数据进行清洗整理，抓取房产信息链接的资源文件（例如图片等），经由人工参与内容国际化，将整理后的原始数据存入MongoDB集群。

自动化的重建索引任务从原始数据湖中抽取核心信息，在Solr Cloud中建立索引，提供给房产检索服务。数据湖中保留全量房产信息，开发团队可以根据前端需求快速调整索引字段。

多维度路由实现受控实验

新功能上线采用A/B Testing策略，系统并没有采用部分流量导入方案来进行A/B测试，为了最大限度收集客户的反馈，系统采用预先规划不同时段的100%流量导入方案进行测试，收到了很好的反馈效果。

系统在用户最经常使用的用户触点（检索按钮、条件过滤器等）上进行了数据采集埋点，采集用户行为数据。用户数据采集、分析以及展现采用 Adobe Analytics (Omniture)，而对于搜索引擎优化的方面的实现主要是利用 Google Analytics 来实现的。

个性化体验服务全球用户

针对不同地域的顾客，前端应用提供符合当地语言、文化、场景、使用习惯的用户体验。结合用户体验的需求，通过BFF适配后端服务接口。

各个前端应用（包含桌面Web应用和移动应用）有各自的持续交付流水线，各个前端应用分别以各自的节奏演进，避免交付节奏上的依赖。

小结

一家发源于澳洲的房产信息网站为了应对全球化过程中出现的挑战，遵循DPS五个支柱的指导原则建立了适合自己的数字平台，有效应对了全球房产信息网络中数据来源复杂、前端系统多样的挑战，使交付团队得以及时响应变化，为顾客提供既有本地特色、又体现公司整体品牌一致性的优质用户体验。

全球房产网络系统的数字平台战略地图如下:

交付基础设施	API和架构治理	数据自服务	实验基础设施	客户触点技术
弹性基础设施	开发者体验	**数据流水线**	**数据采集**	统一顾客视角
持续交付流水线	服务边界	实时架构和API	金丝雀发布	内容战略
部署运行时	事件驱动架构	**数据湖设计**	Toggle架构	**个性化**
监控	**公共网关**	数据即产品	**路由技术**	**移动持续交付**
安全	**微服务** **SOA拓扑**	细粒度授权	**埋点和可视化**	跨渠道引流

虽然DPS的五大支柱都是聚焦于技术，我们深知组织结构和文化支撑的重要性。如果我们此时正在与公司领导就DPS的五大支柱展开对话，值得提醒的是，很有可能因为组织结构不支持这些运作方式而失败。团队组织结构、投资预算方式、实验心态等也是必定要涉及的。

在数字化的浪潮面前，传统企业不必恐惧于互联网企业的技术优势。ThoughtWorks数字平台战略，能帮助您很好地管理多变复杂的环境，促使快速落地，激发创新，创造新的商业机会，激活自己的核心资产，在新的竞争环境中找到自己的一席之地。

了解更多DPS内容：*https://www.thoughtworks.com/cn/digital-platform-strategy*

联系我们：*business-insights@thoughtworks.com*

ThoughtWorker出版、翻译的书籍

ThoughtWorks一直追求卓越的技术，ThoughtWorks内部也有很高的学术氛围。这些书籍可以说是一群极有天分的软件精英的思想和观点的汇聚，是他们多年的宝贵实践经验的凝结。

《ThoughtWorks技术雷达》

为了体现技术卓越，ThoughtWorks全球技术委员会（TAB）定期讨论技术战略，分析对行业产生重大影响的最新技术趋势，这便是我们看到的自2010年起每年两度的《ThoughtWorks技术雷达》。

技术雷达官网：
https://info.thoughtworks.com/technology-radar-subscription-cn.html

技术雷达官网

ThoughtWorks是一家集合了在软件咨询、交付以及产品领域富有激情并且极具前瞻性的技术工作者的软件公司。我们利用颠覆性思维帮助客户成就非凡使命，同时致力于IT产业乃至社会的变革。我们为追求卓越的软件团队提供创造性的工具。我们的产品帮助企业不断进步，不断交付满足他们重要需求的高质量软件。ThoughtWorks已经从20多年前一个芝加哥的小团队，成长为现在拥有超过4500人，分布于全球15个国家，拥有42间办公室的全球企业。这15个国家是：澳大利亚、巴西、加拿大、智利、中国、厄瓜多尔、德国、印度、意大利、新加坡、南非、西班牙、土耳其、英国、美国。

ThoughtWorks商业洞见官方微信

了解案例详情， 请联系:

business-insights@thoughtworks.com