/thoughtworks

TECHNOLOGY RADAR

有态度的前沿技术解析

Volume 24

#TWTechRadar
thoughtworks.com/radar

贡献者

技术雷达由 ThoughtWorks 技术顾问委员会创建。

中国区技术雷达汉化组：

黄进军 / 伍斌 / 姚琪琳 / 张凯峰 / 包欢 / 边晓琳 / 邓奕 / 樊卓文 / 方明 / 韩盼盼 / 李陈泽 / 李君丽 / 李康宁 / 梁凌锐 / 梁若琳 / 刘司琪 / 刘小清 / 孟然 / 闵锐 / 宋烨纯 / 童圣 / 王启瑞 / 向博 / 徐瑾 / 徐培 / 杨光 / 杨慧娟 / 杨璐璐 / 杨旭东 / 杨洋 / 杨召 / 张昂 / 张霄翀

同时感谢思译社对本次技术雷达汉化工作提供的帮助。

Rebecca Parsons (CTO)

Martin Fowler (Chief Scientist)

Bharani Subramaniam

Birgitta Böckeler

Brandon Byars

Camilla Crispim

Cassie Shum

Erik Dörnenburg

Evan Bottcher

Fausto de la Torre

Hao Xu

Ian Cartwright

James Lewis

Lakshminarasimhan Sudarshan

Mike Mason

Neal Ford

Perla Villarreal

Rachel Laycock

Scott Shaw

Shangqi Liu

Zhamak Dehghani

关于技术雷达

Thoughtworker 酷爱技术。我们的使命是支持卓越软件并掀起 IT 革命。我们创建并分享 Thoughtworks 技术雷达就是为了支持这一使命。由 Thoughtworks 中一群资深技术领导组成的 Thoughtworks 技术顾问委员会创建了该雷达。他们定期开会讨论 Thoughtworks 的全球技术战略以及对行业有重大影响的技术趋势。

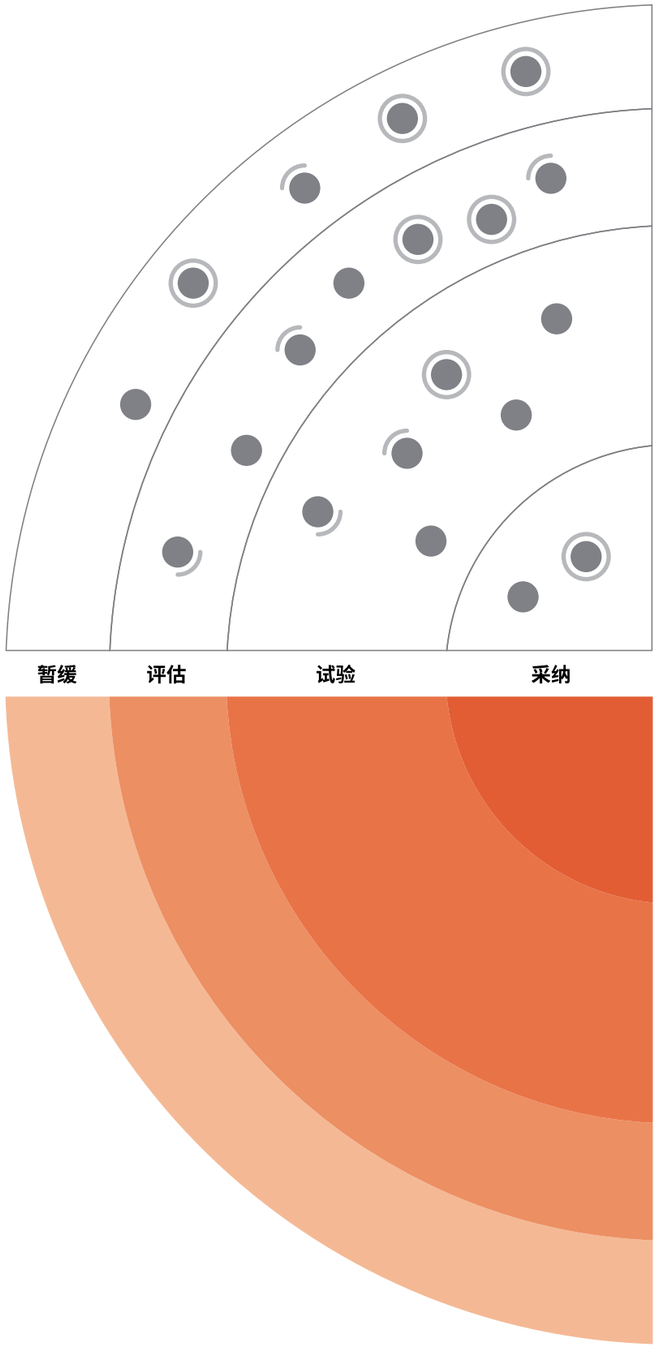
这个雷达以独特的形式记录技术顾问委员会的讨论结果，从首席技术官到开发人员，雷达为各路利益相关方提供价值。这些内容只是简要的总结，我们建议你探究这些技术以了解更多细节。这个雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言及框架。如果雷达技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。

想要了解更多技术雷达相关信息，请点击：
thoughtworks.com/radar/faq

雷达一览

技术雷达持续追踪有趣的技术是如何发展的，我们将其称之为条目。在技术雷达中，我们使用象限和环对其进行分类，不同象限代表不同种类的技术，而环则代表我们对它作出的成熟度评估。

软件领域瞬息万变，我们追踪的技术条目也如此，因此你会发现它们在雷达中的位置也会改变。



新的

移进/移出

没有变化

技术雷达是具有前瞻性的。为了给新的技术条目腾出空间，我们挪走了近期没有发生太多变化的技术条目，但略去某项技术并不表示我们不再关心它。

采纳
我们强烈主张业界采用这些技术。我们会在适当时候将其用于我们的项目。

试验
值得追求。重要的是理解如何建立这种能力。企业应该在风险可控的项目中尝试此技术。

评估
为了确认它将如何影响你所在的企业，值得作一番探究。

暂缓
谨慎推行

本期主题

加速产品上市的平台团队

越来越多的组织正在采用平台团队的理念来开发产品，即成立一个专门团队，来创建和支持内部平台功能（如云原生、持续交付、现代化的可观测性、认证和鉴权模式、服务网格等），并使用这些功能来加速应用程序开发，降低运维复杂性，并缩短产品上市时间。这种日趋成熟的做法值得欢迎，所以我们早在 2017 年，就将该技术引入技术雷达。但是随着成熟度的提高，我们发现组织在应用这项技术时，应避免使用一些反模式。例如，“用一个平台来统治一切”，可能并不是最佳选择。“构建一步到位的大平台”，可能要过数年后才能交付价值。本着“一旦建好，就有人用”的初衷，到头来可能却是巨大的浪费。相反，使用产品思维，有助于根据产品客户的需求，来弄清每个内部平台所应提供的服务。但如果让平台团队只解决技术支持工单系统中所提交的问题，那么这种做法就又产生了老式的运维孤岛团队，出现相应的需求优先级失调的弊端，如反馈和响应缓慢，以及争夺稀缺资源等的问题。此外，我们还看到一些新工具和集成模式涌现出来，以有效划分团队和技术。

整合的便利性盖过单项最佳方案

随着以自动化、规模化和其他现代目标为特征的工程实践在开发团队中变得越来越普遍，我们在许多平台（特别是在云领域）上看到了相应的面向开发者的工具集成。例如，工件库、源码控制、CI/CD 流水线、wiki 以及类似的工具，开发团队通常会手工挑选这些工具并按需拼接在一起。现在，诸如 Azure DevOps 等交付平台和 GitHub 等生态系统已经将许多工具收入囊中。虽然各平台产品的成熟度不尽相同，但交付工具“一站式服务”的吸引力是毋庸置疑的。总的来说，似乎需要权衡的是，拥有合并的工具栈可以为开发人员带来更多的便利和更少的工作，但这套工具集却很少能代表最佳的解决方案。

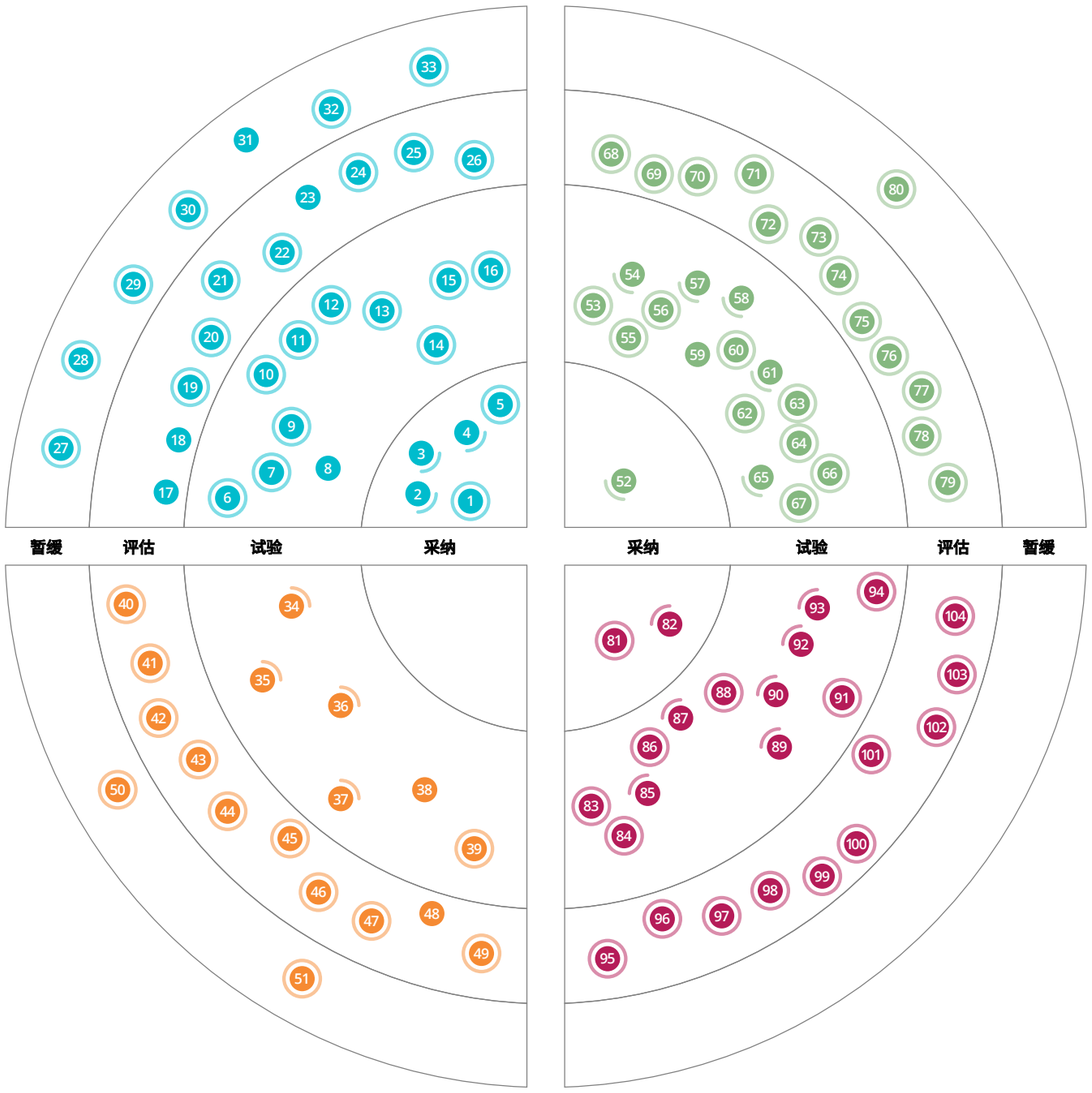
太复杂以至于无法进入雷达

在技术雷达的命名法则里，对许多复杂主题的讨论，最终状态都会是“TCTB——太复杂以至于无法进入雷达（too complex to blip）”，这意味着那些条目无法被分类，因为它们呈现出太多正反两面的特性，以及大量关于建议、工具的适用性和其他原因上的细微差别，这让我们无法用短短几句话总结它们。通常来说，这些主题会变成文章，播客，以及其他非雷达形式的内容。我们热烈参与的一部分讨论，聚焦于这些主题：它们很重要但过于复杂，无法形成一致的观点。许多主题经历了一次又一次会议——而且有些是跟我们的部分客户一起——，最终进入了 TCTB 的状态，包括 monorepos，分布式架构的编排指南以及分支模型等等。如果你好奇为什么这些重要的主题没有进入雷达，可以确信的是，这并不是因为我们缺乏意识或者意愿。就像软件开发中的许多主题一样，那里存在太多权衡，难以提供清晰明确的建议。我们有时也会发现，可以对一些大的主题中的小部分提供建议，使其进入雷达，但这些大的主题对雷达来说，仍然过于微妙难以确认。

识别架构耦合上下文

在软件架构中，如何在微服务、组件、API 网关、集成中心、前端等等之间确定一个适当的耦合级别，是几乎每次会议都会讨论的话题（参见上一主题“太复杂以至于无法进入雷达”）。随处可见的情况是，当两个软件需要连接在一起时，架构师和开发人员都在努力地寻找正确的耦合级别，许多常见的建议会鼓励极致地解耦，但这会使构建工作流变得非常困难。架构中的耦合涉及许多重要的考虑事项：事物如何连接、理解每个问题域中固有的语义耦合、事物间如何相互调用、如何保证事务性正常工作（有时还会与其他棘手的特性，如可伸缩性结合在一起）。除单体系统之外，如果没有某种级别的耦合，软件也就无从谈起；在现代架构中，找到正确的折衷方法来确定耦合的类型和级别是一项关键技能。我们确实看到了一些不好的实践，例如为客户端库生成代码，也看到了一些好的实践，例如明智地使用 BFF 模式。然而，通用的建议在这个领域是无用的，银弹是不存在的。我们需要花时间和精力去理解这些因素，然后因地制宜地做出这些决定，而不是寄希望于找到一个通用却并不恰当的解决方案。

The Radar



新的　移进/移出　没有变化

技术

采纳
1. API 扩张与收缩
2. 机器学习的持续交付 (CD4ML)
3. 设计体系
4. 平台工程产品团队
5. 服务帐户轮换方法

试验
6. 云沙箱
7. Contextual bandits
8. Distroless Docker images
9. Ethical Explorer
10. 假设驱动的遗留系统改造
11. 轻量级的 RFCs 方法
12. 最简单的机器学习
13. 单页应用 (SPA) 注入
14. 团队的认知负担
15. 使用工具管理 Xcodeproj
16. UI/BFF 共享类型

评估
17. 限界低代码平台
18. 去中心化身份
19. 部署漂移提示器
20. 同态加密
21. Hotwire
22. 微前端中的模块映射
23. 开放应用程序模型
24. 关注隐私的网络分析
25. Remote mob programming
26. 安全多方计算

暂缓
27. GitOps
28. 分层的平台团队
29. 天真的密码复杂度要求
30. 代码同行评审等同于 pull request
31. 规模化敏捷框架 (SAFe ™)
32. 代码与流水线的所有权分离
33. 工单驱动的平台运营模式

平台

采纳

试验
34. AWS 云开发工具包
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. 可变字体

评估
40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. 用 Kafka API 而非 Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

暂缓
50. Azure Machine Learning
51. 自研基础设施即代码 (IaC) 产品

工具

采纳
52. Sentry

试验
53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

评估
68. Buildah 和 Podman
69. GitHub Actions
70. Graal 原生镜像
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator 框架
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

暂缓
80. AWS CodePipeline

语言 & 框架

采纳
81. Combine
82. LeakCanary

试验
83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. 按需分发模块
92. Streamlit
93. SWR
94. TrustKit

评估
95. .NET 5
96. bUnit
97. Dagster
98. Flutter for Web
99. Jotai 和 Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 模块联邦

暂缓

TECHNOLOGY RADAR

技术

技术

API 扩张 - 收缩

采纳

API 扩张 - 收缩模式，有时也被称为并行修改，很多人对这个模式都很熟悉，尤其是在与数据库或代码一起使用的时候，而在 API 方面我们只看到较低的采用率。它具体指的是，人们通常会采用复杂的版本控制和破坏性的修改来实现 API 升级，而其实简单的扩张再收缩就可以满足需求。例如，在想要弃用 API 中的某个元素时，先添加一个不包含这个元素的 API，然后在消费者都切换到新的 API 之后，再删除该元素以及包含该元素的旧 API。这种方式确实需要 API 消费者的一些协调和可见性，可能需要通过消费者驱动的契约测试之类的技术来实现。

机器学习的持续交付（CD4ML）

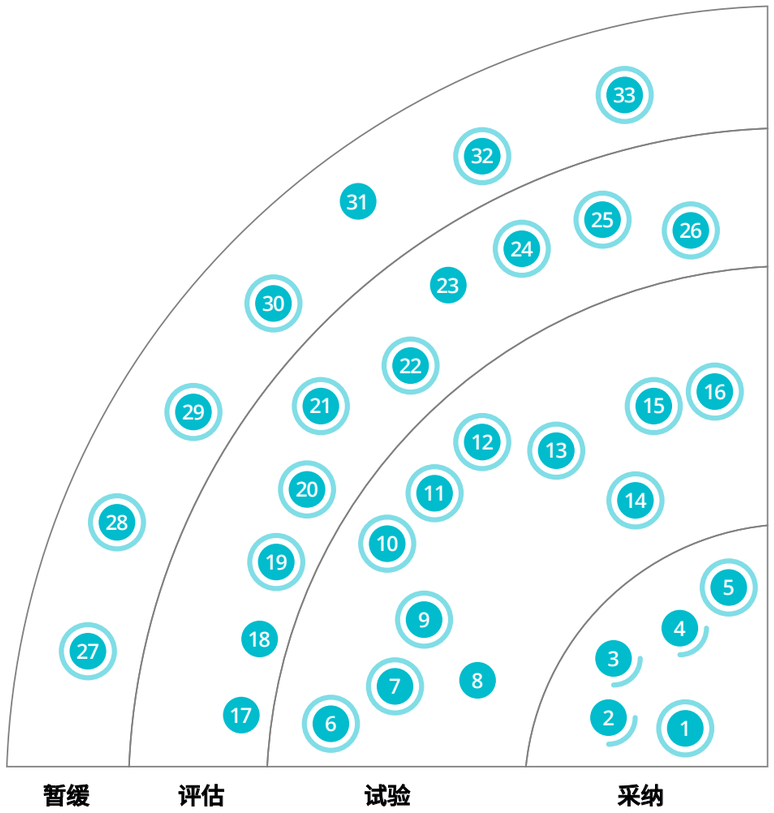
采纳

我们将机器学习的持续交付（CD4ML）作为所有部署在生产环境中的机器学习解决方案的默认起点。许多组织越来越依赖于机器学习解决方案来提供客户产品和内部运营，因此将持续交付（CD）所获得的经验教训和良好实践应用于机器学习解决方案具有良好的商业意义。

设计体系

采纳

随着应用程序开发变得越来越动态和复杂，以一致的风格交付可访问且可用的产品变成了一个挑战。在拥有多个团队从事不同产品开发的大型组织中，尤为如此。设计体系定义了一组设计模式，组件库，以及良好的设计和工程实践，以确保数字产品的一致性。在过去的公司风格指南基础上，设计体系提供了易于查找和使用的共享库和文档。通常指南是以代码的形式编写，并且受版本控制管理，所以相比简单的文档，它更明确且易于维护。设计体系已经成为跨团队和学科产品研发时的标准方法，因为它可以使团队更加专注于解决围绕产品本身的战略挑战，而无需在每次需要新的视觉组件时都重复造轮子。



平台工程产品团队

采纳

正如本期雷达主题之一所指出的那样，业界在创建和支持内部平台的“平台工程产品团队”中积累了越来越多的经验。组织中的团队使用这些平台，可以加速应用程序开发，降低运营复杂性并缩短产品上市时间。随着采用率的提高，我们对于这种方法的好和坏的模式也越来越清楚。创建平台时，至关重要的是要明确定义可以从中受益的客户和产品，而不是凭空建立。我们不仅要谨防分层平台团队，它保留了现有技术孤岛，只是贴上了“平台团队”的标签而已，还要小心工

采纳
1. API 扩张与收缩
2. 机器学习的持续交付（CD4ML）
3. 设计体系
4. 平台工程产品团队
5. 服务帐户轮换方法

试验
6. 云沙箱
7. Contextual bandits
8. Distroless Docker images
9. Ethical Explorer
10. 假设驱动的遗留系统改造
11. 轻量级的 RFCs 方法
12. 最简单的机器学习
13. 单页应用（SPA）注入
14. 团队的认知负担
15. 使用工具管理 Xcodeproj
16. UI/BFF 共享类型

评估
17. 限界低代码平台
18. 去中心化身份
19. 部署漂移提示器
20. 同态加密
21. Hotwire
22. 微前端中的模块映射
23. 开放应用程序模型
24. 关注隐私的网络分析
25. Remote mob programming
26. 安全多方计算

暂缓
27. GitOps
28. 分层的平台团队
29. 天真的密码复杂度要求
30. 代码同行评审等同于 pull request
31. 规模化敏捷框架（SAFe ™）
32. 代码与流水线的所有权分离
33. 工单驱动的平台运营模式

技术

该算法以赌场中的“老虎机”命名，通过探索不同的选择，学习有关预期结果的更多信息，并通过利用表现良好的选项来平衡该结果。

(Contextual bandits)

单驱动的平台运营模型。当考虑如何最好地组织平台团队时，我们仍然是团队拓扑概念的忠实拥护者。我们认为平台工程产品团队是一种标准方法，并且是高性能 IT 的重要推动者。

服务帐户轮换方法

采纳

我们强烈建议组织在确实需要使用云服务帐户的时侯确保这些云服务帐户的凭证能得到轮换。轮换是安全性的三个 R 之一。除非发生安全事件，否则组织很容易忘记这些帐户。这会导致拥有不必要的广泛权限的账户长期处于使用状态，并且缺少如何替换或轮换它们的计划。定期应用云“服务帐户轮换方法”还提供了运用最小权限原则的机会。

云沙箱

试验

由于云服务变得越来越常见，并且创建云沙箱变得更加容易且可大规模应用，我们的团队因此更倾向于使用完全基于云（相对本地而言）的开发环境，并以此来减少维护复杂度。我们发现用于本地模拟云原生服务的工具限制了开发者对构建和测试周期的信心，所以我们将重点放在标准化云沙箱上，而不是在开发机器上运行云原生组件。这样的方式可以使基础设施即代码实践得到更好的强制应用，并为开发人员提供沙箱环境良好的适应过程。当然这种转变也存在风险，因为它假定开发人员将完全依赖于云环境的可用性，并且可能会减慢开发者获得反馈的速度。我们强烈建议您采用一些精益治理的实践来管理这些沙箱环境的标准化，尤其是在安全、访问控制和区域部署方面。

Contextual bandits

试验

Contextual bandits 是一类非常适用于解决探索 / 利用权衡问题的强化学习算法。该算法以赌场中的“老虎机”命名，通过探索不同的选择，学习有关预期结果的更多信息，并通过利用表现良好的选项来平衡该结果。我们已经在一些场景中成功地使用了该技术，在这些场景中只使用了少量的数据来训练和部署一些机器学习模型。事实上，我们可以在此探索 / 利用的权衡过程中添加上下文，使它适合于各种用例，包括 A/B 测试、推荐和布局优化。

Distroless Docker images

试验

在为应用构建 Docker 镜像时，我们常常会考虑两件事：镜像的安全性和大小。通常情况下我们会使用容器安全扫描工具来检测和修复常见的漏洞和风险，并使用 Alpine Linux 之类的小型发行版来解决镜像过大和分发性能的问题。然而随着安全威胁的增长，消除所有潜在的攻击面变得空前重要，这就是 distroless Docker images 正成为部署容器默认选项的原因。Distroless Docker images 通过放弃完整的操作系统发行版来减少内存占用和依赖关系，它还可以减少安全扫描噪声和应用程序攻击面。此外，这一技术还意味着需要修补的漏洞更少了，镜像更小性能也更好了。Google 针对不同的语言发布了一套 distroless container images，你可以使用 Google 构建工具 Bazel 或者简单地使用多阶段 Dockerfiles 创建 distroless 的应用程序镜像。请注意，默认情况下，distroless 容器没有用于调试的 shell，不过，你可以在网上轻松地找到包含 BusyBox shell 的 distroless 容器调试版。Distroless Docker image 是 Google 开创的技术，根据我们的经验，这种技术的应用仍然主要局限于 Google 生成的镜像，如果有更多的供应商可供选择，我们会更放心。另外，在使用 Trivy 或类似的漏洞扫描器时要小心，因为只有较新版本才支持 distroless 容器。

Ethical Explorer

试验

Ethical OS 背后的组织——Omidyar 网络，是 eBay 创始人 Pierre Omidyar 发起的一个自述为致力于社会变革的冒险。Omidyar 网络最近新发布了一个名为 Ethical Explorer 的版本。新的 Ethical Explorer 在 Ethical OS 实践得来的经验教训的基础上，增加了更深入的问题供产品团队考虑。Ethical Explorer 可以在这里免费下载，并可折叠为卡片来引发讨论，这些卡片上有针对技术上的若干种“危险区”的一些开放性问题，包括监视（使用我们产品或服务的人是否可能追踪或识别其他用户信息?）、虚假信息、排异排外、算法歧视、沉迷上瘾、数据控制、安全攻击以及权限过大。其中的实战指南包括一些工作坊、开启对话的思路和获

得组织支持的技巧。为了让我们的行业更好地发挥数字社会当中的伦理外部性（ethical externalities）作用，我们仍然有很多工作要做，但我们已经使用 Ethical Explorer 开展了一些卓有成效的产品讨论，并且对产品决策在解决社会问题中的重要性的认识不断扩大也同样鼓舞了我们。

假设驱动的遗留系统改造

试验

我们经常被要求更新、升级或者修正那些原本不是由我们构建的遗留系统。有时，我们需要注意一些技术问题，比如提升性能和可靠性。解决这些问题的常用方法是使用与用户故事卡相同的格式创建“技术故事卡”，但以技术成果而非业务成果作为目标。但是这些技术任务通常很难估计需要的时间，花费的时间比预期的要长，最终也时常会得不到预想的成果。另一种更成功的方法是应用假设驱动的遗留系统改造。不同于面向标准的 backlog 工作，在这一方法下，团队提出可度量的预期技术成果，并共同建立一组对问题的假设。然后，团队根据优先级在有限时长内进行迭代试验，来验证或推翻每个假设。由此将产生优化后的工作流程，这一流程不是为了按照计划朝着可预测的结果前进，而是为了减少不确定性。

轻量级的 RFCs 方法

试验

随着组织朝着演进式架构的方向发展，记录下围绕设计、架构、技术和团队工作方式的决策是非常重要的。收集和汇总那些会导致这些决策的反馈的过程从 RFCs（Request for Comments）开始。RFCs 是一种用于收集上下文、设计和架构思想，并与团队协作，最终达成决策以及上下文和结果的技术。我们建议组织采用一种轻量级的 RFCs 方法，通过在多个团队中使用简单的标准化模板和版本控制来获取 RFCs。

当未来的团队成员回过头来重新审视这些决策时，掌握这些信息将使他们受益无穷，与此同时记录组织的技术和业务的发展也十分重要。成熟的组织已经在自治团队中，特别是在跨团队相关的决策中使用 RFCs 来推动更好的沟通和协作。

最简单的机器学习

试验

所有主流的云厂商都提供了令人眼花缭乱的机器学习解决方案。这些强大的工具虽然可以为用户提供很多价值，但同时也意味着一定的开销：云厂商收取的除了基本的服务运行成本外，还包含一些营业税。这些复杂的工具需要被使用者理解和操作，随着架构中新工具的添加，税也会随之水涨船高。根据我们的经验，很多团队之所以会选择复杂的工具，常常是因为他们低估了诸如线性回归之类的简单工具的力量。其实，许多机器学习问题并不需要 GPU 或神经网络。因此，我们提倡在现有的计算平台上使用简单的工具和模型，以及少量的 Python 代码来实现一个最简单的机器学习。只有在能够证明它的必要性时，才使用这些复杂的工具。

单页应用（SPA）注入

试验

绞杀榕模式常被用作遗留系统现代化改造的默认策略，这种策略下，新代码包围在旧代码周围，并慢慢地实现旧代码的能力来满足所需功能。这种“由外而内”的方式对于很多遗留系统非常有用，但现在我们有了更多单页应用（SPA）的经验，可以用它们来替换遗留系统，我们正使用相反的“从内到外“的方式来替换遗留系统。与包围遗留系统不同，我们向旧应用 HTML 文档嵌入新的单页应用，并基于此逐渐扩展功能。只要用户能够容忍由于页面大小增加而带来的性能问题，新单页应用的框架就不需要与原有框架保持一致（例如，向原有的 AngularJS 应用嵌入新的 React 应用）。单页应用注入允许你逐渐迭代掉旧的单页应用，直到它被新应用完全替代掉。与绞杀榕模式通过在已有稳定代码的基础上构建新代码并最终替换旧代码相比，这种方式更接近于向已有应用注入一个外部的代理并依赖原有单页应用的功能，直到新应用能够完全接管。

团队的认知负担

试验

系统的架构反映了组织架构和沟通机制。我们应当有意识地关注团队如何互动，这并不是什么大新闻，正如康威逆定律（Inverse Conway Maneuver）所描述的那样，团队的互动是影响团队向客户交付价值的速度和容易程度的重要因素。我们很高兴找到一种方法来衡量这些互动。我们使用组织架构和

技术

当涉及到单页应用（SPAs）的遗留系统现代化时，我们没有包围遗留系统，而是向旧应用 HTML 文档嵌入新的单页应用，并基于此逐渐扩展功能。

（单页应用（SPA）注入）

技术

团队的互动是影响团队向客户交付价值的速度和容易程度的重要因素。组织架构和高效团队（Team Topologies）的作者提供的评估方法可以帮助衡量团队的互动，即我们称之为的团队认知负担。

（团队的认知负担）

高效团队（Team Topologies）的作者提供的评估方法。这种评估方法可以让你理解团队构建、测试和维护其服务的难易程度。通过衡量团队的认知负担，我们能够为客户提供更好的建议，帮助他们改变团队架构，改进团队的互动方式。

使用工具管理 Xcodeproj

试验

许多使用 Xcode 编写 iOS 代码的开发者经常会因为 Xcodeproj 文件总是随着项目配置变化而变化而头痛不已。由于 Xcodeproj 文件格式难以阅读，因此合并文件冲突非常复杂，有可能导致生产力下降甚至打乱整个项目，因为这个文件一旦出现错误，Xcode 将无法正常运行，开发工作也很可能被迫中断。因此，我们建议不要手动合并修复 Xcodeproj 文件，或是对其进行版本管理，而是使用工具管理 Xcodeproj：用 YAML（XcodeGen，Struct），Ruby（Xcake）或是 Swift（Tuist）定义你的 Xcode 项目配置。这些工具会根据配置文件和项目结构生成 Xcodeproj 文件，因此，Xcodeproj 文件合并冲突将成为历史，相比之下配置文件冲突要容易处理得多。

UI/BFF 共享类型

试验

随着 TypeScript 成为前端开发的常用语言以及 Node.js 成为 BFF 的首选技术，我们看到“UI/BFF 共享类型”正在被越来越多地使用。在这种技术中，一个类型定义不仅会被使用在前端请求返回的数据中，也会被使用在服务端用来满足这些查询。由于这种做法会跨越流程边界产生不必要的紧密耦合，因此我们通常会对这种做法保持谨慎。但是，许多团队发现这种方法的好处胜过耦合带来的风险。当一个团队同时拥有 UI 和 BFF 代码时，通常会将组件存在同一个代码库中，这个时候 BFF 模式是最有效的，因此 UI / BFF 对可以被看作是一个单一的内聚系统。当 BFF 提供强类型的查询时，可以针对前端的特定需求量身定制查询结果，而不必重用一个通用的实体，因为这些通用实体需要为很多消费者服务因此会包含很多不必要的字段。这样可以减少意外地将不必要的数据暴露给用户的风险，防止对返回的数据对象进行错误的解释，并使查询更加表意。当使用 io-ts 来增强运行时类型安全性时，这种实践特别有用。

限界低代码平台

评估

现在很多公司正在面临的一个最微妙的决定便是是否要采纳低代码平台或无代码平台，这些平台可以被用来在非常特定的领域里解决一些特定的问题。限界低代码平台这一领域的供应商也有如过江之鲫。现在看来，这类平台的一个突出的问题，便是很难应用一些诸如版本控制之类的优秀的工程实践。而且这类平台上的测试也非常的困难。然而我们还是注意到了这个市场里的一些有趣的新兵，例如 Amazon Honeycode 可以被用来创建一些简单的任务和事件管理应用，还有 IFTTT（类似于云工作流）领域的 Parabola，这也是为何我们会将限界低代码平台纳入这个部分的原因。但是我们仍然对它们更广泛的适用性深表怀疑，因为这些工具，如日本 Knotweed，非常容易超出它们原本的限界而被泛化用于其他场景，这也是为什么我们对采纳这种技术持强烈的谨慎态度。

去中心化身份

评估

SSL/TLS 的核心贡献者 Christopher Allen 在 2016 年给我们介绍了一种用于支撑新型数字化身份的 10 个原则，以及实现这一目标的途径：通往自主身份之路。自主身份也被称为去中心化身份，按照基于 IP 协议栈的信任标准，是一种“不依赖任何中心化权威并且永远不能被剥夺的任何人、组织或事物的终身可转移身份”。采纳和实现去中心化身份正在逐渐升温并变得可能。我们看到了它在隐私方面的应用：客户健康应用、政府医疗基础设施和公司法律身份。如果想快速地应用去中心化身份，你可以评估 Sovrin Network，Hyperledger Aries 和 Indy 等开源软件，以及去中心化身份和可验证凭证标准。我们正在密切关注这个领域，并帮助我们的客户在数字信任的新时代进行战略定位。

部署漂移提示器

评估

部署漂移提示器使得部署在多个环境中的软件版本漂移能够被可视化。使用了自动部署方式的组织在将软件部署到接近生产环境的环境中时，可能需要人工批准，这就意味着这些环境里的代码版本可能比当前的开发版本落后好几个版本。这项技术使得这些延后能够被展示在一个简单的面板内，包括在每个环境当中，每个被部署的组件有多大程度的延后。这能够帮助突出由于已经完成的软件没有部署到生产环境而导致的机会成本，并使得团队注意相关的风险，例如尚未部署的安全修复。

同态加密

评估

完全的同态加密（HE）是指一类允许在加密数据上直接进行计算操作（如搜索和算数运算）的加密方法。同时计算的结果仍然以加密的形式存在，并且稍后可以对其进行解密和显示。虽然同态加密问题早在 1978 年就被提出来了，但直到 2009 年才出现解决方案。随着计算机算力的提升，和诸如 SEAL，Lattigo，HElib 和 Python 中的部分同态加密之类易于使用的开源库的出现，同态加密在现实世界的应用程序中的应用才真正地变得可行。那些令人振奋的应用场景包括在将计算外包给一个不受信的第三方时的隐私保护，例如在云端对加密数据进行计算，或使第三方能够聚合同态加密后的联邦机器学习的中间结果。此外，大多数的同态加密方案被认为是对量子计算机安全的，并且标准化同态加密的努力也正在进行之中。尽管同态加密目前在性能和可支持的计算类型上还存在诸多局限，但是它仍然是一个值得引起我们注意的技术。

Hotwire

评估

Hotwire（跨端传送 HTML）是一项构建网页应用的技术。页面由组件组成，然而与现代单页应用不同，这些组件的 HTML 在服务端生成并“跨端”传送至浏览器。这样的应用只在浏览器端运行少量的 JavaScript 代码用以将 HTML 片段组合在一起。包括我们在内的许多团队在异步 web 请求获得跨浏览器支持的 2005 年前后都尝试了这项技术，然而由于各种原因，这项技术并未引起很多注意。

如今，Hotwire 使用现代网页浏览器和 HTTP 的能力来实现单页应用（SPA）的快速、自适应和动态特性。它通过将逻辑放在服务端并保持客户端代码的简洁，实现了更简单的网页应用设计。Basecamp 团队发布了一些 Hotwire 框架来支持他们自己的应用程序，其中包括 Turbo 和 Stimulus。Turbo 包含了提升应用程序响应速度的一系列技术和框架，例如防止整个页面重新加载、从缓存预览页面以及根据请求将页面切片等。Stimulus 旨在通过将 JavaScript 对象与 HTML 页面元素关联起来，以增强浏览器当中的静态 HTML。

微前端中的模块映射

评估

当使用多个微前端来组建应用程序时，系统的某些部分需要确定加载哪些微前端以及从哪里加载它们。一直以来，我们要么通过构建定制化解决方案，要么采用 single-spa 之类的热门框架来解决这个问题。最近出现了一个新标准模块映射，它对这两种方案都有所助益。我们的初步经验表明，使用微前端中的模块映射可以清晰地分离关注点。它使用 JavaScript 代码说明要导入的内容，同时在初始化 HTML 的响应中使用一个轻量脚本指定从哪里加载微前端。显然，HTML 是在服务器端生成的，这样在渲染过程中就可以使用一些动态配置。在许多方面，这种技术让我们想起了动态 Unix 库的 linker/loader 目录。目前，只有 Chrome 浏览器支持模块映射，但是通过 SystemJS polyfill 就可以使它得到更加广泛的应用。

开放应用程序模型

评估

开放应用程序模型（OAM）旨在为“基础设施即软件”制定标准化方案。利用组件、应用程序配置、范围和特征等抽象，开发人员能以与平台无关的方式描述其应用程序。而平台实现者则完全可以用工作负载、特征和范围等另一套抽象来定义其平台。自从上次提到 OAM 以来，我们一直对其首个实现 KubeVela 保持着关注。如今 KubeVela 即将发布 1.0 版，我们期待着它能证明 OAM 构想中的前景。

技术

使用了自动部署方式的组织在将软件部署到接近生产环境的环境中时，可能需要人工批准，这就意味着这些环境里的代码版本可能比当前的开发版本落后好几个版本。部署漂移提示器使得这些延后能够被展示在一个简单的面板内。

（部署漂移提示器）

技术

安全多方计算（MPC）解决了多方隐私保护的协作计算中，各方互相不信任的问题。其目的是在无可信第三方的情况下，安全地计算出一个商定的问题。

（安全多方计算）

关注隐私的网络分析

评估

关注隐私的网络分析是一种收集网络分析的技术，它通过对终端用户匿名化的处理而防止泄漏其隐私信息。遵守通用数据保护条例（GDPR）的一个令人惊讶的结果是，许多组织不惜降低用户体验地使用复杂的 cookie 同意过程，尤其是在用户没有立即同意“所有 cookies”的默认设置的情况下。关注隐私的网络分析具有双重优势，无论是形式还是实际它都遵守了 GDPR 条例，与此同时也避免了引入具有侵入性的 Cookie 同意书。这项技术的一个实现便是 Plausible。

Remote mob programming

评估

Mob 编程是我们团队发现的一项能使远程工作变得更加方便且容易的技术。远程 mob 编程允许团队成员快速“蜂拥”在一个问题或者一小段代码周围，不用受物理限制而需要找到一个可以容纳多人的房间。团队可以在一个问题或者一段代码进行快速地协作，而无需连接到一个大型显示器，预定会议室或者找一个白板。

安全多方计算

评估

安全多方计算 （MPC）解决了多方隐私保护的协作计算中，各方互相不信任的问题。其目的是在无可信第三方的情况下，安全地计算出一个商定的问题。计算时，每个参与者都必须参与结果计算，并且不能被其他参与者获得计算结果。安全多方计算的一个简单例子是百万富翁问题：两个百万富翁想了解谁是最富有的人，但又不想透露给彼此他们的实际净资产，同时他们也都不信任第三方。安全多方计算的实施方法各不相同，可能包括秘密共享，遗忘传输，混淆电路或同态加密。最近出现的一些商业安全多方计算解决方案（例如 Antchain Morse）声称有助于解决诸如多方联合信用调查和病历数据交换等情形下，秘密共享和安全机器学习的问题。尽管从市场营销的角度来看，这些平台很有吸引力，但它们是否真的有用仍拭目以待。

GitOps

暂缓

我们建议在采用 GitOps 时一定要谨慎，尤其是在分支策略方面。GitOps 可以被视作一种基础设施即代码的实现方式，这种方式持续地从 Git 同步基础设施代码，并将其部署到多个环境当中。在为每个环境创建一个分支的场景下，对基础设施的修改会通过合并代码的方式从一个环境应用到下一个环境。诚然，将代码作为基础设施修改的唯一来源，是一种合理的方式，但我们发现对每个环境创建分支会导致环境之间的不一致。最终，合并特定环境的配置代码带来了许多问题，甚至导致这种方式被废弃。这与我们之前在 Gitflow 的长期分支当中看到的情景非常相似。

分层的平台团队

暂缓

软件平台的流行为组织创造了很多价值，但是在构建基于平台的交付模型的道路上到处都是潜在的死胡同。在这些新流行技术的刺激下，我们往往会发现它们实际上都是“新瓶装旧酒”，是“老技术”在新时代背景下的另一种“复兴”，这很容易使我们忽略一开始选择放弃这些技术的原因。我们在上一期的技术雷达中发表的披着 API 网关外衣的企业服务总线就是一个很好的例子。我们看到的另一个例子是“按技术层级划分团队”，只不过换了个说法将其称为平台。在构建应用程序的上下文中，前端团队，业务逻辑团队和数据团队分开是很常见的，而在组织根据平台能力划分业务或数据层团队时我们看到了与之相似的模型结构。由于康威定律，我们知道围绕业务能力组织平台功能团队是一种更有效的方式，它为团队提供能力的端到端的所有权，包括数据所有权。这有助于避免分层的平台团队的依赖管理问题，否则做任何事情的时候前端团队都必须依赖业务逻辑团队，而业务逻辑团队又必须依赖数据团队。

天真的密码复杂度要求

暂缓

密码策略是当前很多组织会默认启用的标准。然而，我们仍然见到很多组织内部要求密码必须包含符号、数字、大小写字母和特

殊字符。诸如这样的要求就是天真的密码复杂度要求。这些要求会导致错误的安全意识，因为用户会由于满足这些要求的密码太难以记忆和输入，而选择使用更不安全的密码。正如 NIST（美国国家标准技术研究所）推荐所提到的，影响密码强度的主要因素是密码的长度，因此用户应该选择更长的密码，最长为 64 个字符（包括空格）。这些密码会更安全，并且更易于记忆。

代码同行评审等同于 pull request

暂缓

一些组织似乎觉得代码同行评审等同于 pull request。他们认为唯一能够实现代码评审的方法就是 pull request。我们发现这种方法会造成严重的团队瓶颈，并且显著地降低反馈的质量，因为超负荷的评审人员会开始简单地拒绝 pull requests。虽然有观点认为，这是一种展示代码评审强制性的方式，但是我们的一个客户被告知，这种观点没有任何的依据，因为没有证据能够表明代码在被接受之前真的有被阅读过。Pull requests 只是管理代码评审工作流的一种方式，我们敦促人们考虑其它的方法，特别是在需要对代码提供指导和反馈的情况下。

规模化敏捷框架（SAFe ™）

暂缓

我们对“做敏捷之前先得变得敏捷”的定位以及我们对该主题的观点应该不会让人感到意外。根据 Gartner 2019 年 5 月的报告来看，规模化敏捷框架®（Scaled Agile Framework®）是被考虑和使用得最多的企业敏捷框架，我们也看到越来越多的企业正在经历组织变革，因此我们认为是时候再次提高对该主题的认知了。我们遇到过一些组织在过度标准化的规模化敏捷框架（SAFe）和阶段化流程中苦苦挣扎。这些流程在组织架构及其运作模式中造成摩擦。它还会导致组织中形成孤岛，阻碍平台成为真正的业务能力推动者。自上而下的管控会在价值流中产生浪费，阻碍工程人才的创造力，同时限制团队的自主性和试验。相较于衡量工作量并关注标准化的仪式，我们更加推荐采用一种更精益的，以价值驱动的方法和治理来帮助组织减少摩擦，例如价值驱动的数字化转型或者团队认知负荷评估，以识别团队的类型并确定他们应该如何更好地互动。

Scaled Agile Framework® 和 SAFe ™ 均为 Scaled Agile 公司的商标。

代码与流水线的所有权分离

暂缓

理想情况下，尤其是当团队实践 DevOps 时，流水线和被部署的代码应该由同一个团队所有。遗憾的是，我们仍然可以看到代码与流水线的所有权分离的团队：他们的流水线由基础设施团队所有，这会导致变更延迟、改进受阻以及开发团队缺乏对部署的所有权和参与度。造成这种情况的一个原因显然是团队分离，另一个原因可能是想要保留审核的流程和“守门员”的角色。尽管使用这种方式（例如：监管控制）可能有比较合理的解释，但总的来说，我们发现它令人不悦并且收效甚微。

工单驱动的平台运营模式

暂缓

平台化的终极目标之一便是将基于工单的流程降到最低，因为这些流程会在价值流中形成队列。遗憾的是，我们仍然看到一些组织在这一目标的推进上做得不够有力，并因此导致了工单驱动的平台运营模式。尤其是当这些平台化服务是构建于一些自服务的和 API 驱动的公有云服务之上时，这一情况便格外令人沮丧。虽然从一开始就实现自服务且不依赖于工单是很困难且没有必要的，但必须明确的是，这应该是我们努力的目标。

对官僚主义的过度依赖和信任的缺失是组织在面对这种基于工单的流程却不愿做出改变的原因之一。在平台中加入更多的自动化校验和告警是帮助我们从工单审批流程中抽身的一种方法。例如给团队提供运行成本的可见性，并设置自动化的成本护栏以避免意外的成本激增。通过实现安全策略即代码，使用配置扫描器或类似 Recommender 的分析工具，来帮助团队做正确的事。

技术

一些组织似乎觉得代码同行评审等同于 pull request。我们发现这种方法会造成严重的团队瓶颈，并且显著地降低反馈的质量。

（代码同行评审等同于 pull request）

TECHNOLOGY RADAR

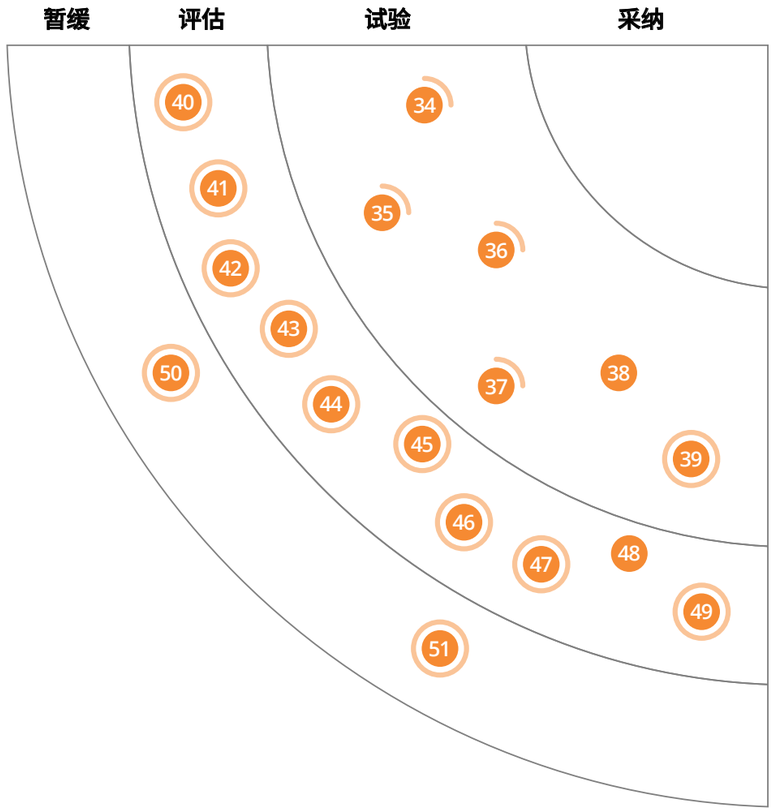
平台

平台

AWS 云开发工具包

试验

我们许多使用 AWS 的团队中发现，AWS 云开发工具包是一个合理的 AWS 默认工具，以实现基础设施的整备工作。其特别之处在于，他们喜欢使用主流编程语言而不是配置文件来进行开发，从而可以使用现有的工具、测试方法和技能。但与类似的工具一样，此时也仍需要谨慎地确保部署易于理解和维护。这个开发工具包目前支持 TypeScript、JavaScript、Python、Java、C# 和 .NET。新语言的支持正在被添加到 CDK 中。此外，我们使用了 AWS 云开发工具包和 HashiCorp 的 Terraform 云开发工具包来生成 Terraform 配置，并成功实现了与 Terraform 平台的整备。



采纳

试验
34. AWS 云开发工具包
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. 可变字体

评估
40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. 用 Kafka API 而非 Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

暂缓
50. Azure Machine Learning
51. 自研基础设施即代码（IaC）产品

Backstage

试验

随着组织在寻求支持和简化其开发环境时，开始采用开发者门户，我们看到人们对 Backstage 的兴趣和使用量在不断增长。随着工具和技术数量的增加，采用某种形式的标准化，对于保持开发的一致性变得越来越重要。因为一旦实现了一致性，开发人员就可以专注于创新和产品开发，而不是陷入重复发明轮子的泥淖。Backstage 是由 Spotify 创建的开源开发者门户平台。它由软件模板、统一的基础设施工具和一致且集中的技术文档所构成。插件式架构为组织的基础设施生态系统，提供了可扩展性和适应性。

Delta Lake

试验

Delta Lake 是由 Databricks 实现的开源存储层，旨在将 ACID 事务处理引入到大数据处理中。在使用了 Databricks 的 data lake 或 data mesh 的项目中，我们的团队更喜欢使用 Delta lake 存储，而不是直接使用 S3 或 ADLS 等文件存储类型。当然，这仅限于那些使用了支持 Delta Lake 的存储平台的项目，并且使用了 Parquet 文件格式。当需要实现文件级事务完整性时，Delta Lake 能实现并发数据读写。尽管还存在一些限制，但 Delta Lake 与 Apache Spark batch 以及 micro-batch 接口的无缝集成，对我们非常有用。尤其有用的是诸如时间旅行（在特定时间点访问数据或进行提交回滚）和对写操作的 schema 演进的支持。

平台

LinkedIn 已经将 Where-Hows 进化为 DataHub，一个通过可扩展的元数据系统实现数据可发现性的下一代平台。

(DataHub)

Materialize

试验

Materialize 是一种流数据库。在无需复杂的数据管道的情况下，只须用标准 SQL 视图描述计算，然后将 Materialize 连接到数据流，就能实现增量计算。底层的差分数据流引擎能够运行增量计算，从而以最小的延迟，提供一致且准确的结果。与传统数据库不同，定义 Materialize 视图不存在任何限制，并且计算可以实时执行。我们将 Materialize 与 Spring Cloud Stream 以及 Kafka 配合使用，从而在分布式事件驱动的系统中，查询事件流并分析结果。其效果令人满意。

Snowflake

试验

自从上次在雷达上提到 Snowflake 以来，对于它的使用，以及作为数据仓库和数据湖的替代方案的 data mesh，我们都获得了更多经验。Snowflake 在时间旅行、零拷贝克隆、数据共享及其应用市场等功能方面，继续给人留下深刻印象。Snowflake 至今还没出现任何令我们不喜欢的地方，所以相较于其他选择来说，我们的顾问会更偏爱使用它。亚马逊的数据仓库产品 Redshift 正在朝着将存储和计算进行分离的方向发展，而这一直都是 Snowflake 的强项。如果使用 Redshift 产品中的 Spectrum 特性进行数据分析，就会感觉它并非那么方便和灵活，部分原因是它受到了 Postgres 的约束（虽然我们也喜欢用 Postgres。而进行联合查询 (federated queries) 可能是使用 Redshift 的原因。在操作方面，Snowflake 的操作会更简单。虽然 BigQuery 是另一种选择，且非常易于操作，但在多云的场景下，Snowflake 是更好的选择。我们已经在 GCP、AWS 和 Azure 上成功地使用了 Snowflake。

可变字体

试验

可变字体能避免查找和引用多个具有不同字重和样式的字体文件。这种方法使得所有变体样式都保存在一个字体文件中，并且可以使用属性来选择所需的样式和字重。尽管这不是新技术，但我们仍然看到网站和项目因使用这种简单方法而受益。如果页面中包含相同字体的不同变体，建议尝试使用可变字体。

Apache Pinot

评估

Apache Pinot 是分布式 OLAP 数据存储系统，旨在提供低延迟的实时分析。它可以从批处理数据源（例如 Hadoop HDFS、Amazon S3、Azure ADLS 或 Google Cloud Storage）以及流式数据源（例如 Apache Kafka）中提取数据。如果需要进行面向用户的低延迟数据分析，则 Hadoop SQL 方案不能保证所需的低延迟。诸如 Apache Pinot（或 Apache Druid 和 Clickhouse 等）现代 OLAP 引擎，可以实现低得多的延迟，特别适合针对不可变数据（如聚合）进行快速分析的场景（或许需要进行实时数据提取）。Apache Pinot 最初由 LinkedIn 构建，并于 2018 年底进入 Apache 孵化阶段。此后，该系统又增加了插件架构和 SQL 支持等关键特性。Apache Pinot 的操作相当复杂，并且具有许多需要控制的部件，但是如果需要分析的数据量足够大，并且对查询要求低延迟，则建议评估一下 Apache Pinot。

Bit.dev

评估

Bit.dev 是一个云托管平台，用于通过 Bit 抽取、模块化并重用 UI 组件。虽然 Web components 已经存在一段时间了，但通过组装从其它项目抽取出来的独立小组件，来构建现代前端应用，仍然很困难。而 Bit 要解决的，就是这个问题。它是一个简洁的命令行工具，可以从已有的库或项目当中抽取组件。要进行组件协作，既可以在 Bit 的基础上自行构建相关服务，也可以直接使用 Bit.dev。

DataHub

评估

自从我们第一次在技术雷达中提及 data discoverability 以来，LinkedIn 已经将 WhereHows 进化为 DataHub，一个通过可扩展的元数据系统实现数据可发现性的下一代平台。与爬取和拉取元数据不同，DataHub 采用了基于推送的模式。数据生态系统中各个组件，都可以通过 API 或者流 (stream) 向中心化的平台上发布元数据。

这种基于推送的数据集成，将数据发现所有权从中心实体转移到各个团队，使他们对自己的元数据负责。随着越来越多的公司试图成为数据驱动型企业，拥有一个有助于数据发现和理解数据质量与渊源的系统，是至关重要的。我们建议评估 DataHub 在这方面的能力。

Feature Store

评估

Feature Store 是一个服务于机器学习的数据平台，可以解决当前我们在特征工程中所遇到的一些关键问题。它提供了三个基本功能：(1) 使用托管的数据管道，以消除新数据与数据管道之间的冲突；(2) 对特征数据进行编目和存储，从而促进跨模型的特征的可发现性和协同性；(3) 在模型的训练和干扰过程中，持续提供特征数据。

自从 Uber 公开了 Michelangelo 平台以来，许多组织和初创企业都建立了自己的特征库； 例 如 Hopsworks、Feast 和 Tecton。我们看到了 Feature Store 的潜力，并建议仔细对其进行评估。

JuiceFS

评估

JuiceFS 是基于 Redis 和对象存储而构建的开源分布式 POSIX 文件系统。如果要构建新的应用程序，那么我们建议始终与对象存储进行直接交互，而无需经过另一个抽象层。另外，如果要将依赖于传统 POSIX 文件系统的遗留系统迁移到云上，也可以考虑使用 JuiceFS。

用 Kafka API 而非 Kafka

评估

随着越来越多的企业开始运用事件在微服务之间共享数据、收集分析数据或传输数据到数据湖，Apache Kafka 已经成为支撑事件驱动架构的最受欢迎的平台。尽管 Kafka 的可伸缩的消息持久化概念是革命性的，但要使其正常工作，还是需要依赖众多的活动部件，包括 Zookeeper、代理、分区和镜像。虽然实现和维护这些组件会很棘手，但是它们确实在需要的时候，尤其是在企业规模的应用中，提供了极大的灵活性和强大功能。因为采用 Kafka 完整生态系统的门槛较高，所以我们乐于见到一些平台在最近的爆发式增长。这些平台提供用 Kafka API 而非 Kafka 的功能。最近涌现出的 Kafka on Pulsar 和 Redpanda，就是属于这类平台。而 Azure Event Hubs for Kafka 则提供了对 Kafka 生产者和消费者 API 的兼容。但由于 Kafka 的某些功能（例如数据流客户端库）与这些替代代理不兼容，因此仍然有理由选择 Kafka 而不是这些替代代理。然而究竟开发者是否会采用“用 Kafka API 而非 Kafka”的策略，抑或这只是 Kafka 的竞争对手试图将用户引诱到 Kafka 平台之外，还有待观察。最终，也许 Kafka 最持久的影响力，就是其提供给客户的易用协议和 API。

NATS

评估

NATS 是一种快速和安全的消息队列系统，具有异常广泛的功能和潜在的市场。有人会问，为什么还需要另一个消息队列系统?自从企业开始使用计算机以来，各种形式的消息队列系统已经存在了很长时间了，并且针对各种任务，经历了多年的改进和优化。但是，NATS 的确具有几个有趣的特征，并且其独特的伸缩性，既能用于嵌入式控制器，又能用于全球范围云托管的超级集群。NATS 旨在支持来自移动设备或 IoT 并通过互连系统的网络所传递的连续数据流，对此我们特别感兴趣。但是，该系统也需要解决一些棘手的问题，其中最重要的，是确保消费者仅看到他们被允许访问的消息和主题，尤其是当网络跨越组织边界时。NATS 2.0 引入了一个安全和访问控制框架。该框架支持多租户群集。在该群集中，帐户限制了用户对消息队列和主题的访问。NATS 是 Go 语言编写的，最初主要被 Go 语言社区所接受。尽管该系统针对几乎所有广泛使用的编程语言都提供客户端，但是 Go 语言所实现的客户端是迄今为止最受欢迎的。我们的一些开发人员发现，所有编程语言的客户端库，都倾向于要提供 Go 语言客户端所具备的特性。小型无线设备的带宽和处理能力的提高，意味着企业必须实时处理的数据量只会增大。可以评估 NATS 作为在企业内部和企业之间，以流的形式传输数据的可行性。

平台

随着越来越多的企业开始运用事件在微服务之间共享数据、收集分析数据或传输数据到数据湖，Apache Kafka 已经成为支撑事件驱动架构的最受欢迎的平台。因为采用 Kafka 完整生态系统的门槛较高，所以我们乐于见到一些平台在最近的爆发式增长。这些平台提供用 Kafka API 而非 Kafka 的功能。

(用 Kafka API 而非 Kafka)

平台

有时组织会倾向于在现有的外部产品之上，构建框架或抽象，来满足组织内非常特定的需求，并认为这种适配会比其现有的外部产品具备更多好处。然而他们低估了根据其需求不断演进这些自研解决方案而所需投入的工作量。很快他们就会意识到，所基于的外部产品的原始版本要比他们自己的产品好得多。

（自研基础设施即代码（IaC）产品）

Opstrace

评估

Opstrace 是一个用于实现系统可观测性的开源平台，旨在部署于用户自己的网络中。如果不使用像 Datadog 这样的商业解决方案（例如，由于成本或数据驻留地点的考虑），那么唯一的解决方案就是构建由开源工具组成的自己的平台。这需要投入很大的工作量，而 Opstrace 就是来解决这个问题的。它使用开源 api 和接口，如 Prometheus 和 Grafana，并在上面添加了额外的特性，如 TLS 和身份验证。Opstrace 的核心运行了一个 Cortex 集群，提供可伸缩的 Prometheus API 和 Loki 日志集群。与 Datadog 或 SignalFX 等解决方案相比，它是崭新的平台，所以仍然缺少一些特性。尽管如此，它所解决的上述问题，使其在该领域仍然很有前景，值得关注。

Pulumi

评估

我们已经看到人们对 Pulumi 的兴趣正在缓慢且稳步地上升。虽然 Terraform 在基础设施编程世界中地位稳固，但 Pulumi 却填补了其中的一个空白。尽管 Terraform 是一个久经考验的常备选项，但其声明式编程特质，深受抽象机制不足和可测试性有限的困扰。如果基础设施完全是静态的，那么 Terraform 就够用了。但是动态基础设施但定义，要求使用真正的编程语言。Pulumi 允许以 TypeScript/ JavaScript、Python 和 Go 语言（无需标记语言或模板）编写配置信息，这使其脱颖而出。Pulumi 专注于原生云架构，包括容器、无服务器函数和数据服务，并为 Kubernetes 提供了良好的支持。最近，AWS CDK 的推出对其形成了挑战，但 Pulumi 仍然是该领域唯一的能独立于任何云平台厂商的工具。我们期望将来人们能更广泛地采用 Pulumi，并期待出现能对其提供支持的可行的工具和知识生态系统。

Redpanda

评估

Redpanda 是一个数据流处理平台。由于具备与 Kafka 兼容的 API，使得开发人员不必像 Kafka 那样进行复杂的安装，就能从 Kafka 生态系统中受益。例如，Redpanda 可以通过下述方法来简化操作，即仅发布一个二进制文件就可进行安装，同时无须诸如 ZooKeeper 这样的外部依赖。为了做到这一点，它实现了 Raft 协议，并进行了全面的测试，以验证其对该协议的实现是正确的。Redpanda 的一项功能（仅对企业客户可用）是使用嵌入式 WASM 引擎，来实现 WebAssembly（WASM）的内联转换。这允许开发人员使用他们所选择的编程语言，并将其编译为 WASM，来创建事件转换器。在完成了一系列优化之后，Redpanda 还大大减少了尾部延迟，并提高了吞吐量。Redpanda 是一个令人兴奋的 Kafka 的替代品，值得评估。

Azure Machine Learning

暂缓

我们之前已经观察到，云供应商会将越来越多的服务推向市场。同时雷达还记录了我们担忧的一些问题，即有些服务还没有完全准备就绪，就被推向市场供人使用。不幸的是，以我们的经验来说，Azure Machine Learning 就属于这类服务。作为限界低代码平台领域的新产品，Azure ML 承诺为数据科学家提供更多的便利。但是，最终它没有实现诺言。实际上，我们的数据科学家仍然认为，使用 Python 会更为容易。尽管我们付出了巨大的努力，但仍难以使其规模化。此外缺少足够的文档也是它的另一个问题。所以我们将其移动至暂缓环。

自研基础设施即代码（IaC）产品

暂缓

那些由公司或社区所支持的（至少在业界引起关注的）产品，正在不断发展。但有时组织会倾向于在现有的外部产品之上，构建框架或抽象，来满足组织内非常特定的需求，并认为这种适配会比其现有的外部产品具备更多好处。我们发现一些组织试图基于现有外部产品，创建自研基础设施即代码（IaC）产品。他们低估了根据其需求不断演进这些自研解决方案而所需投入的工作量。很快他们就会意识到，所基于的外部产品的原始版本要比他们自己的产品好得多。在有些情况下，构建在外部产品之上的抽象，甚至削弱了外部产品的原始功能。虽然目睹过组织自研产品的一些成功案例，但是我们仍然建议要审慎地考虑这种方式。因为这会带来不可忽视的工作量，并且需要确立长期的产品愿景，才能达到预期的结果。

TECHNOLOGY RADAR

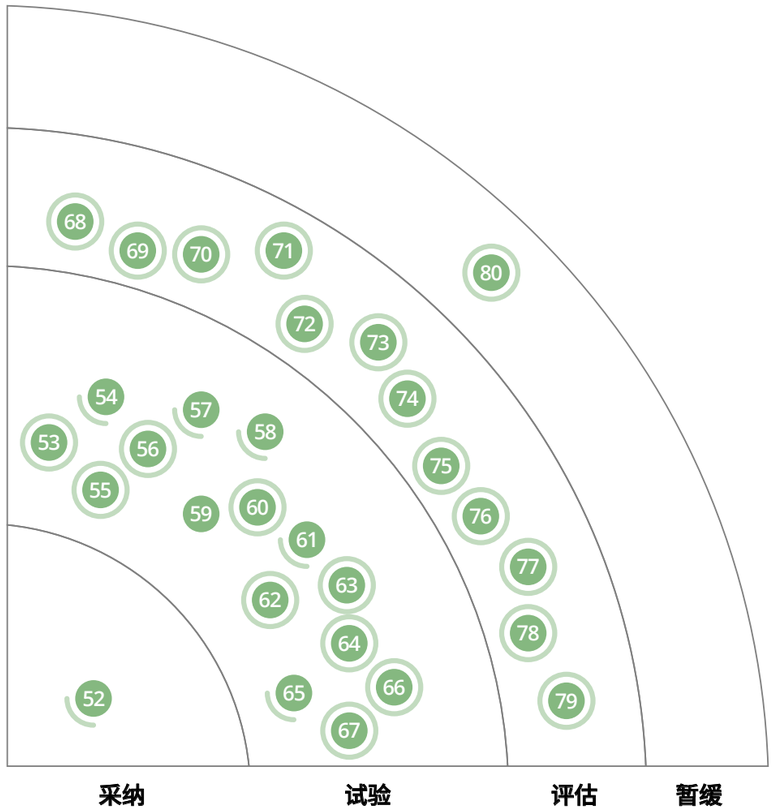
工具

工具

Sentry

采纳

在前端错误报告方面，Sentry 已经成了许多团队的默认选项。Sentry 提供了一些便利的功能，比如错误分组，以及使用适当的参数定义错误过滤规则，可以极大地帮助处理来自终端用户设备的大量错误。通过将 Sentry 集成到持续交付流水线中，你可以上传源码映射文件，从而更高效地调试错误，并能很容易追踪到是在哪个版本的软件中产生了这些错误。我们很欣赏尽管 Sentry 是一个 SaaS 产品，但它的源代码是公开的，这样就可以免费用于一些较小的用例和自托管中。



axe-core

试验

让网络具有包容性，就需要重视在软件交付的所有阶段都考虑并验证它的可访问性。许多流行的可访问性测试工具，都是为对完成后的 Web 应用程序进行测试而设计的。结果问题往往发现得较晚，并且更难解决，最终累积成为债务。最近在 Thoughtworks 网站上工作时，我们将开源可访问性（a11y）测试引擎 axe-core 纳入了构建流程。它为团队成员提供了关于遵守无障碍规则的早期反馈，即使在早期增长阶段也是如此。但是，并非每个问题都可以通过自动检查发现。商业版的 axe DevTools 扩展了 axe-core 的功能，其中包括可以指导团队成员针对大多数可访问性问题进行探索性测试。

dbt

试验

自从上一次介绍 dbt 以来，我们在一些项目中使用了它。例如，我们倾向于使用 dbt 完成 ELT 管道中转换部分的工作，使其更容易被数据消费者访问，而不是仅由数据工程师构建 ELT 管道。dbt 通过鼓励良好的工程实践，如版本控制、自动化测试和部署，来实现这一点。SQL 仍然是数据世界（包括数据库、仓库、查询引擎、数据湖和分析平台）的通用语言，大多数系统都在一定程度上支持它。这就使得这些系统可以通过构建适配器来使用 dbt 进行转换。原生连接器的数量不断增长并囊括了 Snowflake、BigQuery、Redshift 和 Postgres，社区插件的范围也在扩张。我们看到像 dbt 这样的工具正在帮助数据平台变得更加“自助”。

采纳
52. Sentry

试验
53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

评估
68. Buildah 和 Podman
69. GitHub Actions
70. Graal 原生镜像
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator 框架
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

暂缓
80. AWS CodePipeline

工具

OR-Tools 是用于解决组合优化问题的开源软件套件。这些优化问题有很多可能的解决方案，而 OR-Tools 之类的工具对于寻求最佳解决方案非常有帮助。

（OR-Tools）

esbuild

试验

我们一直渴望找到可以缩短软件开发反馈周期的工具。esbuild 就是这样一个工具。随着前端代码库逐渐变大，我们经常会遇到打包时间过长的问题。esbuild 是一个对打包速度进行了优化的 JavaScript 打包器，它可以将打包时间减少 10 到 100 倍。它是用 Golang 编写的，并且使用了更高效的方式进行解析、打印和生成 sourcemap，构建速度远远超过了 Webpack 和 Parcel 这样的工具。esbuild 可能在 JavaScript 语法转换方面没有它们全面，但这并不妨碍我们很多团队将 esbuild 作为默认的打包工具。

Flipper

试验

Flipper 是可扩展的移动应用程序调试工具。它开箱即用，支持 profiling，交互式布局检查，日志查看器以及适用于 iOS，Android 和 React Native 应用程序的网络检查器。与其他用于移动应用程序的调试工具相比，我们发现 Flipper 轻量级，功能丰富且易于设置。

Great Expectations

试验

在之前的技术雷达中，我们就关注到了 Great Expectations，我们继续看好它，并在本期技术雷达中将它挪入试验阶段。Great Expectations 这个框架可以搭建内置控件，来标记数据流水线中的异常或质量问题。正如单元测试在构建流水线中运行一样，Great Expectations 在执行数据流水线时也会进行断言。它的简单性和易用性深得我们喜爱——断言的规则用 JSON 文件存储，可以由我们的数据科学家来修改，所以不需要数据工程技能。

k6

试验

自从第一次将 k6 放入技术雷达中，我们对使用 k6 进行性能测试有了更多经验，使用结果也很不错。我们的团队很喜欢这种关注开发人员体验并且很灵活的工具，虽然 k6 很容易上手，但它真正的亮点在于很容易集成到开发人员的生态系统当中。比如 Datadog 适配器，团队能快速可视化分布式系统中的性能，并在将系统发布到生产环境之前识别重大的问题。再比如另一个团队使用 k6 的商业版本，可以使用 Azure 流水线插件商店将性能测试集成进他们的持续交付流水线中，并可以轻松获得 Azure DevOps 报告。由于 k6 支持允许自动化测试断言的阈值，因此在流水线中添加一个阶段来检测新增代码是否会导致性能下降会变得更轻松，从而为开发人员添加强大的反馈机制。

MLflow

试验

MLflow 是一款用于机器学习实验跟踪和生命周期管理的开源工具。开发和持续进化一个机器学习模型的工作流包括，一系列实验（一些运行的集合），跟踪这些实验的效果（一些指标的集合），以及跟踪和调整模型（项目）。MLflow 可以通过支持已有的开源标准，以及与这个生态中许多其他工具的良好集成，来友好地辅助这个工作流。在 AWS 和 Azure 中，MLflow 作为云上 Databricks 的受管服务，正在加速成熟，我们已经在我们的项目中成功使用过它。我们还发现 MLflow 是一个模型管理，以及跟踪和支持基于 UI 和 API 交互模型的很棒的工具。唯一的担忧在于，MLflow 作为单一平台，一直在尝试交付太多的混淆关注点，比如模型服务和打分。

OR-Tools

试验

OR-Tools 是用于解决组合优化问题的开源软件套件。这些优化问题有很多可能的解决方案，而 OR-Tools 之类的工具对于寻求最佳解决方案非常有帮助。你可以使用任何一种支持语言（Python，Java，C# 或 C++）对问题建模，然后从几种支持的开源或商业选项中选择解决方案。我们已经在整数和混合整数编程的多个优化项目中成功使用了 OR-Tools。

Playwright

试验

Playwright 允许你使用同一套 API，为

Chromium，Firefox 和 WebKit 编写 Web UI 测试。由于对所有主流浏览器引擎的支持（包括 Firefox 和 WebKit 的修复版本），这个工具已经引起了一些关注。我们持续听到关于 Playwright 积极的使用体验和反馈，尤其是它的稳定性。一些团队发现从 Puppeteer 迁移到 Playwright 也非常容易，因为它们的 API 非常相似。

Prowler

试验

我们很高兴看到基础设施配置扫描工具的可用性和成熟度都越来越好：Prowler 帮助团队扫描 AWS 基础设施配置，并根据扫描结果提高安全性。尽管 Prowler 已经存在了一段时间，但在过去的几年中，它有了长足的进步，我们也发现了通过一个较短的反馈闭环来提升项目安全性的价值。Prowler 将 AWS CIS benchmarking 分为几类（身份和权限管理，日志，监控，网络，CIS Level 1，CIS Level 2，EKS-CIS），其中包括许多检查，可以帮助你深入了解 PCI DSS 和 GDPR 合规性。

Pyright

试验

尽管鸭子类型很自然地被许多 Python 程序员看作是一个功能，但有时类型检查仍然很有用，尤其是在一些大型的代码库中。基于这个原因，许多类型注解被作为 Python 增强建议（PEP）而提出，Pyright 就是一个可以与这些注解协同工作的类型检查器。此外，它还提供了一些类型推断和类型保护，来理解条件代码流的构造。由于设计时考虑到了一些大型的代码库，Pyright 运行速度很快，它的“监视”模式允许它在修改文件时执行快速增量更新，以进一步缩短反馈周期。Pyright 可以直接通过命令行使用，但也同样可以与 VS Code，Emacs，vim，Sublime 以及其它一些编辑器集成使用。在我们的经验中，相比 mypy 之类的替代方案，我们更青睐 Pyright。

Redash

试验

采用“谁构建，谁运行”的 DevOps 理念，意味着团队已经增加了对技术和业务指标的关注，这些指标可以从他们部署的系统中提取出来。我们经常发现，分析工具对于大多数开发人员来说都很难使用，所以捕获和呈现指标的工作就留给了其他团队——而这已经是在交付功能给最终用户很久之后的事了。我们的团队发现，Redash 可以让普通开发人员以自助的方式创建仪表板，这对于查询产品指标非常有用，它缩短了反馈周期，并让整个团队专注于业务产出。

Terratest

试验

Terratest 在过去曾经作为基础设施测试的一个选项，吸引了我们的注意。从那时候起，我们的团队就一直在使用它，并且对它的稳定性及它所提供的体验感到非常兴奋。Terratest 是一个 Golang 库，用来简化基础设施代码的自动化测试编写。通过基础设施即代码的工具，例如 Terraform，你可以创建真实的基础设施组件（如服务器、防火墙或负载均衡器），在它们之上部署应用程序，并使用 Terratest 验证预期的行为。在测试结束后，Terratest 可以取消应用的部署并清理资源。对于基础设施在真实环境中的端到端测试，该工具非常有用。

Tuple

试验

Tuple 是一个相对较新的远程结对编程工具，它希望能够填补 Slack 放弃 Screenhero 后留下的市场空白。虽然它现在还有比如暂时只能在 Mac OS 上使用（即将支持 Linux），以及一些界面问题需要解决。但即使有这些限制，我们也已经在使用中拥有了很好的体验。和类似 Zoom 这种通用的视频屏幕共享工具不同，Tuple 支持有两个鼠标光标的双侧控制；也和其他的选择比如 Visual Studio Live Share 不同，它不需要受限于 IDE 内。Tuple 支持语音和视频通话、剪切板共享和相对其他工具较低的延迟；你还可以轻松的在另一方的屏幕上画画和擦除。这些特性让 Tuple 成为了一款十分方便直观和对开发者友好的工具。

Why Did You Render

试验

使用 React 开发时，我们经常会遇到的情况是：因为某些组件进行了不必要的重复渲染而导致页面性能很差。Why Did You

工具

Tuple 是一个相对较新的远程结对编程工具，它希望能够填补 Slack 放弃 Screenhero 后留下的市场空白。

(Tuple)

工具

imgcook 是阿里巴巴旗下的软件即服务产品，它可以通过智能化技术把不同种类的视觉稿（Sketch/PSD/ 静态图片）一键生成前端代码。

(imgcook)

Render 是一个能够帮助侦测组件重新渲染的库。它是通过对 React 的猴子补丁实现了这一点。我们在一些项目中使用它来调试性能问题，并取得了很好的效果。

Buildah 和 Podman

评估

尽管 Docker 已经成为容器化的默认选项，这个领域的新玩家还是吸引了我们的注意 Buildah 和 Podman 就是这种情况，它们是在多个 Linux 发行版中使用 rootless 方法构建映像（Buildah）和运行容器（Podman）的互补项目。Podman 引入了一个无守护进程的引擎，来管理和运行容器，与 Docker 相比，这是一个有趣的方法。事实上，Podman 可以使用 Buildah 或 Docker 创建的 OCI（Open Container Initiative）镜像，这样工具变得更有吸引力，也更容易使用。

Github Actions

评估

在我们的工具箱中，CI 服务器与构建工具最为古老，使用也最广泛。其工作范围包括简单的云托管服务，一直到复杂的、由代码定义的、支持大型构建机群的流水线服务。鉴于市面上已经有大量的同类产品，当另一个用于管理构建与集成工作流 GitHub Actions 出现时，基于经验我们起初抱有怀疑的态度。但是 GitHub Actions 为开发者提供了小步启动并可以轻松自定义的行为，并逐渐成为小型项目的默认选项。将构建工具直接集成到源代码库实在很方便，因此社区也很活跃。大量用户贡献了工具以及工作流，便于快速上手。工具供应商也可以通过 GitHub Marketplace 提供服务。但是，我们仍然建议你保持谨慎。因为尽管代码和 Git 历史记录可以导出到其他服务器中，但是基于 GitHub Actions 的开发工作流程却不能。另外，你也需要自行判断何时项目会变得足够大或足够复杂，而需要使用有独立支持的流水线工具。但对于初创而需要快速运行的、较小的项目，仍然值得考虑 GitHub Actions 及其不断发展的生态系统。

Graal 原生镜像

评估

Graal 原生镜像是一种以静态链接可执行文件或共享库的形式，将 Java 代码编译为操作系统本机二进制代码的技术。原生镜像经过优化，减少了应用程序的内存占用和启动时间。我们的团队已经成功地在 serverless 架构中，将 Graal 原生镜像作为小型 Docker 容器执行，减少了启动时间。尽管 Graal 原生镜像是为与 Go 或 Rust 等编程语言一起使用而设计的，这些编程语言需要本机编译，需要更小的二进制文件尺寸和更短的启动时间，但对于有其他需求并希望使用基于 jvm 的语言的团队来说，Graal 原生镜像也同样有用。

Graal 原生镜像构建器，native-image，支持基于 jvm 的语言 —— 如 Java、Scala、Clojure 和 Kotlin——并能在多个操作系统上构建可执行文件，如 Mac OS、Windows 和众多 Linux 发行版。由于它需要一个封闭的假设，即所有代码在编译时都是已知的，因此需要对诸如反射或动态类加载这样的特性进行额外的配置，因为不能在构建时仅从代码推断出类型。

HashiCorp Boundary

评估

在代理访问你的主机和服务的场景下，安全网络和身份管理的能力是不可或缺的，HashiCorp Boundary 将这些能力合并在一处，如果需要，还可以连接多种云服务和本地自行部署的资源。密钥管理可以通过集成你选择的密钥服务来实现，无论是云厂商提供的密钥服务，还是诸如 HashiCorp Vault 这样的工具。HashiCorp Boundary 支持越来越多的身份认证提供方，并且可以集成到你的服务整体架构当中，来帮助定义主机甚至是服务级别的权限。比如说，它可以用来实现对 Kubernetes 集群的细粒度访问控制。HashiCorp Boundary 也正在继续开发以支持更多功能，诸如从不同的来源动态拉取服务目录。所有这些实现都被封装在 HashiCorp Boundary 当中，对作为终端用户的、习惯于 shell 使用体验的工程师来说是不可见的，这一切都是通过 Boundary 的网络管理层安全地进行连接。

imgcook

评估

还记得在研究项目 pix2code 中，如何通过图形用户界面的截图自动生成代码吗？现在这个技术已经出现了产品化的版本—imgcook，它是阿里巴巴旗下的软件即服务产品。它可以通过智能化技术把不同种类的视觉稿（Sketch/PSD/ 静态图片）一键生成

前端代码。在双十一购物狂欢节期间，阿里巴巴需要定制大量的活动广告页面。经常会有一次性页面需要被快速开发完成。通过深度学习方法，用户体验设计师的设计，首先被处理为前端代码，然后由开发人员进行调整。我们的团队正在评估这项技术：尽管图像处理是在服务器端进行的，主页界面却在网页上，imgcook 提供可以集成软件设计及开发生命周期的工具。imgcook 可以生成静态代码，如果你定义了领域专用语言，它也可以生成数据绑定模块代码，该技术还没达到完美的程度，设计人员需要参考某些规范，以提高代码生成的准确性（此后仍需开发人员的调整）。我们对于魔术代码生成一直十分谨慎，因为从长远看，生成的代码通常很难维护，imgcook 也不例外。但是如果你限定它用于特定的上下文，例如一次性活动广告页，这项技术值得一试。

Longhorn

评估

Longhorn 是 Kubernetes 的分布式块存储系统。Kubernetes 有很多持久性存储选项，但是，Longhorn 与大多数存储选项不同，它是从头开始构建的，提供了增量快照和备份，从而减轻了在非云托管的 Kubernetes 上运行复制存储的痛苦。随着 Longhorn 最近对 ReadWriteMany (RWX) 的实验性支持，你甚至可以挂载相同的存储卷，来进行跨多个节点的读写访问。为 Kubernetes 选择合适的存储系统是一项艰巨的任务，我们建议你根据需求评估使用 Longhorn。

Operator 框架

评估

Operator 框架是一套开源工具，可简化 Kubernetes operators 的构建和生命周期管理。Kubernetes operator 模式最初由 CoreOS 引入，是一种使用 Kubernetes 原生能力来封装操作应用程序知识的方法；它包括要管理的资源和确保资源与其目标状态匹配的控制器代码。这种方法已被用于扩展 Kubernetes，以原生化管理众多应用程序，特别是有状态的应用程序。Operator 框架有三个组件：Operator SDK，简化了 Kubernetes operators 的构建、测试和打包；Operators 生命周期管理器负责 operators 的安装、管理和升级；以及发布和共享第三方 operators 的目录。我们的团队发现 Operator SDK 在快速开发 kubernetes 原生应用程序时特别强大。

Recommender

评估

大型云服务供应商的服务数量持续增长，对应工具的便利性和成熟度也在不断提高。Recommender 是谷歌云的一项服务，可以分析现有资源，并根据实际使用量给出如何优化的建议。该服务包含了针对安全、计算使用以及节省成本等方面的一系列 Recommender。例如，IAM 角色通过指出未曾使用过的可能过于宽泛的权限，来帮助更好地实现最小权限原则。

Remote - WSL

评估

在过去的几年里，我们曾经讨论过几次 Windows Subsystem for Linux（WSL）。尽管我们欣喜于看见 WSL 的进步，比如 WSL2 的改进，却一直没有将它收录在技术雷达中。这一期里，我们想要着重介绍一个 Visual Studio Code 插件，它能够显著提升 WSL 的使用体验。Windows 中的代码编辑器虽然可以访问 WSL 文件系统，但是并不能感知隔离的 Linux 环境。然而有了 Remote - WSL 插件，Visual Studio Code 不仅可以感知 WSL，还允许开发者启动 Linux shell，并且支持在 Windows 下调试 WSL 里运行的二进制代码。在 WSL 支持中可以看到，Jetbrains 的 IntelliJ 也在这方面做出了稳步的改进。

Spectral

评估

我们在本期技术雷达中反复看到的一种模式是，当一种新的语言变得流行以后，静态错误和样式检查工具会迅速浮现出来。这些工具通常被称作 linters——以经典且深受欢迎的可以静态分析 C 代码的 Unix 工具 lint 命名。我们喜欢这些工具，因为它们会更早捕获异常，甚至在代码未编译之前。这个模式最新的例子是关于 YAML 和 JSON 的 linter Spectral。尽管 Spectral 可用作这些文件格式的通用检查工具，但它的主要目标是

工具

Recommender 是谷歌云的一项服务，可以分析现有资源，并根据实际使用量给出如何优化的建议。

(Recommender)

工具

Zally 是一个简便的基于 OpenAPI 的代码扫描工具，它有助于确保 API 遵循团队制定的 API 样式指南。

(Zally)

OpenAPI（Swagger 和 AsyncAPI 的演化版本）。Spectral 为这些规范提供了全面的开箱即用的规则，帮助开发者们在设计和实现 API 或事件驱动协作中避免麻烦。这些规则可以检查 API 参数规范或者规范中存在的许可声明等。虽然这个工具成为 API 开发工作流中广受欢迎的补充，它仍然提出了一个问题：即非执行文件的规范是否应该如此复杂，以至于需要为编程语言设计专门的错误检查技术。也许开发者们应该写的是代码而非规范？

Yelp detect-secrets

评估

Yelp detect-secrets 是一个用于检测代码库中存储的密码的 Python 模块；它会扫描一个路径下所有的文件寻找密码。它可以被用作 Git 预提交钩子或在 CI/CD 流水线的适当位置来进行扫描。它的默认配置上手十分容易，如果有需要也可以进行自定义配置。你还可以安装自定义插件去扩充它默认的启发式搜索。与其他相似的工具比较，我们发现这款工具光以开箱即用的配置，就可以检测到更多种类的密码。

Zally

评估

随着 API 规范生态系统的成熟，我们看到了更多可以自动化检查样式的工具。Zally 是一个简便的基于 OpenAPI 的代码扫描工具，它有助于确保 API 遵循团队制定的 API 样式指南。以开箱即用的方式，Zally 会针对为 Zalando 的 API 样式指南开发的规则集进行验证，同时它还支持基于 Kotlin 扩展机制开发自定义的样式规则。Zally 提供了直观的 UI 界面，展示样式违规的地方，同时也提供了命令行工具，这样可以轻松地集成到持续交付流水线中。

AWS CodePipeline

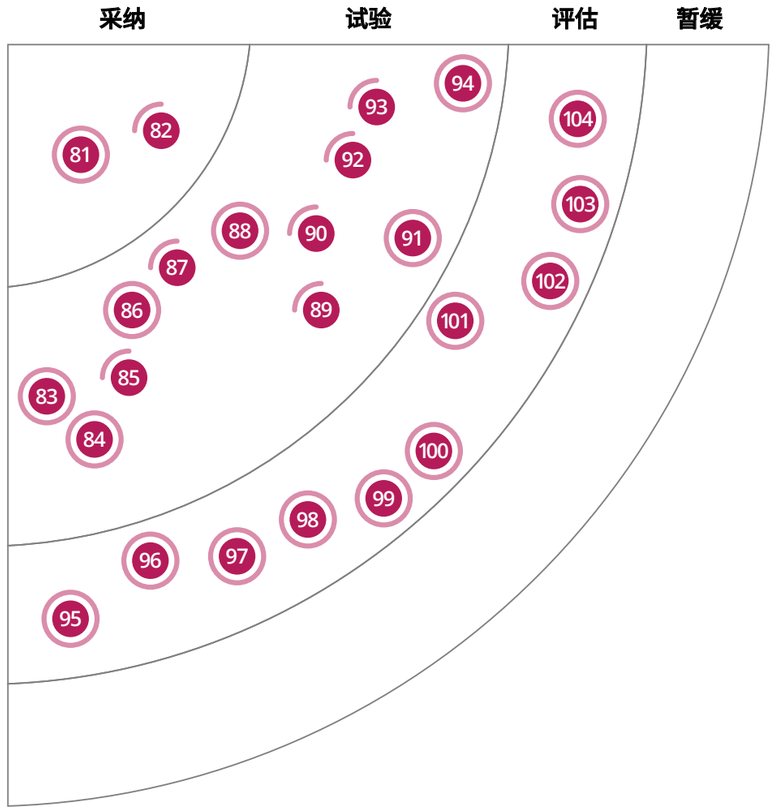
暂缓

根据 Thoughtworks 多个团队的使用经验，我们建议你谨慎使用 AWS CodePipeline。具体来说，我们发现一旦团队的需求超出简单的流水线范畴，此工具就会变得难以使用。尽管初次使用 AWS 时，像是赢得了“快速的胜利”，但我们建议你后退一步，评估 AWS CodePipeline 是否可以满足你的长期需求，例如流水线的 fan-out 和 fan-in，或者是更复杂的部署，以及具有特殊依赖关系及触发条件的测试场景。

TECHNOLOGY RADAR

语言&框架

语言&框架



Combine

采纳

我们几年前把 ReactiveX （反应式编程开源框架中的一个系列）移到了技术雷达的“采纳”环中。2017 年，我们提到了 RxSwift，它可以将反应式编程应用到基于 Swift 的 iOS 开发中。此后，Apple 以 Combine 的形式推出了自己的反应式编程框架。对于仅支持 iOS 13 及更高版本的 App 而言，Combine 已经成为默认的选择。它比 RxSwift 更容易学习，并且与 SwiftUI 集成得很好。如果您想要将现有项目框架从 RxSwift 转换为 Combine，或者在一个项目中同时使用两者，可以了解一下 RxCombine。

LeakCanary

采纳

如今，我们的移动开发团队认为 LeakCanary 是 Android 开发中的默认选项。它的集成极其简单，同时提供能够清晰回溯内存泄漏原因的通知，从而帮助我们侦测到 Android 上恼人的泄漏问题。我们推荐你把 LeakCanary 加入工具包，它可以帮你节省在多个设备上排查内存泄漏的时间。

Angular Testing Library

试验

随着继续使用 JavaScript 开发 Web 应用，我们也从 Testing Library 的应用测试方法中获益良多；并且继续针对它的一系列软件包进行探索和经验积累，不仅仅限于 React Testing Library。Angular Testing Library 在以用户为中心测试 UI 组件的方面上拥有这个大家庭的全部优势，鼓励测试用户行为，而非 UI 细节的实现方式，使得测试更具有可维护性。虽然缺乏完善的文档，但 Angular Testing Library 提供了良好的测试示例来帮助我们针对于不同场景更好地上手。我们已经成功地在 Angular 项目上应用了这个测试库，并且建议你们也能尝试一下这个可靠的测试方法。

AWS Data Wrangler

试验

AWS Data Wrangler 是一个开源库，可以将数据框连接到 AWS 数据相关的服务，从而

采纳
81. Combine
82. LeakCanary

试验
83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. 按需分发模块
92. Streamlit
93. SWR
94. TrustKit

评估
95. .NET 5
96. bUnit
97. Dagster
98. Flutter for Web
99. Jotai 和 Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 模块联邦

暂缓

语言&框架

FastAPI 是使用 Python 3.6 或更新版本来构建 API 的现代、快速（高性能）的 Web 框架。

(FastAPI)

将 Pandas 的功能扩展到 AWS。另外，该库还利用 Apache Arrow 和 Boto3 暴露了一些 API，用于从数据湖和数据仓库中加载、转换和保存数据。AWS Data Wrangler 最大的限制是不支持大型的分布式数据流水线操作。但是，你可以使用原生的数据服务（如 Athena、Redshift 和 Timestream 等）进行大批量的数据上传和提取工作，来表示适用于数据框的复杂转换。我们在生产环境中使用过 AWS Data Wrangler。它可以使你专注于编写转换，而不必在连接 AWS 数据服务上花费太多时间。

Blazor

试验

虽然 JavaScript 及其生态系统已经在 Web UI 领域占据了统治地位，但随着 WebAssembly 的出现，一些新的机遇之窗也正在打开。Blazor 依然值得我们的关注；我们的团队使用 C# 在 WebAssembly 之上构建交互丰富的用户界面，取得了良好的效果。同时，能够在前端使用 C#，也意味着可以共享代码和复用现有的库。除此以外，配合使用 bUnit 这类现存的调试和测试工具，这个开源框架值得一试。

FastAPI

试验

越来越多的团队正在使用 Python 作为他们解决方案的首选语言，这不仅针对数据科学，也同样对于后端服务。在这些场景中，FastAPI 给了我们优秀的体验。这是一个使用 Python 3.6 或更新版本来构建 API 的现代、快速（高性能）的 Web 框架。不仅如此，该框架及其生态还包括了诸如用 OpenAPI 来创建 API 文档这样的功能，使团队可以聚焦在业务功能并快速创建 REST API。这些特性使 FastAPI 在这个领域成为了现有解决方案的良好替代品。

io-ts

试验

一直以来我们都很享受使用 TypeScript 的体验，喜欢它的强类型带来的安全性。然而，当获取的数据（如调用后端服务返回的数据）与 TypeScript 类型定义不一致时，却可能会导致运行时错误。一个叫做 io-ts 的库可以帮我们解决这个问题。它通过提供编码和解码的功能，帮我们弥补了外部数据在编译期类型检查和运行时数据消费之间的鸿沟。它也可以用作自定义类型保护。随着在工作中获得越来越多使用 io-ts 的经验，我们对它最初的好印象得到验证，并且现在仍然喜欢这种优雅的方式。

Kotlin Flow

试验

Kotlin 协程的引入为 Kotlin 的创新打开了一扇大门——直接集成到协程库中的 Kotlin Flow 就是其中之一。Kotlin Flow 是一种基于协程的响应式流的实现。与 RxJava 不同的是，流是 Kotlin 原生的 API，与熟悉的序列 API 类似，包括 map 和 filter 方法。跟序列一样，流是“冷”的，这就意味着只有当需要使用的时候才构造序列的值。所有这些特性使多线程代码的编写比其他方法更加简单和易于理解。可以预见，通过 toList 方法将流转换成列表将会成为测试中的一种常见模式。

LitElement

试验

自从我们于 2014 年第一次提到 Web Components 以来，它已经取得了稳步的发展。作为 Polymer 项目的一部分，LitElement 是一个简单的库，你可以使用它创建轻量级 Web 组件。它实际上只是一个基类，它去掉了很多常见的模板，从而使编写 Web 组件变得更加容易。我们已经在项目上成功地使用了它，并且随着技术的成熟和 LitElement 库的广受欢迎，它在 Web Components 项目中的应用将会越来越普遍。

Next.js

试验

在上一次我们介绍了有关如何使用 Next.js 构建 React 代码库之后，我们又有了更多的使用经验。Next.js 是一个一站式零配置的框架，该框架提供了包括简化的路由、采用

Webpack 和 Babel 进行自动打包和编译、快速热重载等其它可为开发人员提供便利工作流的工具。它默认提供服务器端渲染、搜索引擎优化和改善初始加载时间等功能，并支持增量静态生成模块代码。使用过 Next.js 的团队为我们提供了积极的体验报告，同时鉴于其庞大的社区，我们对该框架的发展保持乐观的态度。

按需分发模块

试验

适用于 Android 系统的按需分发模块是一个框架，允许为适当结构的 App 下载和安装仅包含所需功能的量身定制的 APK。对于下载速度可能堪忧的大型应用，或用户可能仅在初次安装时使用某些功能的应用来说，此框架值得一试。它还可以简化对多个设备的处理，而无需使用不同的 APK。iOS 平台下也有类似的框架。

Streamlit

试验

Streamlit 是开源的 Python 应用程序框架，供数据科学家使用，以构建交互式数据应用程序。调整机器学习模型需要花费大量时间，但是我们可以在 Streamlit 中快速构建独立的原型，并在实验循环中收集反馈，而不用在主程序（使用模型的应用程序）中反复调整。Streamlit 相较于 Dash 等竞争对手表现更佳出众，因为它专注于快速原型开发，并支持大量的可视化库（例如 Plotly 以及 Bokeh）。我们已经在几个项目中使用了 Streamlit，并且很喜欢用它轻松地构建交互式可视化应用。

SWR

试验

我们团队发现在合适的情况下，使用 React Hooks 的库 SWR 可以达到代码整洁和性能大幅提升的效果。SWR 实现了更新的同时使用过期数据 （stale-while-revalidate，缩写即 SWR）的 HTTP 缓存策略，即第一次从缓存中返回（过期）数据，然后发送拉取数据的请求（更新），最后用最新的返回数据刷新缓存。我们告诫团队仅在应用程序应返回过期数据时才使用 SWR 缓存策略。需要注意的是，HTTP 要求以最新的数据缓存来响应请求，而只能在经过深思熟虑的情况下才允许返回过期的响应。

TrustKit

试验

我们在使用 SSL 公钥绑定时需要格外小心。因为一旦选择了错误的安全策略或者忘记进行绑定备份，应用程序可能会因为异常而停止运行。这正是 TrustKit 的有用之处。它是 iOS 平台上用于简化 SSL 公钥绑定的一个开源框架，同样也支持了 Android 平台。关于如何选择合适的绑定策略，以及更多细节问题的指导，您可以查阅这份上手指南。我们已经在多个项目的生产环境中运用了 TrustKit，效果很好。

.NET 5

评估

我们不会在雷达中介绍每一个新的 .NET 版本，但 .NET 5 意味着在将 .NET Core 和 .NET Framework 合并为单一平台方面迈出了重要一步。各组织应该开始制定策略，当 .NET 5 或 6 版本可用时，将他们的开发环境（根据部署目标的不同而混合不同的框架）迁移到单一版本的 .NET 5 或 6。这种方法的优势将是一个通用的开发平台，不必考虑预期环境：Windows、Linux、跨平台移动设备（通过 Xamarin）或浏览器（使用 Blazor）。虽然对于有工程文化支持的公司来说，多语言开发仍将是首选方法，但其他公司会发现在单一平台上进行标准化的 .NET 开发更有效率。目前，我们希望将其保留在“评估”环中，看看 .NET 6 中的最终统一框架表现如何。

bUnit

评估

bUnit 是一款专门为 Blazor 量身打造的测试库，它可以在已有的单元测试框架（如 NUnit、xUnit 或 MSUnit） 中轻松地创建 Blazor 组件测试。它为组件提供了一个外观（facade），可以在熟悉的单元测试范式中运行和测试，从而允许快速反馈并对组件测

语言&框架

Streamlit 是开源的 Python 应用程序框架，供数据科学家使用，以构建交互式数据应用程序。

(Streamlit)

语言&框架

Jotai 和 Zustand 都是 React 的状态管理包，且目标都是小巧易用。也许不完全是巧合，两者的名字分别是日语和德语中对于单词“状态”的翻译。

(Jotai 和 Zustand)

试进行隔离。如果你正在开发 Blazor，我们建议你把 bUnit 添加到工具列表并进行尝试。

Dagster

评估

Dagster 是一个用于机器学习、机器分析，以及纯 ETL 数据管道的开源数据编排框架。与其他任务驱动框架不同，Dagster 知晓流经流水线的数据并可以提供类型安全性。借助对流水线和产生资产的统一视图，Dagster 可以计划和编排 Pandas，Spark，SQL，或任何 Python 可调用的接口。该框架相对较新，我们建议您对其在数据流水线中的能力进行评估。

Flutter for Web

评估

到目前为止，Flutter 主要支持 iOS 和 Android 原生应用。但是 Flutter 团队的愿景是让 Flutter 支持在任何平台上构建应用。Flutter for Web 朝此方向迈出了一步，让我们能够基于同一代码库构建适用于 iOS、Android 和浏览器的应用。它的测试版在一年前已处于可用状态，而随着最近 Flutter 2.0 版本的发布，Flutter for Web 迎来了一个稳定的里程碑版本。在此版本中，Flutter 团队专注于渐进式 Web 应用、单页面应用，以及将现有的移动应用扩展到 Web 平台。它的应用和框架代码（全部基于 Dart）也被编译为 JavaScript，而非用于移动应用的 ARM 机器码。Flutter 的 Web 引擎为我们提供了两种渲染器：使用 HTML，CSS，Canvas 和 SVG 的 HTML 渲染器，以及使用 WebAssembly 和 WebGL 将 Skia 绘制命令展现到浏览器 canvas 的 CanvasKit 渲染器。我们有一些团队已经开始使用 Flutter for Web，并对试用结果表示满意。

Jotai 和 Zustand

评估

我们在之前的技术雷达中评价过尚处试验开始阶段的 React 应用程序状态管理。我们将 Redux 移回到试验环，标明它不再是我们的默认选项，并且提到了 Facebook 的 Recoil。在这一期的技术雷达中，我们要强调的是 Jotai 和 Zustand。它们都是 React 的状态管理包，且目标都是小巧易用。也许不完全是巧合，两者的名字分别是日语和德语中对于单词状态的翻译。除了这些相同点，两者的设计也是有所不同的。Jotai 的设计与 Recoil 相似，都是状态由存储在 React 组件树中的原子组成，而 Zustand 将状态存储在 React 外部的单个状态对象中，就像 Redux 采取的方式一样。Jotai 的作者提供了一个非常有用的对照表，可供你决策使用哪一个工具。

Kotlin Multiplatform Mobile

评估

在跨平台移动开发的趋势下，Kotlin Multiplatform Mobile （KMM） 成为了该领域的新成员。KMM 是 JetBrains 提供的 SDK，它利用了 Kotlin 的跨平台能力，包含丰富的工具和特性，使整个构建跨平台移动应用的体验变得更加愉悦和高效。使用 KMM 时，您只需用 Kotlin 对业务逻辑和核心应用程序编写一份代码，然后即可被 Android 和 iOS 应用共享。仅在必要时（如利用原生 UI 元素），您才需要针对特定的平台编写代码，且这些特定代码保存在各个平台的不同视图中。尽管仍处在 Alpha 测试阶段，Kotlin Multiplatform Mobile 正在高速演进中。我们一定会密切关注它，希望您也如此。

LVGL

评估

随着智能家居和可穿戴设备的日益普及，对直观的图形用户界面（GUI）的需求越来越大。但是，如果从事嵌入式设备的开发，而不是 Android/iOS，GUI 的开发可能需要花费很多精力。作为一个开源的嵌入式图形库，LVGL 越来越受欢迎。LVGL 已经适配了主流的嵌入式平台，如 NXP、STM32、PIC、Arduino 和 ESP32。它的内存占用空间非常小：64 kB 闪存和 8 kB RAM 就足以让它工

作，而且它可以在各种 Cortex-M0 低功耗 MCU 上平稳地运行。LVGL 支持触摸屏、鼠标和按钮等输入类型，并包含 30 多个控件，包括适合智能手表的 TileView。它选择的 MIT 授权并不限制企业和商业用途。我们的团队对这个工具的反馈是积极的，且一个使用 LVGL 的项目已经投入生产，更确切地说是小批量生产。

React Hook Form

评估

构建 Web 表单仍然是前端开发的长期挑战之一，尤其在使用 React 技术栈的时候。我们有很多团队在使用 React，他们也一直在使用 Formik 来简化这个过程。但是现在一些团队正在评估 React Hook Form 作为潜在的替代方法。React Hooks 在 React Hook Form 出现之前就已经存在，因此可以将它们作为一级概念：该框架通过钩子将表单元素注册和跟踪为不受控制的组件，从而大大减少了重新渲染的需要。它的大小和所需的样板代码数量也非常轻量级。

River

评估

众多机器学习方法的核心皆在于从一组训练数据创建一个模型。一旦创建了模型，就可以反复使用它。然而世界并不是静止的，通常模型需要随着新数据的出现而改变。单纯地重新训练模型可能会非常缓慢和昂贵。增量学习解决了这个问题，它使从数据流中增量地学习成为可能，从而更快地对变化做出反应。作为额外的好处，计算和内存需求更低，而且是可预测的。我们在基于 River 框架的实现中积累了良好的经验，但到目前为止，我们需要在模型更新后增加校验，有时要手动进行。

Webpack 5 模块联邦

评估

Webpack 5 模块联邦功能的发布，受到了微前端架构开发者的高度期待。该功能引入了一种更加标准化的方式来优化模块依赖和共享代码的管理与加载。模块联邦允许共享模块规范，通过一次加载多个模块使用的代码，从而帮助消除微前端之间的重复依赖。它还能够区分本地模块和远程模块，远程模块实际上并不是构建的一部分，而是异步加载的。与诸如 npm 软件包之类的构建时依赖项相比，这可以显著简化那些随着多个下游依赖而更新的模块的部署。但是请注意，这要求你将所有微前端与 Webpack 捆绑在一起，而不是像导入映射那样，最终可能会成为 W3C 标准的一部分。

语言&框架

机器学习方法的模型通常需要随着新数据的出现而改变。增量学习解决了这个问题，它使从数据流中增量地学习成为可能，从而更快地对变化做出反应。我们在基于 River（用于在线机器学习的 Python 库）框架的实现中积累了良好的经验。

(River)

/thoughtworks

Thoughtworks 是一家软件咨询公司，也是一个充满热情、以目标为导向的社区。我们帮助客户以技术为核心，推动其商业变革，与他们并肩作战解决最核心的技术问题。我们致力于积极变革，希望能够通过软件技术创造更美好的社会，与此同时我们也与许多志向相投的组织合作。

创办 27 年以来，Thoughtworks 已经从一个小团队，成长为现在拥有超过 9000 人，分布于全球 17 个国家、拥有 48 间办公室的全球企业。

想要了解技术雷达最新的新闻和洞见？
请选择你喜欢的渠道来关注我们

现在订阅

ThoughtWorks 洞见

/thoughtworks

thoughtworks.com/radar

#TWTechRadar