技术雷达

有态度的前沿技术解析

/thoughtworks

第 26 期

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。

技术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。

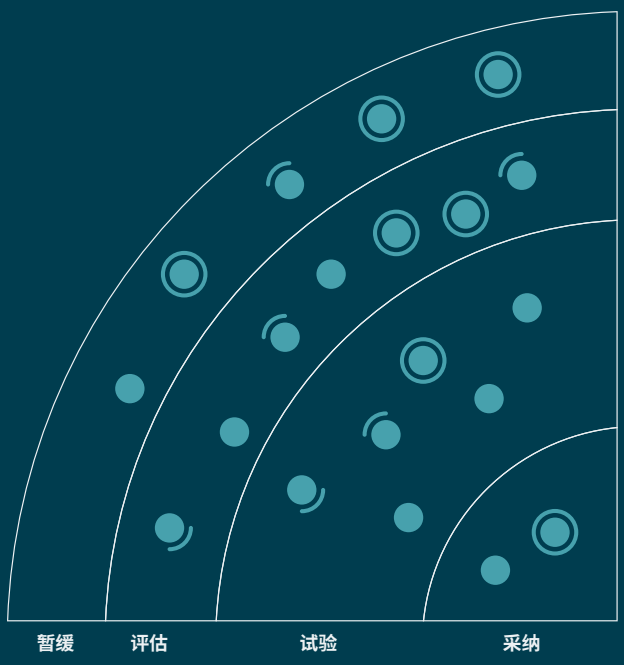
我们建议您探究这些技术以了解更多细节。技术雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言和框架。如果雷达技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。

想要了解更多技术雷达相关信息，请点击：
thoughtworks.com/cn/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，因此您会发现它们在雷达中的位置也会改变。



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

新的 **挪进 / 挪出** **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，我们挪出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。

贡献者

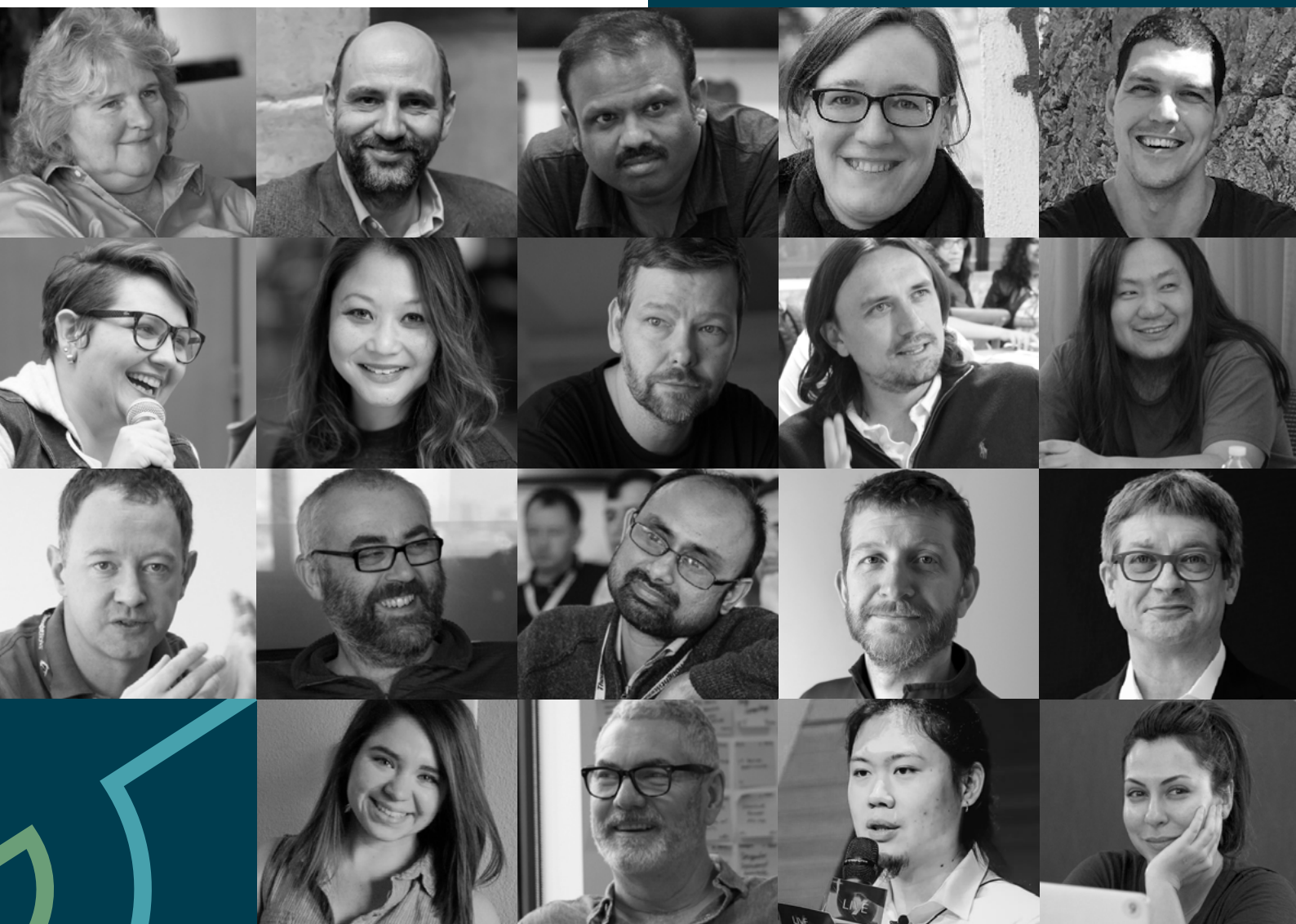
技术顾问委员会（TAB）由 Thoughtworks 的 18 名高级技术专家组成。TAB 每年召开两次面对面会议，每两周召开一次电话会议。技术顾问委员会由 Thoughtworks 首席技术官 Rebecca Parson 组建。

技术雷达由 Thoughtworks 技术顾问委员会（TAB）创建。受疫情影响，本次技术雷达仍然通过线上讨论完成。

中国区技术雷达汉化组：

徐培 | 王妮 | 杨光 | 杨洋 | 张凯峰 | 禚娴静 | 边晓琳 | 邓奕 | 樊卓文 | 费庆毅 | 谷中仁 | 郭晨 | 韩宇坤 | 黄浩 | 黄进军 | 江文韬 | 蒋宏宇 | 李辉 | 李嘉骏 | 李妍 | 梁晶 | 梁若琳 | 林晨 | 马宇航 | 孟然 | 孟瑶 | 闵锐 | 潘茂茂 | 潘啸 | 彭天俊 | 孙郁俨 | 田彪 | 王梦蛟 | 王启瑞 | 王乔阳 | 王若宇 | 王薇 | 王桢琦 | 魏斯特 | 吴瑞峰 | 向博 | 熊欣 | 阳沁珂 | 杨君君 | 杨开广 | 杨麟 | 杨旭东 | 杨阳 | 张广洁 | 张丽 | 张颖嘉 | 朱雪晴

Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
Bharani Subramaniam
Birgitta Böckeler
Brandon Byars
Camilla Falconi Crispim
Cassie Shum
Erik Dörnenburg
Fausto de la Torre
Hao Xu
Ian Cartwright
James Lewis
Lakshminarasimhan Sudarshan
Mike Mason
Neal Ford
Perla Villarreal
Scott Shaw
Shangqi Liu
Zhamak Dehghani



本期主题

离奇集市：开源软件不断变化的经济学

Thoughtworks 一直以来都是开源软件的粉丝。因为 Eric Raymond 发表的著名文章《大教堂与集市》，开源软件普及开来，提高了开发者的能动性，并用众包的形式修复错误，以及创新。然而，商业化的尝试呈现出当前生态系统巨大的经济复杂性。例如，Elastic 为了要求从中获利的云服务提供商做出回馈，更改了许可证，这导致 AWS 在 2021 年 9 月将 Elasticsearch 复刻为 OpenSearch。这表明商业开源软件要守护竞争的护城河有多么困难（同样的问题也适用于免费的闭源软件，因为 Docker 一直在努力寻找合适的商业模式，我们目睹了一些公司在探索 Docker Desktop 的替代品）。有时，权力动态会反过来：由于 Facebook 资助了开源软件 Presto，Presto 的贡献者得以保留 IP（知识产权），并在他们离开公司后将其重新命名为 Trino，这实际上得益于 Facebook 的投资。大量关键基础设施都不是由企业赞助，这会让情况会变得更加混乱，通常只有在发现关键安全漏洞时，这些企业才会注意到他们对无偿劳动的依赖程度（就像最近发生的 Log4J 问题）。在某些情况下，通过 GitHub 或 Patreon 资助业余爱好者和维护者，可以提供足够的动力来产生影响；但对其他人而言，它是在他们的日常工作之上，增加了额外的责任感，并会导致倦怠。我们仍然是开源软件的坚定支持者，但也认识到经济学正变得越来越离奇，并且没有简单的解决方案来找到正确的平衡。

软件供应链的创新

众所周知的重大安全事故——Equifax 数据泄露、SolarWinds 攻击、Log4J 远程零日漏洞攻击等等，都是由于软件供应链的管理不善造成的。开发团队现在意识到，可靠的工程实践也包括项目依赖项的验证和管理，因此本期技术雷达上出现了许多与之相关的条目。这些条目包括一些检查清单和标准，如**软件工件供应链层级（SLSA）**：一个由谷歌主推，旨在为供应链的常规威胁提供指导；以及 **CycloneDX**：一个由 OWASP 社区推动的另一套标准。我们还介绍了一些具体的工具，如 **Syft**，它能够为容器镜像生成**软件物料清单（SBOM）**。黑客越来越多地利用安全领域攻防不对称的特点，并且结合日益复杂的黑客技术进行攻击——这意味着他们只需要找到一个漏洞即可，而防御者必须确保整个攻击面的安全。在我们努力保持系统安全的过程中，改善供应链安全是我们应对措施中的关键部分。

为什么开发者们总热衷于在 React 中实现状态管理?

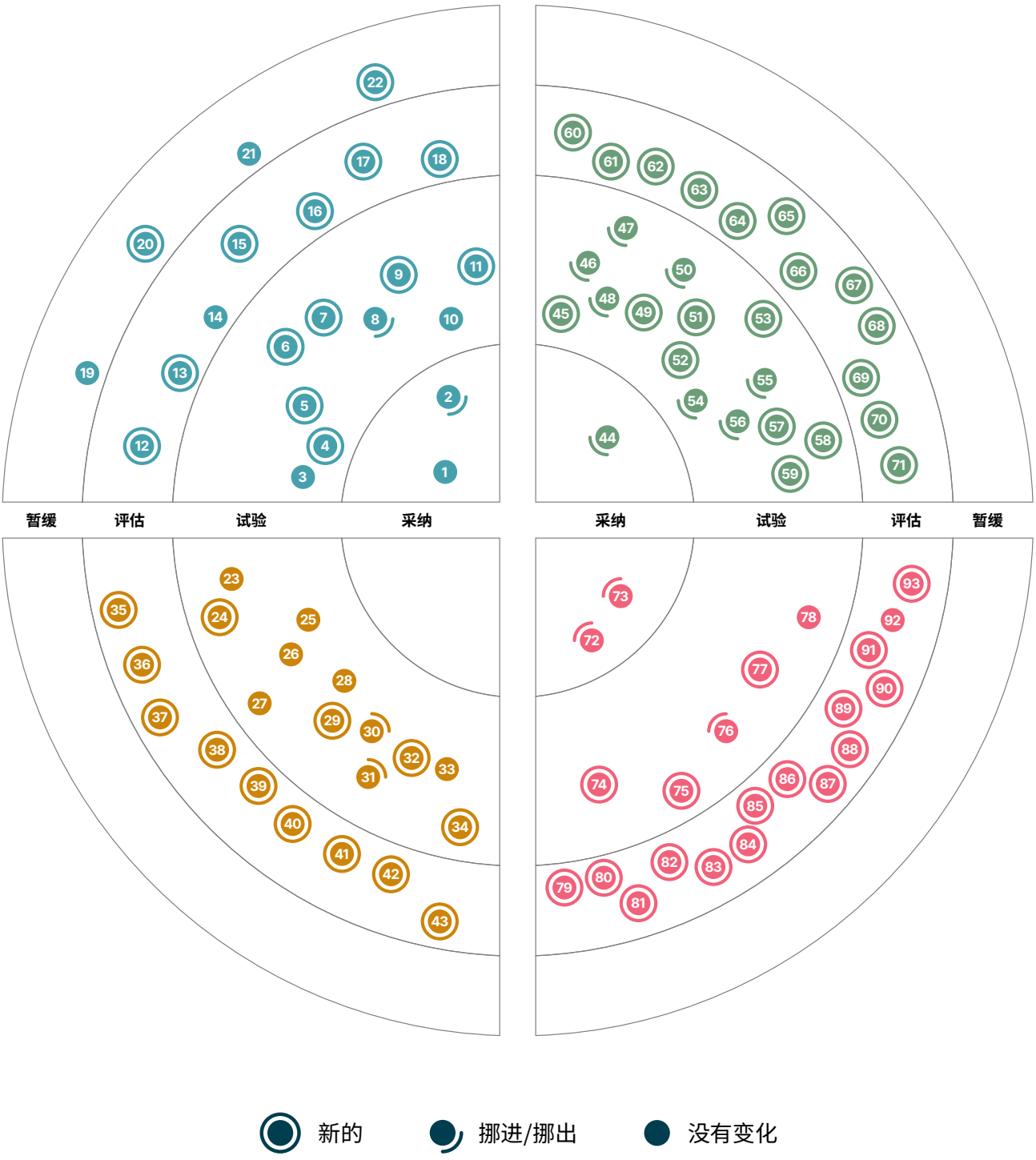
蓬勃发展的框架似乎成为了技术雷达中一种常见的模式：一个基础的框架变得流行起来，紧接大量工具的浮现创建出一个针对常见缺陷和改进的生态系统，最后以几个流行工具的巩固而结束。然而，React 的状态管理似乎在抵制这种通用的趋势。自从 Redux 发布后，我们看到源源不断的工具和框架以略微不同的方式来管理状态，每一种都有不同的权衡取舍。我们不知道原因，我们只能推测：这是 JavaScript 生态系统似乎提倡的自然流失吗？或者是 React 存在的一个有意思且看似易于修复的潜在缺陷，鼓励着开发人员多做实验吗？还是文档阅读格式（浏览器）与在它之上托管的应用程序所需要的交互性（和状态）之间存在永远无法匹配的障碍？我们不知道原因，但是我们期待下一轮的尝试，来解决这个看似永恒的问题。

对主数据编目永无止境的追求

从企业数据资产中挖掘更多价值的愿望，驱动了我们目前所看到的在数字技术方面的大部分投资。这项工作的核心通常集中在发现并访问所有相关数据上。几乎只要企业在收集数字数据，它就一直在努力将其合理化并编目到一个单一的、自上而下的企业目录。然而一次又一次，这一直觉上吸引人的概念与大型组织中固有的复杂性、冗余性和模糊性的残酷现实背道而驰。最近，我们注意到人们对企业数据编目重新产生了兴趣，以及大量相关巧妙新工具的雷达条目提议，例如 **Collibra** 和 **DataHub**。这些工具能够对跨仓库数据谱系和元数据提供一致的、可发现的访问，但它们不断扩展的功能集也延伸到了数据治理、质量管理、发布等方面。

与这种趋势相反，远离自上而下的中心化数据管理，走向基于数据网格架构的联合治理和数据发现的趋势也在增长。这种方法对期望和标准进行集中设置，而数据管理则根据业务领域线来划分，解决了企业数据的固有复杂性。面向领域的数据产品团队控制并分享自己的元数据，包括可发现性、质量和其它信息。此时，编目只是一种为搜索和浏览而展示信息的方式。生成的数据编目更加简单且更易维护，从而减少了对功能丰富的编目和治理平台的需求。

The Radar



The Radar

技术

采纳

1. 交付核心四指标
2. 统一远程团队墙

试验

3. 数据网格
4. 生产就绪的定义
5. 文档象限
6. 重新思考远程站会
7. 服务器端驱动 UI
8. 软件物料清单
9. 策略性复刻
10. 团队的认知负载
11. 过渡架构

评估

12. CUPID
13. 包容性设计
14. 非集群资源的 Operator 模式
15. 无边车服务网格
16. SLSA
17. 流式数据仓库
18. TinyML

暂缓

19. 面向编排的 Azure Data Factory
20. 混杂平台团队
21. 测试环境中的生产数据
22. 默认选择SPA

平台

采纳

—

试验

23. Azure DevOps
24. Azure Pipeline模板
25. CircleCI
26. Couchbase
27. eBPF
28. GitHub Actions
29. GitLab CI/CD
30. Google BigQuery ML
31. Google Cloud Dataflow
32. Github Actions 中的可复用工作流
33. Sealed Secrets
34. VerneMQ

评估

35. actions-runner-controller
36. Apache Iceberg
37. Blueboat
38. Cloudflare Pages
39. Colima
40. Collibra
41. CycloneDX
42. Embeddinghub
43. Temporal

暂缓

—

The Radar

工具

采纳
44. tfsec

试验
45. AKHQ
46. cert-manager
47. 云服务的碳足迹
48. Conftest
49. kube-score
50. Lighthouse
51. Metaflow
52. Micrometer
53. NUKE
54. Pactflow
55. Podman
56. Sourcegraph
57. Syft
58. Volta
59. Web Test Runner

评估
60. CDKTF
61. Chrome Recorder panel
62. Excalidraw
63. GitHub Codespaces
64. GoReleaser
65. Grype
66. Infracost
67. jc
68. skopeo
69. SQLFluff
70. Terraform Validator
71. Typesense

暂缓
—

语言和框架

采纳
72. SwiftUI
73. Testcontainers

试验
74. Bob
75. Flutter-Unity widget
76. Kotest
77. Swift 包管理器
78. Vowpal Wabbit

评估
79. Android Gradle 插件 — Kotlin DSL
80. Azure Bicep
81. Capacitor
82. Java 17
83. Jetpack Glance
84. Jetpack Media3
85. MistQL
86. npm工作区
87. Remix
88. ShedLock
89. SpiceDB
90. sqlc
91. 可组合架构
92. WebAssembly
93. Zig

暂缓
—

技术

采纳

1. 交付核心四指标
2. 统一远程团队墙

试验

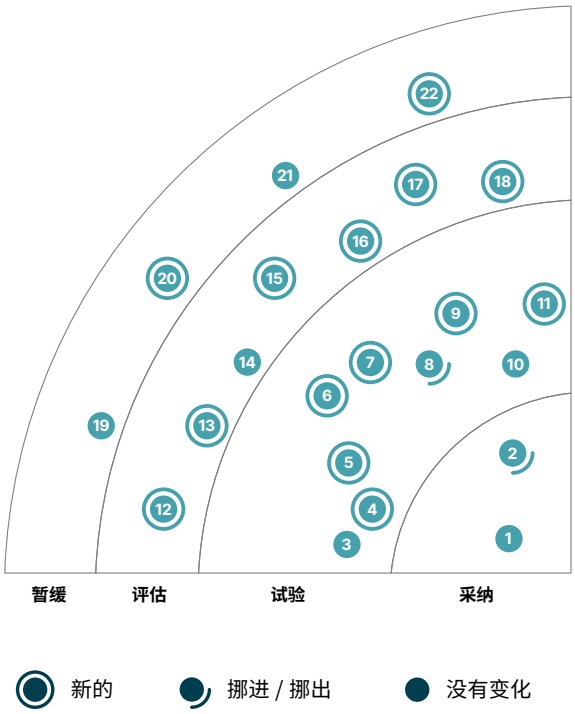
3. 数据网格
4. 生产就绪的定义
5. 文档象限
6. 重新思考远程站会
7. 服务器端驱动 UI
8. 软件物料清单
9. 策略性复刻
10. 团队的认知负载
11. 过渡架构

评估

12. CUPID
13. 包容性设计
14. 非集群资源的 Operator 模式
15. 无边车服务网格
16. SLSA
17. 流式数据仓库
18. TinyML

暂缓

19. 面向编排的 Azure Data Factory
20. 混杂平台团队
21. 测试环境中的生产数据
22. 默认选择SPA



1. 交付核心四指标

采纳

为了度量软件交付的效能，越来越多的组织默认采用由 DORA 研究项目定义的交付核心四指标，即：更改前置时间、部署频率、平均恢复时间（MTTR）和变更失败率。这项研究及其统计分析展示了高效能交付团队和这些指标的高度相关性，它们为衡量整个交付组织的表现提供了极佳的领先指标。

虽然我们依然是这些指标的坚定拥护者，但我们也吸取了一些教训。我们持续看到被误导的度量方式，这些方式使用的工具单纯基于持续交付（CD）流水线。尤其在衡量稳定性指标（MTTR 和变更失败率）时，仅依赖 CD 流水线数据提供的信息并不足以确定部署失败对实际用户的影响。只有包含真实事故，如用户服务降级，稳定性指标才有意义。

我们建议始终牢记度量指标背后的终极目的，并使用它们来思考和学习。比如，在花费数周时间构建复杂仪表板工具之前，可以考虑定期在团队回顾会议上进行 DORA 快速检查。这能够使团队有机会思考他们应该在哪些能力上努力以提升他们的指标，这会远比过于详细的开箱即用工具更有效。需要牢记这四个核心指标源于对高效能交付团队的组织级研究，在团队级别使用这些指标应该作为团队反思自身行为的一种方式，而不仅仅是添加到仪表板上的另一组度量标准。

2. 统一远程团队墙

采纳

统一远程团队墙是一项简单的技术，可以用虚拟的方式重新引入团队墙实践。我们建议分布式工作的团队采用这种方式；我们从迁移到远程工作的团队那里听到的一件事是，他们怀念使用实体团队墙的时候。这是一个能统一显示各种故事卡、任务、状态和进度的地方，在团队中充当信息辐射器和信息枢纽的角色。但是这种墙仅仅是作为一个集成点存在，实际的数据存储在各种不同的系统中。随着团队越来越远程化，他们不得不回归到各个信息源系统中去查看所需信息，很难再通过团队墙一目了然地了解项目。尽管保持团队墙的最新状态可能会需要一些代价，但我们还是认为这些都是值得去做的。对于一些团队来说，更新实体墙是团队日常“仪式”的一部分，所以远程墙也可以像这样去做。

3. 数据网格

试验

数据网格是一种面向分析和机器学习的技术方法，以去中心化的组织和技术方式分享、访问和管理数据。当前数据组织架构日渐复杂，数据使用案例激增以及数据源愈加多样化，面对当前的数据环境，数据网格希望创建一种社会技术方法，旨在规模化的获取数据中的价值。从本质上讲，它创建了一个可靠的数据共享模式，与组织同步发展并持续拥抱变化。据我们所知，业界对数据网格的兴趣正在显著增长。这种方法已经激发了许多组织去拥抱它所提倡的内容，技术供应商也使用现有技术为数据网格找到了新的部署方式。尽管业界对数据网格颇有兴趣，并且也越来越有经验，但是其实施却面临着高昂的整合成本。此外，它所提倡的内容仍旧局限于那些依据数据网格的硬性社会层面进行组织拆分的大型机构以及技术供应商——去中心化的数据所有权和联合治理的运作模式。

这些想法在 Data Mesh: Delivering Data-Driven Value at Scale 一书中被探讨，用于指导从业人员、架构师、技术领导以及决策者从传统大数据架构到数据网格的转型。该书完整地介绍了数据网格的原理及其构成要素；涵盖了如何设计数据网格架构、指导和执行数据网格战略以及引导组织设计向去中心化的数据所有权模式发展。本书的目标是为更深层次的对话创造一个新的框架，并引导数据网格到成熟的下一阶段。

4. 生产就绪的定义

试验

在一个实践“谁构建，谁运行”原则的组织中，生产就绪的定义（definition of production readiness DPR）是一个可以支持团队评估和准备新服务上线运营就绪情况的有用技术。DPR 以清单或者模板的形式实现，为团队在新服务投入生产前应该考虑的内容提供指导。尽管 DPR 没有定义要实现的特定服务级别的目标（service-level objectives SLOs）（这些目标很难被一刀切地定义），但它们提醒团队要考虑哪些类别的 SLO，要遵守哪些组织标准，还有需要哪些文档。DPR 提供了一个输入源，团队将这个输入源转换为各自产品特定的需求，以注入到他们的产品待开发项中例如可观察性和可靠性。

DPR 和 Google 的生产就绪审查 production readiness review (PRR) 密切相关。因组织太小而无法拥有专门的网站可靠性工程团队，或者是担心评审过程可能会给团队上线的流程带来负面影响时，DPR 至少可以为组织提供一些指导，并记录商定好的准则。对于非常关键的新服务，可以在需要时通过 PRR 来添加额外的审查来实现 DPR。

5. 文档象限

试验

在软件开发中，撰写良好的文档总是被忽视的一环，它经常被留到最后被随意地完成。我们的团队发现文档象限是一种确保生成正确文档的便捷方式。该技术沿两个轴对文档进行分类: 第一个轴与文档信息的性质有关——实践的或是理论的; 第二个轴描述了文档的使用情境——学习或是工作。这样定义的四个象限中就可以放置和帮助人们理解诸如教程、操作指南或参考页面之类的文档。该分类系统不仅可以确保关键文档不会被遗漏，而且还可以指导内容的呈现方式。我们发现这十分有助于创建入门文档，可以让开发人员在加入新团队时快速上手。

6. 重新思考远程站会

试验

站会一词源于在每日同步会议期间站起来的想法，目的是缩短会议时间。许多团队在他们的站会中试图遵守的共同原则是：保持简洁扼要。但我们现在看到一些团队挑战这一原则并重新思考远程站会。在线下协作时，在一天的剩余时间里有很多自发进行相互同步的机会，以此作为短暂站会的补充。而在远程办公时，我们的一些团队正在尝试一种更长的会议形式，类似于 Honeycomb 的成员所说的“漫谈式团队同步”。这不是要完全摆脱日常同步。我们仍然认为它非常重要和有价值，尤其对于远程的组织方式而言。相反，我们想把日常同步会议的时间延长至一个小时，用以淘汰一部分团队会议，并使团队联系更紧密。会议中仍然可以包括常用的团队

看板审阅，但是也可以加入更细节的澄清讨论，快速决策以及社交时间。只要它可以减少整体会议负担并提升团队凝聚力，就是成功的技巧。

7. 服务器端驱动 UI

试验

当汇总新一期技术雷达的时候，我们时常被一种似曾相识的感觉所倾倒。随着开发框架的涌现，服务器端驱动 UI 技术引发了一个热议话题。这项技术允许移动端开发者利用更快的变更周期，而不违反应用商店关于重新验证移动应用的任何政策。我们在之前的雷达中从赋能移动开发跨团队扩展的角度介绍过这项技术。服务器端驱动 UI 将渲染分离到移动应用程序的一个通用容器中，而每个视图的结构和数据由服务器提供。这意味着对于过去那些需要经历一次应用商店发布之旅的修改，现在只需要简单改变服务器发送的响应数据即可实现。需要说明的是，我们不推荐对所有 UI 开发都使用这种方式。我们的确陷入过一些可怕的过度配置的困境，但是在 AirBnB 和 Lyft 这样巨头的背书下，我们怀疑不只是我们 Thoughtworks 厌倦了一切交给客户端。这一领域值得关注。

8. 软件物料清单

试验

在保障系统安全性的压力不变且总体安全威胁不减的情况下，一个机器可读的软件物料清单（SBOM）可以帮助团队掌握他们所依赖的库中的安全问题。最近的 Log4Shell 零日远程漏洞十分严重且影响广泛。如果团队准备好了一个 SBOM，它就可以被扫描并被快速修复。现在我们已经具备了在项目中使用 SBOM 的生产经验，项目范围从小型企业到大型跨国公司，甚至是政府部门，而且我们确信它能为我们提供益处。一些工具（例如：Syft），能够让使用 SBOM 进行漏洞检测变得容易。

9. 策略性复刻

试验

策略性复刻是一种有助于代码库重组或从单体代码库迁移到微服务的技术。具体来说，更常见的方式是首先对代码库完全模块化，但在很多情况下会花费很长时间或者很难实现。而策略性复刻则为之提供了一种可能的替代方案。通过这种技术，团队可以创建代码库的新分支，并使用它来处理和抽取某个特定的关注点或领域，同时删除不需要的代码。这种技术的应用可能只是整个单体的长期计划的一部分。

10. 团队的认知负载

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。团队的交互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度量这些交互。我们使用高效能团队模式(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载，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。

11. 过渡架构

试验

过渡架构（transitional architecture）在替换遗留系统时是一种有用的做法。就像在建筑物建造或翻新过程中可能会建造、重新配置并最终拆除脚手架一样，在遗留系统迁移中也经常会需要临时架构步骤。尽管过渡架构最终会被移除或替换，但它们并不是一种浪费，这是因为过渡架构可以帮助降低风险，并可以将困难问题的解决分解为较小的步骤。这些较小的临时步骤很难构成一个完善的架构，因此避开了使用“大爆炸”方式替换遗留系统的陷阱。需要注意的是，替换结束后要记得移除架构的“脚手架”以避免它在以后成为技术债。

12. CUPID

评估

你应该如何编写好的代码？如何判断自己是否写了好的代码？作为软件开发者，我们总是在寻找一些自然易记的规则、原则和模式，以便在讨论如何编写简单的、易修改的代码时，我们有统一的语言和价值观。

Daniel Terhorst-North 最近尝试为好代码创建了一个类似于检查表的东西。他认为与其拘泥于像 SOLID 这样一套规则，不如使用一组特性作为目标。他设计出了名为 CUPID 的特性组，来描述为了写出“令人愉悦”的代码，我们需要做出哪些努力: 在该特性指导下的代码应该是可组合的，遵循 Unix 哲学的，可预测的，风格自然的以及基于领域的。

13. 包容性设计

评估

我们建议各类组织评估包容性设计，以确保自己将无障碍性视为头等需求。因为大家常常直到软件发布之前，甚至是发布之后，才想起与无障碍性和包容性相关的要求。满足这些要求的成本最低也是最简单的方法，就是将它们完全纳入开发过程并同时向团队提供早期反馈。在过去，我们强调了针对安全性和跨功能需求实施“左移”的技术；关于这种技术的一种观点认为，它与实现无障碍性目标一致。

14. 非集群资源的 Operator 模式

评估

除了管理部署在集群上的应用程序，我们看到 Kubernetes Operator 模式越来越多地用于其他地方。非集群资源的 Operator 模式可以利用自定义资源和在 Kubernetes 控制面板中实现的事件驱动调度机制，来管理与集群外部相关的活动。该技术建立在由 Kube 管理的云服务的思想之上，并将其扩展到其他活动中，例如持续部署或者及时响应外部存储库的变化。与专门构建的工具相比，这种技术的一个优势就是它开辟了一系列的工具，这些工具有的是 Kubernetes 自带的，有的则来自更广泛的生态社区。您可以使用 diff、dry-run 或 apply 等命令与 Operator 的自定义资源进行交互。Kube 的调度机制消除了以正确顺序编排活动的必要性，从而使开发更容易。如 Crossplane、Flux 和 Argo CD 等开源工具都利用了这项技术。随着时间的推移，我们希望看到更

多这样的工具出现。我们还观察到的一个现象，虽然每个工具都有自己的适用场景，但它们不可避免的会被误用或过度使用。对此我们有个“老生常谈”：一项工具可以应用在某个场景，并不表示它就应当应用在这个场景。在确认简单的传统方法不适用之前，请不要创建自定义资源定义，因为这会导致额外的复杂性。

15. 无边车服务网格

评估

服务网格的通常实现形式为与每个服务实例一并部署的反向代理进程，即“边车（Sidecar）”。尽管这些“边车”属于轻量级进程，但每个新服务实例的创建都意味着一个“边车”的新增，采用服务网格的整体开销和运维复杂度也会随之增加。然而，随着 eBPF 的发展，我们发现一种被称为无边车服务网格的模式能够将网格的功能安全地下沉到操作系统内核层，从而使得相同节点上的服务可以通过套接字透明地通信，而无需额外的代理。您可以通过 Cilium 服务网格对上述模式进行尝试，并从“每个服务一个代理”的部署模式简化为“每个节点一个代理”。我们对 eBPF 的能力十分感兴趣，并发现这一服务网格的演进十分重要且值得评估。

16. SLSA

评估

随着软件复杂性的不断增加，软件依赖项的威胁路径变得越来越难以守护。最近的 Log4J 漏洞表明了解这些依赖关系有多困难——许多没有直接使用 Log4J 的公司在不知不觉中就变得脆弱，因其生态系统中的其他软件依赖于 Log4J。软件工件供应链层级，又称 SLSA（读作“salsa”），是一个由联盟组织策划的，为组织机构提供防范供应链攻击的指南集。该框架衍生于一个 Google 多年来一直使用的内部指南。值得称赞的是，SLSA 并没有承诺提供“银弹”，即仅使用工具确保供应链安全的方法，而是提供了一个基于成熟度模型的具体威胁和实践的清单。这威胁模型是很容易理解的，其中包含了真实世界发生的攻击实例，并且要求文档中也提供了指南，帮助组织基于日渐增强的稳健性水平为其行动措施排定优先级，以改善他们供应链的安全态势。我们认为 SLSA 提供了适用的建议，并期待更多组织机构从中学习。

17. 流式数据仓库

评估

更快地响应客户洞察的需求助推了越来越多事件驱动架构和流式处理技术的采用。例如 Spark、Flink、Kafka Streams 等框架提供了一种范式，让简单事件的消费者和生产者可以在复杂的网络中合作，提供实时的数据洞见。但是这种编程风格需要投入时间和精力去掌握，并且当作为单点应用实现时，会缺乏互通性。这也使得流处理技术的大规模广泛应用需要大量的工程投资。如今，一大批新工具正崭露头角，为日渐庞大的使用 SQL 进行熟练分析的开发者群体提供流处理应用方面的帮助。同时，SQL 作为通用流式处理语言的标准化也降低了实现流数据应用的门槛。另外，还有一些工具，如 ksqlDB 和 Materialize 有助于将这些独立的应用整合为统一的平台。总而言之，企业中这些基于 SQL 的流处理应用集合可以称为流式数据仓库。

18. TinyML

评估

时至今日，在人们眼中，运行机器学习（ML）模型仍然需要高昂的计算成本，并且在某些情况下还需要使用专用硬件。虽然模型的创建仍然大致属于上述情况，但可以通过一种方式创建模型，使它们能够在小型、低成本和低功耗设备上运行。这种技术被称作 TinyML，它为在看上去不可行的情况下运行 ML 模型开辟了新的可能性。例如，在由电池供电的设备上，或者在受限或不稳定的网络环境中，TinyML 能够使模型运行在本地且不需要高昂的成本。如果你一直在考虑使用 ML 但苦于计算能力或网络环境的限制而放弃，那么这种技术值得你去评估。

19. 面向编排的 Azure Data Factory

暂缓

目前，Azure Data Factory 是以 Azure 为主要云供应商的组织用来编排数据处理流水线的默认产品。它支持数据提取、在使用不同存储类型的自有产品或者 Azure 服务之间复制数据，以及执行数据转换逻辑。尽管我们已有足够的经验使用 Azure Data Factory 对简单的自有服务数据进行云迁移，但我们不鼓励使用 Azure Data Factory 来编排复杂的数据处理流水线和工作流。对于使用 Azure Data Factory 进行系统间的数据迁移，我们已经获得了一些成功案例，但对于更复杂的数据流水线，它仍然存在一些挑战——可调试性和错误报告都不尽人意；可观察性也有限（因为 Azure Data Factory 日志记录功能未与 Azure Data Lake Storage 或 Databricks 等其他产品集成，因此难以实现端到端的观察）；以及只有部分地区能够使用数据源触发机制等。因此，目前我们推荐使用其他开源编排工具（例如 Airflow）来处理复杂的数据流水线，而仅使用 Azure Data Factory 进行数据复制或数据快照。我们的团队会继续使用 Azure Data Factory 来移动和提取数据，但对于更庞杂的操作，我们推荐其他功能更全面的工作流工具。

20. 混杂平台团队

暂缓

平台工程产品团队帮助交付团队自行完成部署和运营，缩短交付周期，降低整体复杂性，因此我们在之前的技术雷达中将它分类为采纳状态，认为它有益于内部平台团队的运营。但不幸的是，我们也看到，那些没有清晰产出或者没有明确用户群的团队也被打上了“平台团队”的标签。于是这些团队不得不面对一堆混杂且毫不相干的系统，在高强度但是优先级不明确的工作中勉强完成交付，变成了所谓的混杂平台团队。它们实际上变相的成为了另一个通用支持团队，解决那些其他团队不适合或者不愿意做的事情。我们仍然坚信，平台工程产品团队只有关注那些清晰并且定义明确的（内部）产品，才能交付更好的产出。

21. 测试环境中的生产数据

暂缓

我们一直认为测试环境中的生产数据是值得关注的领域。首先，它引发了许多最终导致了声誉受损的案例，例如从测试系统向整个客户群发送了不正确的警报。其次，测试系统的安全级别往往较低，尤其是围绕隐私数据的保护。当每个开发和测试人员都可以访问测试数据库中的生产数据副本时，对生产数据访问的精心控制就失

去意义了。尽管您可以混淆数据，但这往往仅适用于特定字段，例如信用卡号。最后一点，当从不同国家或地区托管或访问测试系统时，将生产数据复制到测试系统可能会违反隐私法，这在复杂的云部署中尤其麻烦。为解决这些问题，使用假数据是一种更安全的策略。现存的工具也能帮我们创建假数据。在某些场景，例如重现错误或训练特定的机器学习模型时，我们承认确实有复制生产数据特定元素的必要。但我们建议此时一定要谨慎行事。

22. 默认选择 SPA

暂缓

通常来说，我们会避免将建议过于浅显的条目放在暂缓状态中，包括那些盲目地遵循一种架构风格却没有注意权衡利弊的条目。然而，团队搭建网站时会默认选择单页面应用（SPA）的普遍现象让我们担心人们甚至没意识到 SPA 原本只是一种架构风格时，就立即进行了项目的框架选型。SPA 会招致传统基于服务器的网站所不具备的复杂性：譬如搜索引擎优化，浏览历史管理，网站分析，首页加载时间等。这些复杂性通常是为了确保用户体验，并且工具的持续发展也使得这些问题更容易解决（尽管 React 社区有关于状态管理的混乱透露出想要得到一个普适的解决方案是多么的困难）。然而，我们通常看不到团队去进行这种权衡分析，即使是在业务需求不能证明这种使用是合理的情况下，也盲目地接受了默认选择 SPA 的复杂性。事实上，我们已经开始注意到许多新的开发人员甚至都不知道有替代的方法，因为他们整个职业生涯都是在类似 React 这样的框架中度过的。我们相信，许多网站都会受益于服务端逻辑的简洁性，并且我们从例如 **Hotwire** 这种有助于减少用户体验差异的技术中受到了鼓励。

平台

采纳

—

试验

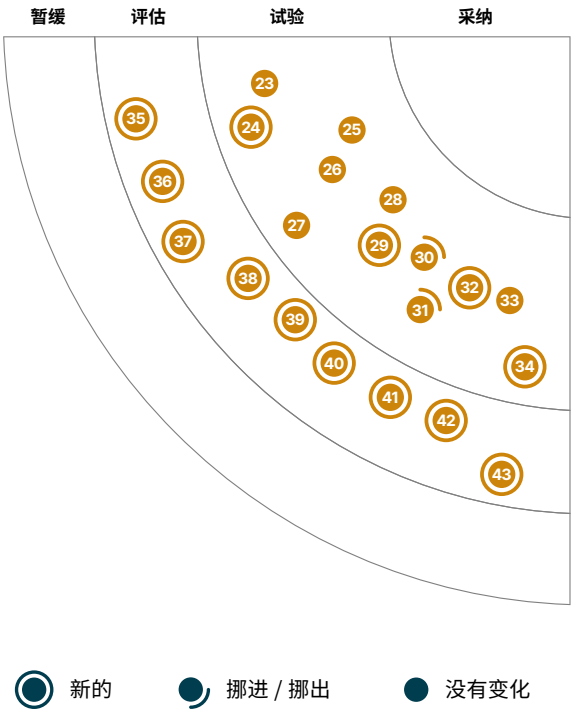
23. Azure DevOps
24. Azure Pipeline模板
25. CircleCI
26. Couchbase
27. eBPF
28. GitHub Actions
29. GitLab CI/CD
30. Google BigQuery ML
31. Google Cloud Dataflow
32. Github Actions 中的可复用工作流
33. Sealed Secrets
34. VerneMQ

评估

35. actions-runner-controller
36. Apache Iceberg
37. Blueboat
38. Cloudflare Pages
39. Colima
40. Collibra
41. CycloneDX
42. Embeddinghub
43. Temporal

暂缓

—



23. Azure DevOps

试验

随着 Azure DevOps 生态系统的不断发展，我们的团队正在更多的使用它，并取得了成功。这些服务包含一组托管服务，包括托管 Git 代码仓库、构建和部署流水线、自动化测试工具、待办工作管理工具和构件仓库。我们已经看到我们的团队在使用该平台时获得了良好的体验，这意味着 Azure DevOps 正在走向成熟。我们特别喜欢它的灵活性；它甚至允许用户使用来自不同供应商的服务。例如，你可以在使用 Azure DevOps 的流水线服务的同时也使用一个外部 Git 数据仓库。我们的团队对 Azure DevOps Pipelines 尤其感到激动。随着生态系统的成熟，我们发现越来越多的团队已经加入了 Azure 技术栈，因为它可以轻松地与微软世界的其他部分集成。

24. Azure Pipeline 模板

试验

Azure Pipeline 模板提供了两种机制来对 Azure Pipeline 定义去重。通过“includes”模板，你可以引用一个模板使其像参数化的 C++ 宏一样内联展开，从而以一种简单的方式将各个阶段、任务和步骤的公共配置分解出来。通过“extends”模板，你可以定义一个具有公共流水线配置的外壳，结合所需模板检查机制，如果流水线没有扩展特定的模板，你可以拒绝构建以防止对流水线配置本身的恶意攻击。Azure Pipeline 模板、CircleCI Orbs 以及刚崭露头角的 GitHub Actions 可复用工作流，是流水线设计模块化趋势在不同平台上的体现，我们从多个团队收到了好的反馈。

25. CircleCI

试验

我们的许多团队选择 CircleCI 来满足他们的持续集成需求，他们很欣赏它高效运行复杂流水线的能力。CircleCI 的开发人员在持续增加 CircleCI 的新功能，现在是 3.0 版本。我们的团队认为其中 Orbs 和 executors 非常有用。Orbs 是可重复使用的代码片段，可用来自动化重复的流程，进而加快项目的配置，并使其易于与第三方工具集成。多种多样的 executor 为在 Docker、Linux、macOS 或 Windows 虚拟机中配置作业提供了灵活性。

26. Couchbase

试验

当我们最初在 2013 年收录 Couchbase 时，它主要被视为是一个由 Membase 和 CouchDB 合并而来的持久化缓存。从那时起，它经历了持续的的改进，一个由相关工具以及商业产品组成的生态系统也围绕着它成长了起来。它新增的产品套件包括 Couchbase Mobile 和 Couchbase Sync Gateway。这些功能协同工作，即使在设备由于网络不稳定而离线的时间段内也能够使数据保持最新。随着这些设备的激增，我们看到了市场对嵌入式数据持久化的需求不断增加，无论设备是否被连接，这种数据的持久化都能够保证设备继续工作。近期，我们的一个团队对 Couchbase 的离线同步能力进行了评估，发现这种现成的功能为他们节省了大量的精力，避免了投入成本自己去构建这种离线同步能力。

27. eBPF

试验

近些年来，Linux 内核已经包括了扩展的伯克利数据包过滤器（eBPF），一个提供了将过滤器附加到特定套接字能力的虚拟机。但是，eBPF 远远超出了包过滤的范围，它允许在内核的不同点位上触发自定义脚本，而且开销非常小。虽然这项技术并不新鲜，但随着越来越多的微服务通过容器编排来部署，eBPF 逐渐自成一体。Kubernetes 和服务网格技术（如 Istio）被普遍使用，它们采用“边车”（sidecars）来实现控制功能。有了诸如 Bumblebee 这样使 eBPF 程序的构建、运行和发布变得更加容易的新工具，eBPF 可以被看作是传统边车的替代品。Cilium 的维护者甚至宣布了边车的消亡。基于 eBPF 的方法减少了一些由边车带来的性能和运维上的开销，但它不支持如本地终结 SSL 会话这样的常见功能。

28. GitHub Actions

试验

GitHub Actions 的使用量在去年大幅增长。之前的使用经历已经证明它可以处理更复杂的工作流程，并在复合操作中调用其他操作。但是，它仍存在一些缺点，例如无法重新触发工作流的单个作业。尽管 GitHub Marketplace 中的生态系统有其明显的优势，但让作为第三方的 GitHub Actions 访问你的构建流水线可能会以不安全的方式共享机密信息（我们建议遵循 GitHub 关于安全强化的建议）。尽管如此，GitHub Actions 以其在 GitHub 中的源代码旁直接创建构建工作流的便利性，结合使用 act 等开源工具在本地运行的能力，是一个利于团队刚开始开展工作以及新人上手的强有力选项。

29. GitLab CI/CD

试验

如果你正在使用 GitLab 管理软件交付，可以看看 GitLab CI/CD 能否满足持续集成和交付的需求。我们发现配合本地部署的 GitLab 以及自托管运行器时，GitLab CI/CD 尤其好用，因为这种组合可以解决使用基于云的解决方案经常会遇到的授权问题。自托管运行器可以完全根据需求进行配置，并安装合适的操作系统以及依赖项，因此流水线的运行速度比使用云供应的运行器要快得多，因为云供应的运行器每次都需要配置。

除了基本的构建、测试和部署流水线，GitLab 的产品还支持 Services、Auto Devops、ChatOps 以及其他高级功能。Services 十分适合将 Docker 服务（如 Postgres 或 Testcontainer）连接至用于集成测试与端到端测试的作业。Auto Devops 功能无需配置即可创建流水线，非常适用于刚开始进行持续交付的团队，以及有许多代码仓库的组织，可以避免手动创建许多流水线。

30. Google BigQuery ML

试验

自从雷达上次收录了 Google BigQuery ML 之后，通过连接到 TensorFlow 和 Vertex AI 作为后台，BigQuery ML 添加了如深度神经网络以及 AutoML Tables 等更复杂的模型。BigQuery 还引入了对时间序列预测的支持。之前我们关注一个问题是模型的可解释性。今年早些时候，BigQuery Explainable AI 被宣布为公众开放使用，在解决上述问题上迈出了一步。我们还可以将 BigQuery ML 模型作为 Tensorflow SavedModel 导出到 Cloud

Storage，并将它们用于在线预测。但仍有一些需要权衡的事情，例如是否需要降低“机器学习持续交付”的难易程度以使其低门槛好上手，BigQuery ML 仍然是一个有吸引力的选择，特别是当数据已经存储在 BigQuery 中的时候。

31. Google Cloud Dataflow

试验

Google Cloud Dataflow 是一个基于云平台的数据处理服务，适用于批量处理和实时流数据处理的应用。我们团队正在使用 Dataflow 来创建用于集成、准备和分析大数据集的数据处理流水线，在这之上使用 Apache Beam 的统一编程模型来方便管理。我们在 2018 年首次介绍了 Dataflow，它的稳定性、性能和丰富的功能让我们有信心在这一次的技术雷达中将它移动到试验环。

32. Github Actions 中的可复用工作流

试验

自从在两期技术雷达之前首次发布了 GitHub Actions 后，我们关注到业界对于 GitHub Actions 的兴趣在持续上升。GitHub 一直在持续迭代演进该产品，可复用工作流的发布解决了一些早期的缺陷。Github Actions 中的可复用工作流将流水线设计模块化，只要工作流依赖的代码仓库是 public 状态，你甚至可以跨代码仓库进行参数化复用。可复用工作流不但支持将机密值作为秘钥显示传递，也支持将输出结果传递给调用任务。无需离开 GitHub 这个平台，只需配置几行 YAML 文件，GitHub Actions 就能为你提供 CircleCI Orbs 或 Azure Pipeline Templates 在 CI 方面的灵活性。

33. Sealed Secrets

试验

Kubernetes 原生支持称为“机密”（secret）的键值对象。然而默认情况下，这些 Kubernetes 的“机密”其实并不真的那么机密。尽管它们与其他键值数据分开处理，可以单独采取预防措施或访问控制，且支持在将“机密”存储在 etcd 之前，对其进行加密，但在配置文件中，“机密”是以纯文本字段的形式保存的。Sealed Secrets 提供组合运算符和命令行实用程序，使用非对称密钥来对“机密”进行加密，以便仅在集群中用控制器将其解密。此过程可确保“机密”在 Kubernetes 用于部署的配置文件中不会泄漏。一旦加密，这些文件就可以安全地共享或与其他部署制品一起存储。

34. VerneMQ

试验

VerneMQ 是一个开源、高性能的分布式 MQTT 消息服务器。在之前的技术雷达中我们评估过一些 MQTT 消息服务器，比如 Mosquitto 和 EMQ。与它们类似，VerneMQ 也基于 Erlang/OTP 开发，具有高度可扩展性。它可以在硬件上水平和垂直扩展，以支持大量并发客户端的发布和订阅，同时保持低延迟和容错性。在我们的内部基准测试中，它已经能够帮助我们在单个集群中实现几百万个并发连接。它并不是新技术，我们在生产环境中使用了一段时间，目前运行良好。

35. actions-runner-controller

评估

actions-runner-controller 是一种 Kubernetes 控制器，它在 Kubernetes 集群上为 GitHub Actions 运行自托管运行器。这个工具可以在 Kubernetes 上创建一个运行器资源，它可以运行和操作自托管运行器。当你的 GitHub Actions 运行的作业需要访问 GitHub 云运行器主机无法访问的资源，或者依赖于某些特定的操作系统和环境而 GitHub 没有提供时，自托管运行器会很有帮助。当你有一个 Kubernetes 集群，你可以将自托管运行器作为一个 Kubernetes pod 运行，并根据 GitHub webhook 事件来伸缩。actions-controller-runner 具有轻量级和可伸缩的特性。

36. Apache Iceberg

评估

Apache Iceberg 是一个面向超大的分析数据集的开放表格格式。Iceberg 支持现代数据分析操作，如条目级的插入、更新、删除、时间旅行查询、ACID 事务、隐藏式分区和完整模式演化。它支持多种底层文件存储格式，如 Apache Parquet、Apache ORC 和 Apache Avro。已有许多数据处理引擎支持 Apache Iceberg，包括一些 SQL 引擎，如 Dremio 和 Trino，以及（结构化）流处理引擎，如 Apache Spark 和 Apache Flink。

37. Blueboat

评估

Blueboat 是一个无服务器 web 应用的多租户平台。它采用了被广泛使用的 V8 JavaScript 引擎，同时，出于安全和性能的考虑，它使用 Rust 原生地实现了常用的网络应用程序库。Blueboat 可以被看作是 CloudFlare Workers 或 Deno Deploy 的替代品，但与它们之间存在一个重要区别——你必须操作和管理底层基础设施。因此，我们建议你仔细评估 Blueboat 是否满足你的本地无服务器需求。

38. Cloudflare Pages

评估

当 Cloudflare Workers 发布的时候，我们着重介绍它是一个面向边缘计算的早期函数即服务（FaaS）方案，实现方案十分有趣。去年（2021 年）四月发布的 Cloudflare Pages 并没有特别引人注目，因为 Pages 只是众多基于 Git 的网站托管解决方案中的一个。虽然 Cloudflare Pages 的确有一个大多数替代方案不具备的有用功能——持续预览。不过，现在 Cloudflare 已经将 Workers 和 Pages 更紧密地集成了起来，创建了一个运行在 CDN 上的、完全集成的 JAMstack 解决方案。键值存储和高度一致的协调原语让新版 Cloudflare Pages 更具吸引力。

39. Colima

评估

Colima 正成为 Docker 桌面版的一个热门开放替代方案。它通过在 Lima VM 中配置 Docker 容器运行时环境，可以在 macOS 上配置 Docker CLI 并处理端口转发和挂载存储。Colima 使用 containerd 作为容器运行时，这

也是大多数托管 Kubernetes 服务采用的容器运行时（提升了开发与生产环境的一致性）。您可以基于 Colima 轻松地使用和测试 containerd 的最新特性，例如容器镜像的惰性加载。凭借其良好的性能，我们期待 Colima 成为 Docker 桌面版的强有力开源替代方案。

40. Collibra

评估

在日益拥挤的企业数据目录市场中，我们的团队很喜欢使用 Collibra。他们喜欢 SaaS 或自托管实例的部署灵活性，许多功能可以开箱即用，包括数据治理、数据血缘、数据质量和数据的可观测性。用户还可以选择使用仅需更小功能集合的更加去中心化的管理方法（如 Data mesh）。真正令人引以为傲的的是他们经常被忽略的客户支持，对此我们认为是一种协作与支持。诚然，简单的数据目录和更全功能的企业平台之间存在矛盾，但到目前为止，使用它的团队对使用 Collibra 支持其需求的方式非常满意。

41. CycloneDX

评估

CycloneDX 是一个用来描述机器可读的软件物料清单（SBOM）的标准。随着软件和计算架构日渐复杂，软件变得越来越难以定义。CycloneDX 起源于 OWASP，它对旧的 SPDX 标准进行了改进，提供了更广泛的定义，不仅包含了本地机器依赖，还包含运行时服务依赖。你还会发现它提供了一个用于集成的生态系统，包括多种编程语言的实现，以及允许你通过适当的签名和验证来分析和更改 SBOM 的命令行工具。

42. Embeddinghub

评估

Embeddinghub 是一个与 Milvus 十分类似的，面向机器学习嵌入领域的向量数据库。不同的是，它提供了开箱即用的近似最邻近运算、表分区、版本及访问控制等功能，我们建议你根据你的嵌入向量化场景对 Embeddinghub 进行评估。

43. Temporal

评估

Temporal 是一个用于开发长期运行工作流的平台，尤其适用于微服务架构。作为 Uber 开源项目（OOS）Cadence 的衍生项目，Temporal 对于长期运行的工作流采用了事件溯源（event-sourcing）模式，因此它们可以在进程或主机的崩溃后恢复。尽管我们不推荐在微服务架构中使用分布式事务，但如果你确实需要分布式事务或者长期运行的 Sagas，你或许会对 Temporal 有兴趣。

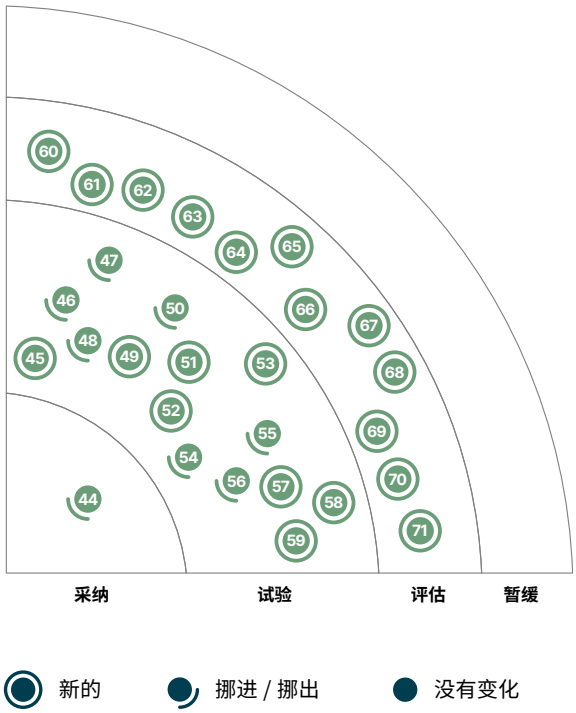
工具

采纳
44. tfsec

试验
45. AKHQ
46. cert-manager
47. 云服务的碳足迹
48. Conftest
49. kube-score
50. Lighthouse
51. Metaflow
52. Micrometer
53. NUKE
54. Pactflow
55. Podman
56. Sourcegraph
57. Syft
58. Volta
59. Web Test Runner

评估
60. CDKTF
61. Chrome Recorder panel
62. Excalidraw
63. GitHub Codespaces
64. GoReleaser
65. Grype
66. Infracost
67. jc
68. skopeo
69. SQLFluff
70. Terraform Validator
71. Typesense

暂缓
—



44. tfsec

采纳

对于那些我们正在使用 Terraform 的项目来说，在需要检测潜在安全风险时，tfsec 已经迅速成为默认的静态分析工具。它很容易被集成到 CI 流水线，而且拥有一个持续增长的检查库，可以用来检查所有主要的云供应商和诸如 Kunernetes 的平台。鉴于它的易用性，我们相信对任何 Terraform 项目而言，tfsec 都会是一个非常好的补充。

45. AKHQ

试验

AKHQ 是 Apache Kafka 的图形用户界面（GUI），可以帮助你管理主题、主题数据、消费者组等。我们的一些团队发现 AKHQ 是用来监控 Kafka 集群实时状态的有效工具。比如，你可以浏览集群上的主题，对于每个主题，你都能可视化它的名称、存储的消息数量、使用的磁盘空间、最后一条记录的时间、分区数、同步数量的复制因子和消费者组。通过 Avro 和 Protobuf 的反序列化，AKHQ 可以帮助你了解 Kafka 环境中的数据流。

46. cert-manager

试验

cert-manager 是一款在 Kubernetes 集群里管理 X.509 证书的工具。它将证书和签发者建模为一等资源类型，并将证书作为服务安全地提供给工作在 Kubernetes 集群上的开发人员和应用程序。在使用 Kubernetes 默认 ingress 控制器时，cert-manager 是个显而易见的选择，但也推荐用在其他的控制器上，尤其在你不应该手动管理自己的证书的时候。我们的几个团队一直在广泛使用 cert-manager，而且发现它的可用性在过去几个月里有了很大的提升。

47. 云服务的碳足迹

试验

Cloud Carbon Footprint（CCF）是一款通过云 API 来查看 AWS、GCP、Azure 云平台上碳排放的可视化工具。Thoughtworks 的团队已经成功使用这个工具与多个组织合作，其中包括能源科技公司、零售商、数字服务的供应商和使用人工智能的公司。云平台提供商意识到，帮助客户理解在使用云服务时产生的碳排放的影响是很重要的。所以他们开始自主构建类似的功能。因为 CCF 是独立于云架构的，它允许使用者在一个位置查看多个不同云服务商的能源使用和碳排放情况，同时将碳足迹转化为对现实世界的影响，比如排放量相当于多少次航班，或者多少棵树。在最近的发布中，CCF 已经开始包含针对 Google 云和 AWS 云上可能的节能与减少二氧化碳排放的优化建议，以及支持更多类型的云实例，比如 GPU。考虑到现在这个工具已经备受关注和持续增加新功能，我们对未来把它挪入试验状态充满信心。

48. Conftest

试验

Conftest 是一款针对结构化配置数据编写测试的工具。它依赖于开放策略代理中 Rego 语言，能够为 Kubernetes 配置、Tekton 的流水线定义、甚至 Terraform 计划编写测试。在我们的实际使用中，Conftest 的体验一直都非常棒，这也得益于它平缓的学习曲线。借助测试的快速反馈，我们的团队可以又快又安全地迭代变更 Kubernetes 的配置。

49. kube-score

试验

kube-score 是一款针对 Kubernetes 对象定义，进行代码静态检查的工具。它的输出是一份建议列表，里面包含了如何提升你的应用程序安全性及弹性的相关建议。它有一份包含了最佳实践的预定义检查，比如以非 root 权限运行容器，正确指定资源限制等。它已经存在了一段时间，我们在一些项目中将它作为 Kubernetes manifests 构建的 CD 流水线的一部分来使用。kube-score 的一个显著缺陷是你无法添加自定义策略。在这种情况下，我们使用像 Conftest 这样的工具，以弥补它的缺陷。

50. Lighthouse

试验

Lighthouse 是一个由 Google 编写的工具，可以评估 Web 应用和页面，以及从出色的开发实践中收集性能指标和洞见等信息。我们一直主张性能测试乃第一公民，五年前技术雷达中提到的对 Lighthouse 的补充内容对此也有帮助。我们关于适应度函数的思考，也为在构建流水线中运行 Lighthouse 这样的工具创造了强烈的动机。随着 Lighthouse CI 的引入，将 Lighthouse 纳入由不同工具管理的流水线，会变得比以往任何时候都容易。

51. Metaflow

试验

Metaflow 是一个对用户友好的 Python 库和后端服务，可以帮助数据科学家和工程师构建和管理可用于生产的数据处理、机器学习训练及推理的工作流。Metaflow 提供一系列 Python API，将代码组织为由步骤组成的有向图。每一个步骤都可以灵活配置，例如其所需的计算和存储资源。每个步骤执行（也就是任务）的代码和数据副本都被保存起来，并可以在今后的运行或流程的下一步中被检索出来，帮助你从错误中恢复，重新执行任务，还可以追踪模型的版本以及多个运行之间的依赖关系。

Metaflow 的价值主张是其惯用的 Python 库的简洁性：它与构建和运行时的基础设施完全集成，以支持在本地和规模化的生产环境中运行数据工程和科学任务。在撰写本条目时，Metaflow 和 AWS 服务高度集成，例如使用 S3 来做数据存储，step functions 来做编排。除 Python 以外，Metaflow 还支持 R 语言。其核心功能都是开源的。

如果你正在 AWS 上构建和部署生产环境的机器学习和数据处理流水线，作为一个轻量级的全栈框架，Metaflow 可以替代例如 MLflow 这类更复杂的平台。

52. Micrometer

试验

Micrometer 是一个跨平台的库，用于 JVM 的指标检测，支持 Graphite、New Relic、CloudWatch 和许多其他集成。Micrometer 让库作者和团队都受益：库作者可以在他们的库中包含指标检测代码，而无需支持库用户正在使用的每个指标系统；团队可以在后端注册表上支持许多不同的指标，这使组织能够以一致的方式收集指标。

53. NUKE

试验

NUKE 是一个面向 .NET 的构建系统，也是传统的 MSBuild、Cake 以及 Fake 等自动化构建系统的替代品，我们曾在之前的技术雷达中介绍过它们。NUKE 以 C# 领域特定语言（DSL）的形式表达构建指令，不但降低了学习成本，而且 IDE 支持性也很好。在我们的实际体验中，使用 NUKE 进行 .NET 项目的自动化构建十分便捷。我们喜欢 NUKE 提供的精准代码静态检查和提示功能，并且它支持无缝使用各种 NuGet 包，这样可以编译自动化代码，避免运行时发生错误。尽管 NUKE 已不是新技术，但它采用 C# DSL 的全新方法，以及使用 NUKE 时全方位的良好体验，促使我们一定要将它收录在技术雷达里。

54. Pactflow

试验

在长时间使用 Pact 进行契约测试的过程中，我们目睹了规模化带来的复杂性。我们的一些团队已经使用 Pactflow 成功减少了这种复杂性引发的后果。Pactflow 既可以作为 SaaS 运行，也可以部署在本地，并提供与 SaaS 产品相同的功能，它在开源产品 Pact Broker 的基础上，提升了可用性、安全性以及审计体验。到目前为止，我们很满意 Pactflow 的使用体验，并且很高兴看到它在持续致力于降低管理大规模契约测试所带来的开销。

55. Podman

试验

Podman 作为 Docker 的替代方案，已经通过我们许多团队的验证。与 Docker 不同的是，Podman 使用一个无守护引擎来管理和运行容器，这是一种有趣的方案。此外，Podman 可以以普通用户身份运行而无需 root 权限，从而减少了攻击面。通过使用 Buildah 构建的开放容器倡议（OCI）镜像或者 Docker 镜像，Podman 可以适用于大多数容器使用场景。除了与 macOS 的一些兼容性问题外，我们团队在 Linux 各发行版上使用 Podman 的总体感觉非常好。

56. Sourcegraph

试验

在往期的技术雷达中，我们介绍了两个基于抽象语法树（AST）表征的代码搜索和替换工具，Comby 和 Sourcegraph。它们尽管有一些相似之处，但也有一些不同的地方。Sourcegraph 是一个商业工具（也有最多支持 10 个用户的免费版本），特别适合在大型代码库中进行搜索、导航或交叉引用等操作，重视与开发者的交互体验。相比之下，Comby 是一个用于自动化重复性任务的轻量级开源命令行工具。由于 Sourcegraph 是一个托管服务，它能持续监测代码库，并在成功匹配时发出警报。现在我们对 Sourcegraph 有了更多的经验，决定将其挪到试验状态，以反映我们从中获得的良好体验——但这并不意味着 Sourcegraph 比 Comby 更好。每个工具都有各自专注的方向。

57. Syft

试验

使用软件物料清单（SBOM）是改善“供应链安全”的关键要素之一，因此在发布软件构件的同时，发布相应的 SBOM 正变得越来越重要。Syft 是一个致力于为容器镜像和文件系统生成 SBOM 的 CLI 工具和 Go 语言库。它可以生成包括 JSON，CycloneDX 和 SPDX 在内的多种格式的 SBOM。Syft 输出的 SBOM 可以被 Grype 用于漏洞扫描。使用 Cosign 将 SBOM 添加为证明文件，可以将生成的 SBOM 和镜像一起发布。这使得镜像的消费者可以对 SBOM 进行验证，并将其用于后续的分析。

58. Volta

试验

当同时在多个 JavaScript 代码库上工作时，我们往往需要使用不同版本的 Node 和其他 JavaScript 工具。在开发机器上，这些工具通常安装在用户目录或本机中，这意味着需要一个解决方案，帮助开发者在多个版本之中进行切换。对于 Node 而言，nvm 能够做到这一点，但我们想重点强调一个替代方案 Volta，我们的一些团队正在使用它。与使用 nvm 相比，Volta 有几个优点：它可以管理其他 JavaScript 工具，如 yarn；它还具备一个基于项目绑定工具链某个版本的理念，这意味着开发人员可以简单使用给定代码目录中的工具，而不必担心需要手动切换工具版本——Volta 是通过使用路径中的 shims 来选择被绑定的版本。Volta 采用 Rust 编写，速度极快，以独立二进制文件进行分发，没有任何依赖。

59. Web Test Runner

试验

Web Test Runner 是 Modern Web 项目中的一个套件，该项目为现代 Web 开发提供了若干高质量的工具，支持像 ES 模块之类的 Web 标准。Web Test Runner 是一个针对 Web 应用的测试运行器。与其他现有测试运行器相比，它的一个优势是可以在浏览器中运行测试（也可以无图形界面运行）。它支持多种浏览器启动器——包括 Puppeteer，Playwright 和 Selenium，并且使用 Mocha 作为默认测试框架。Web Test Runner 运行测试的速度非常快，我们很喜欢在调试的时候能打开一个带 devtools 的浏览器窗口。它在内部采用了 Web Dev Server，这意味着我们可以利用其出色的插件 API，为测试套件添加自定义插件。Modern Web 项目的工具看起来是一套非常有前景的开发者工具链，我们已经在一些项目中使用它。

60. CDKTF

评估

迄今为止，许多组织已经创造了广阔的云服务图景。当然，这只有在使用基础设施即代码和成熟的工具时才可能实现。我们仍然喜欢 Terraform，尤其是它丰富且日渐增长的生态系统。然而，Terraform 的默认配置语言 HCL 缺乏抽象性，导致了它的玻璃天花板。虽然使用 Terragrunt 缓解了这一点，但我们的团队越来越渴望像现代编程语言所能提供的那种抽象性。由 AWS CDK 团队和 Hashicorp 合作开发的 Terraform 云开发工具包 (CDKTF)，让团队有可能使用多种不同的编程语言，包括 TypeScript 和 Java，去定义并配置基础设施。通过这种方法，它在 Terraform 生态系统中紧跟 Pulumi 的领先地位。我们已经对 CDKTF 有了很好的经验，但仍然决定将其暂留在评估状态，直到它脱离 beta 版本。

61. Chrome Recorder panel

评估

Chrome Recorder panel 是 Google Chrome 97 的预览功能，允许简单地录制和回放用户旅程。虽然这绝对不是一个新想法，但它集成在 Chrome 浏览器中的方式能允许快速地创建、编辑和运行脚本。Chrome Recorder panel 也很好地集成了性能面板，这让获取重复、持续的页面性能反馈变得更加容易。虽然总是需要谨慎使用录制 / 回放风格的测试，以避免脆弱的测试，但我们认为这个预览功能值得评估，特别是如果你已经在使用 Chrome 性能面板来测量页面。

62. Excalidraw

评估

Excalidraw 是我们团队喜欢使用的简单但功能强大的绘图工具。有时候团队只是需要一张草图而不是正式的图表，Excalidraw 为远程团队提供了一种可以快速创建和共享图表的方式。我们团队也喜欢它生成的低保真图表样式，这让人联想到团队在同地协作时绘制的白板图表。提醒一点：你需要注意它默认的安全性，在你进行绘制时，任何拥有链接的人都可以看见图表。付费版本则提供了进一步的身份验证功能。

63. Github Codespace

评估

Github Codespace 允许开发者在云上创建开发环境，你可以通过 IDE 访问它，就像在本地环境一样。Github 不是第一家实现这个想法的公司，我们之前还提到过 Gitpod。我们喜欢 Codespace 允许通过使用 dotfiles 文件来标准化配置环境的功能，这能够帮助新团队成员更快上手；我们也十分中意 Codespace 能提供最高 32 核 64GB 内存虚拟机的特性，这些虚拟机可以在 10 秒钟内启动，有可能提供比开发笔记本电脑更强大的环境。

64. GoReleaser

评估

GoReleaser 是一个通过多个库和通道来支持不同架构的 Go 项目自动化构建和发布的工具，这是面向不同平台 Go 项目的常见需求。你可以在本地机器或者 CI 上运行该工具，它支持在多种 CI 服务上运行，从而最大限度降低安装和维护成本。GoReleaser 能够用于每个发布版本的构建、打包、发布和声明，并且支持不同的包格式、

包库和源代码控制的组合。虽然它已经出现好几年了，但我们惊讶并没有多少团队使用它。如果你经常发布 Go 代码库，这个工具值得一试。

65. Grype

评估

保证软件供应链的安全性已经得到交付团队的普遍关注，这种关注也反映在越来越多的新工具涌现在该领域中。Grype 就是一个新的针对 Docker 和 OCI 镜像进行漏洞扫描的轻量级工具。它可以以二进制文件安装，能在镜像被推至仓库前对其进行扫描，而且不需要在你的构建服务器上运行 Docker 守护进程。Grype 与 Syft 出自同一个团队，后者用于为容器镜像生成不同格式的软件物料清单。Grype 可以使用 Syft 输出的软件物料清单扫描安全漏洞。

66. Infracost

评估

迁移到云端的一个常被提及的优势是将基础设施开销透明化。但根据我们的经验，情况却往往相反。团队并不总是从财务成本的角度来考虑他们围绕基础设施所做的决定，这就是为什么我们之前提到了将运行成本实现为架构适应度函数。我们对一个名为 Infracost 的新工具感到好奇，该工具可以在 Terraform pull request 中可视化成本权衡。它是一个开源软件，在 macOS、Linux、Windows 和 Docker 均可访问，开箱即用支持 AWS、GCP 和微软 Azure 的定价。它还提供了一个公共 API，可以查询到当前的成本数据。我们的团队对它的潜力感到兴奋，特别是它还将支持在 IDE 中提供更好的成本可见性。

67. jc

评估

在之前的技术雷达中，我们将现代 Unix 命令放在了评估状态。在该工具集中，jq 命令实际上是一个支持 JSON 的 sed。而 jc 命令执行的是与之相关的任务：它获取常见 Unix 命令的输出，并将输出解析为 JSON。jq 和 jc 这两个命令一起为 Unix CLI 世界以及大量基于 JSON 工作的库和工具之间架起了一座桥梁。当编写一些像软件部署或者故障诊断信息收集的简单脚本时，将五花八门的 Unix 命令输出格式映射到定义明确的 JSON，可以为我们节省大量的时间和精力。与 jq 命令一样，你需要确保该命令可用。它可以通过许多著名的软件库进行安装。

68. skopeo

评估

skopeo 是一款可以对容器镜像和镜像仓库执行各种操作的命令行工具。它的大部分操作都不要求用户以 root 角色执行，也不需要运行守护进程。它是 CI 流水线中的实用部分，在推广镜像时，我们可以用 skopeo 把镜像从一个注册表拷贝到另一个注册表。这样的操作比直接拉取和推送镜像更好，因为我们不需要在本地存储这些镜像。skopeo 不是一个新工具，但它足够有用且未被充分认识到，所以我们认为它值得一提。

69. SQLFluff

评估

尽管代码静态检查已经是软件工程中的古老实践了，但它在数据领域中的应用仍十分缓慢。SQLFluff 是一个 python 实现的跨 SQL 方言的 linter，它提供了简单的命令行界面（CLI），可以很容易地整合进 CI/CD 流水线。如果默认配置就适合你，那么 SQLFluff 在安装后无需任何额外设定就可工作，它会强制执行一套鲜明风格的标准来格式化代码，当然，你也可以通过添加一个 dotfile 设定自己的代码规范。这个命令行工具还能自动修复诸如空格或者关键词大小写等违反代码规范设定的格式错误。SQLFluff 虽然还很年轻，但是 SQL 代码静态检查圈内获得更多关注是一件让人兴奋的事。

70. Terraform Validator

评估

一些已经采用了基础设施即代码和自服务基础设施平台的组织，正在寻找在执行良好安全实践和组织政策的同时，能给予团队最大限度自主权的方法。我们之前已经着重强调过 tfsec，并在这一期技术雷达中将它挪到了采纳中。对于使用谷歌云平台（GCP）的团队来说，可以使用 Terraform Validator 构建策略库，作为检查 Terraform 配置的约束条件。

71. Typesense

评估

Typesense 是一个快速、容错的文本搜索引擎。在有大量数据的情形下，Elasticsearch 可能仍然是一个不错的选择，因为它提供了一个基于磁盘且可横向扩展的搜索解决方案。然而如果你正在构建一个对延迟敏感的搜索应用，并且搜索索引的尺寸可以容纳在内存中，那么 Typesense 会是一个强大的替代方案，你也可以考虑与 Meilisearch 等工具一起评估。

语言和框架

采纳

72. SwiftUI
73. Testcontainers

试验

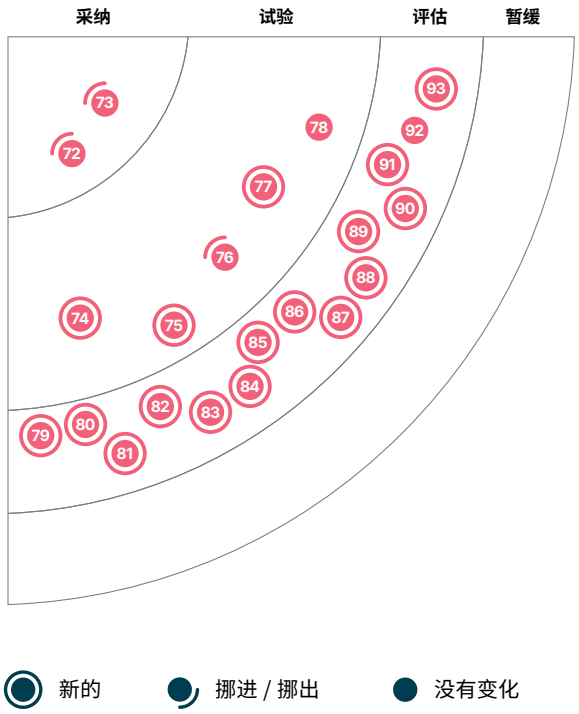
74. Bob
75. Flutter-Unity widget
76. Kotest
77. Swift 包管理器
78. Vowpal Wabbit

评估

79. Android Gradle 插件 — Kotlin DSL
80. Azure Bicep
81. Capacitor
82. Java 17
83. Jetpack Glance
84. Jetpack Media3
85. MistQL
86. npm工作区
87. Remix
88. ShedLock
89. SpiceDB
90. sqlc
91. 可组合架构
92. WebAssembly
93. Zig

暂缓

—



72. SwiftUI

采纳

对于在苹果生产的各种设备上实现用户界面来说，苹果在几年前推出 SwiftUI 是一个很大的进步。从一开始，我们就喜欢 Combine 提供的声明式的、以代码为中心的方法和反应式编程模型。但我们注意到，在苹果提供的 XCUITest 自动化框架下，仍需使用模型 — 视图 — 视图模型（MVVM）模式编写大量的视图测试，并不是非常合理。这个缺陷已经被 ViewInspector 所弥补。最后一个障碍是所需的最低操作系统版本。在发布时，只有最新版本的 iOS 和 macOS 可以运行用 SwiftUI 编写的应用程序，但由于苹果的定期更新，SwiftUI 应用程序现在几乎可以在所有接受安全更新的 macOS 和 iOS 版本上运行。

73. Testcontainers

采纳

根据长期使用 Testcontainers 的经验，我们认为它是创建可靠的环境来运行自动化测试的默认选项。Testcontainers 是一个拥有多种语言版本的库，并且 docker 化了常见的测试依赖——包括了不同种类的数据库，队列技术，云服务和 UI 测试依赖（例如 web 浏览器），还具有按需运行自定义 Dockerfile 的能力。它与类似 JUnit 的测试框架兼容，而且足够灵活，可以让用户管理容器的生命周期和高级网络，并迅速建立一个集成测试环境。我们的团队一直认为这个可编程的、轻量级的、一次性的容器库可以使功能测试更加可靠。

74. Bob

试验

在使用 React Native 构建应用时，有时你会发现不得不创建自己的模块。例如，我们在为 React Native 应用程序构建一个 UI 组件库时就遇到了这种需求。创建这样一个模块项目并不简单，但我们的团队成功地使用 Bob 来自动化实现了这一任务。Bob 提供了一个命令行界面来为不同的构建目标创建脚手架。这个脚手架并不限于核心功能，还可以选择性地包括示例代码、代码检查工具、构建流水线和其他功能。

75. Flutter-Unity widget

试验

Flutter 在构建跨平台移动应用方面越来越受欢迎，Unity 非常适合于构建增强现实（AR）和虚拟现实（VR）体验。而 Flutter-Unity widget 则是整合 Unity 和 Flutter 的一个关键组件。它允许开发者在 Flutter widget 内嵌入 Unity 应用。该插件提供的重要能力之一是能够提供 Flutter 和 Unity 之间的双向通信。我们发现它的性能也相当不错，我们期待在更多的 Flutter 应用中使用 Unity。

76. Kotest

试验

Kotest（原名 KotlinTest）是 Kotlin 生态中的一个独立测试工具，它在我们团队各式各样的 Kotlin 实现（原生、JVM 或 JavaScript）中越来越受到关注。Kotest 的主要优点在于它提供了丰富的测试风格来构建测试套件，其中还有一套全面的匹配器，可以帮助你使用优雅的内部领域专用语言（DSL）编写表达式测试用例。Kotest 除

了支持基于属性的测试（一项我们在以前的技术雷达中提到过的技术）之外，我们团队还看好它可靠的 IntelliJ 插件以及来自于社区的持续支持。

77. Swift 包管理器

试验

一些编程语言，尤其是较新的编程语言，内置了包和依赖管理解决方案。当 Swift 在 2014 年被推出的时候，它并没有附带包管理器，所以 macOS 和 iOS 开发者社区只能继续使用为 Objective-C 创建的第三方解决方案 CocoaPods 和 Carthage。几年后，Swift Package Manager（SwiftPM）作为一个苹果的官方开源项目被推出。那之后又过了几年，苹果才在 Xcode 中添加了对它的支持。尽管如此，在那时，许多开发团队仍在继续使用 CocoaPods 和 Carthage，主要是因为许多软件包根本无法通过 SwiftPM 获得。既然现在大多数包已经被添加在了 SwiftPM 中，并且对于包的创建者和使用者来说，流程都被进一步地简化了，我们的团队也自然地越来越依赖 SwiftPM。

78. Vowpal Wabbit

试验

Vowpal Wabbit 是一个多用途的机器学习库。Vowpal Wabbit 最初是雅虎研究院于十多年前创建的，如今它依然在持续实现新的强化学习算法。我们想要特别提及的是 Vowpal Wabbit 9.0，它是六年后的一个主要版本，同时鼓励你规划迁移，因为它拥有数个可用性改进，新降维算法和错误修复。

79. Android Gradle 插件 — Kotlin DSL

评估

Android Gradle 插件 Kotlin DSL 增加了 Gradle 构建脚本对 Kotlin Script 的支持，让它成为除 Groovy 之外的另一种选择。用 Kotlin 代替 Groovy 的目的在于 Kotlin 能更好得支持重构，并且在 IDE 里编写它更加简便，最终能够产出更易于阅读和维护的代码。对于已经在使用 Kotlin 的团队来说，这还意味着可以用更熟悉的语言编写构建脚本。我们曾经有一个团队在几天之内就对一份至少有七年、长达 450 行的构建脚本完成了迁移。如果你有一份庞大或者复杂的 Gradle 构建脚本，那么 Kotlin Script 值得一试，看看它是否会对你的团队产生帮助。

80. Azure Bicep

评估

Azure Bicep 是一种使用声明式语法的领域特定语言（DSL），主要面向那些喜欢使用比 JSON 更自然的语言来编写基础设施代码的人。它支持可重用参数化模板来实现模块化资源定义。它有 Visual Studio Code 插件为其提供实时类型安全、智能感知和语法检查的功能，并且它的编译器允许双向转换 ARM 模板。Bicep 面向资源的 DSL 以及与 Azure 生态系统的原生集成使其成为 Azure 基础设施开发人员的不二之选。

81. Capacitor

评估

我们争辩跨平台移动开发工具的优点几乎与我们发布技术雷达的时间一样长。在 2011 年发布 cross-mobile

platforms 时，我们首次注意到新一代工具的诞生。尽管我们最初对其持怀疑态度，然而这些工具在近些年逐渐被完善并被广泛采用。React Native 经久不衰的人气和用处是毫无争议的。Capacitor 是以 PhoneGap 为起源，之后被重命名为 Apache Cordova 的系列工具的最新一代。Capacitor 将 Ionic 完全重写并让独立应用拥抱渐进式网页应用风格。迄今为止，我们的开发者喜欢这种用单一代码库统一管理网页、iOS 和 Android 应用代码的方式，他们还可以按需访问原生 API 分别管理各个原生平台。尽管 React Native 已经坐拥多年的跨平台经验，Capacitor 还是为跨平台提供了一种额外选择。

82. Java 17

评估

我们通常不会专门介绍某一语言的新版本，但我们还是想关注一下 Java 新的长期支持（LTS）版本——Java 17。尽管出现了前景不错的新特性，比如模式匹配等预览特性，但其实向新 LTS 版本升级的方案更应该吸引各个组织的兴趣。我们建议各个组织在 Java 有新的发布时对其进行评估，确保能够恰当地适配这些新特性和版本。尽管定期更新有利于简化开发并且方便管理，但许多组织却意外地并不经常更新语言的版本。希望通过 LTS 版本的升级以及开发团队对语言的定期更新，能够使生产软件免于因为“更新成本太高”而一直困在 Java 的过时版本上。

83. Jetpack Glance

评估

Android 12 对应用小部件做了重大更改，从而改善了用户和开发人员的体验。对于编写常规的 Android 应用程序，我们已经表达了对 Jetpack Compose 作为以一种现代方式来构建原生用户界面的偏好。现在，借助构建在 Compose 运行时之上的 Jetpack Glance，开发人员可以使用类似的声明式 Kotlin API 来编写小部件。最近，Glance 已被扩展以支持在 Wear OS 中构建 Tiles。

84. Jetpack Media3

评估

现如今安卓拥有多个媒体 API：Jetpack Media（也被称为 MediaCompat），Jetpack Media2 和 ExoPlayer。然而，这些库都是分别开发的，它们的目的不同但是功能重叠。这就导致安卓开发者在编码的时候不仅需要斟酌类库的选型，当使用的特性来自于多个库的时候，还需要编写适配器或者兼容代码。Jetpack Media3 尝试去解决上述情况。它是从现有 API 中选取通用的功能——包括 UI、播放和媒体会话处理，然后将它们合并和改进成一个新的 API。Mediia3 目前仍处于早期开发版本。此外，ExoPlayer 的播放器界面也进行了更新、增强和简化，被用作 Media3 的通用播放器界面。

85. MistQL

评估

MistQL 是一个在类 JSON 结构上执行计算的小型领域特定语言。MistQL 最初的设计是用于在前端手动提取机器学习模型的特征，而如今它有了能够运行在浏览器端的 JavaScript 实现版本以及能够运行在服务器端的 Python 实现版本。我们非常喜欢 MistQL 的简洁组合函数式语法，并建议你根据需要进行试用。

86. npm 工作区

评估

在 node.js 的世界里，许多工具都支持多包开发，而 npm 7 中加入了 npm 工作区来直接支持此特性。将相关联的包集中管理可以让开发更加便利，比如你可以在一个代码仓库中存储多个相关的库。应用 npm 工作区后，一旦你在顶级的 package.json 文件中添加配置，引入了一个或多个嵌套的 package.json 文件，像 npm install 这样的命令就可以跨多个包使用，依赖的源包会符号链接到根目录的 node_modules 路径下。其他的 npm 命令也可以作用于工作区，例如，你可以只用一条命令在多个包中执行 npm run 和 npm test 命令。这种开箱即用的灵活性减少了一些团队对于其他包管理器的需要。

87. Remix

评估

我们见证了浏览器从服务器端渲染到单页应用的变迁，而如今的 Web 开发似乎又回到了两者中间。Remix 就是这样一个例子。Remix 是一个全栈 JavaScript 框架，它并没有使用笨拙的静态构建，而是通过利用分布式系统和本地浏览器两者的特点一起来加快页面的加载速度。它分别对嵌套路由和页面加载进行了部分优化，这使得页面渲染看起来特别快。Remix 与 Next.js 的定位十分相似，很多人也会将它们放在一起比较。我们很高兴地看到，这些框架可以巧妙地将浏览器渲染与服务器端渲染结合起来，从而提供更好的用户体验。

88. ShedLock

评估

有一种很常见的需求，是在分布式处理器集群上执行一次定时任务，且只执行一次。例如处理一批数据，发送一条通知，或者执行某个常规的清理操作，都属于这类情况。但是谁都知道这个问题很难，一组处理器如何通过有延迟而且不稳定的网络来实现稳定的协作？这就需要在集群中存在某种锁定机制，来协调这些操作。幸好有很多分布式存储可以实现这种锁定，ZooKeeper 和 Consul 等系统，以及 DynamoDB 或 Couchbase 等数据库都有必要的底层机制来管理集群内部的一致性。ShedLock 是一个小型类库，如果你正在尝试用 Java 来实现自己的定时任务，它可以使你的代码更方便地和上述工具集成。ShedLock 有获得和释放锁的 API，还有各种连接器，可以适配不同工具的锁。如果您正在编写自己的分布式任务，但是不想使用 Kubernetes 这种复杂的重量级平台，ShedLock 值得一试。

89. SpiceDB

评估

SpiceDB 受谷歌 Zanzibar 启发，是一个用于管理应用程序权限的数据库系统。你可以通过 SpiceDB 创建一个数据模式以对你的权限需求进行建模，并使用客户端库将创建的模式应用到任何一个受支持的数据库中；你也可以向数据库中插入数据，并高效地检索问题的答案，例如查询“这个用户有权访问某个资源吗？”，或者相反的问题：“这个用户有哪些资源可以访问？”通常，我们提倡将授权策略与代码分离开，但 SpiceDB 更进一步，将数据与策略分离并将其以图的形式进行存储，以高效地应答授权信息的查询。正因为这种分离，因此你必须确保应用程序的主要数据存储的变更会反映到 SpiceDB 中。我们发现，在受 Zanzibar 启发的各种实现中，SpiceDB 是一个值得你基于当前授权需求进行评估的有趣框架。

90. sqlc

评估

sqlc 是一个特别的编译器，它可以根据 SQL 生成类型安全并且风格自然的 Go 代码。与其他基于对象关系映射（ORM）的方法不同，sqlc 允许你根据需要编写原生的 SQL。一旦 sqlc 被调用，它会检查 SQL 代码的正确性并生成高性能的 Go 代码，这些代码可以直接被应用程序的其它部分调用。凭借对 PostgreSQL 和 MySQL 的稳定支持，sqlc 值得我们一试，因此我们鼓励你对其进行评估。

91. 可组合架构

评估

随着时间的推移，iOS 应用程序开发变得越来越顺畅，SwiftUI 进入采纳环就是一个标志。作为应用构建工具库与架构风格的集合体，可组合架构（TCA）在 SwiftUI 和其他常见框架之上更进了一步。TCA 是在一系列视频课程的基础上设计的。作者表示，他们在 The Elm Architecture 和 Redux 的基础上考虑了构图、测试和人体工程学。正如预期的那样，“适用面窄”和“有态度性（opinionatedness）”既是 TCA 的优势也是劣势。我们认为，对于需要维护多种不同技术栈代码库的团队来说，如果他们对编写 iOS 应用没有太多专业知识时，他们就能从使用像 TCA 这样的“有态度”的框架中获取最大收益。我们很喜欢 TCA 表现出的这种“态度”。

92. WebAssembly

评估

WebAssembly（WASM）是一项 W3C 标准，旨在为浏览器提供执行代码的能力。它是二进制的编码格式，其设计目标是可以发挥硬件的能力，让代码以接近原生的速度在浏览器中运行，目前 WASM 已被所有的主流浏览器支持并向下兼容。前端开发编程语言早期主要聚焦在 C、C++ 以及 Rust 上，WASM 的出现拓宽了可选范围。同时 WASM 还被 LLVM 支持，纳入为一个编译目标。当 WASM 在浏览器的沙盒环境中运行时，能够与 JavaScript 交互并共享相同的权限和安全模型。凭借其可移植性和安全性这两项关键能力，WASM 可以适配包括移动端、IoT 在内的更多平台。

93. Zig

评估

Zig 是一门新的语言，它与 C 语言共享了许多属性，但是具有更强的类型，更简便的内存分配，以及对命名空间和众多其他特性的支持。然而它的语法，比起 C 更容易让人想到 JavaScript，这点会引起一些人的反对。Zig 的目标是为大家提供一种非常简单的语言，可以直接编译以减少副作用，并且程序执行是可预测和易于追踪的。Zig 还提供了 LLVM 交叉编译功能的简化接口。我们的一些开发同事发现这一特性非常重要，以至于他们尽管没有使用 Zig 编程，但是仍然把它当做一个交叉编译器使用。Zig 是一种新颖的语言，对于正在考虑或者已经使用 C 语言的应用程序，以及需要显式内存操作的底层系统应用程序，值得一试。

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 成立于 1993 年，如今已从一个小团队发展成为在 17 个国家拥有 49 家办公室和超过 10,000 名员工的全球领先的软件及咨询公司。我们拥有专业卓越的跨职能团队，汇集了大量战略专家、开发人员、数据工程师和设计师，25 年多来，他们为全球各地的众多合作伙伴倾力服务。

Thoughtworks 首创“分布式敏捷”概念，我们深知如何集全球团队之力大规模交付卓越的软件。如今，我们致力于帮助客户开启流畅数字化之路，提升公司应变能力，引航未来征程。

/thoughtworks