ThoughtWorks®

TECHNOLOGY RADAR

有态度的前沿技术解析

Volume 23

#TWTechRadar

thoughtworks.com/radar

贡献者

技术雷达由ThoughtWorks技术顾问委员会创建，
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：

黄进军 / 伍斌 / 张凯峰 / 樊卓文 / 梁若琳 / 汪志成
王祎 / 徐栋栋 / 鄢倩 / 杨洋

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

Ni Wang

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和对行业有重大影响的技术趋势。

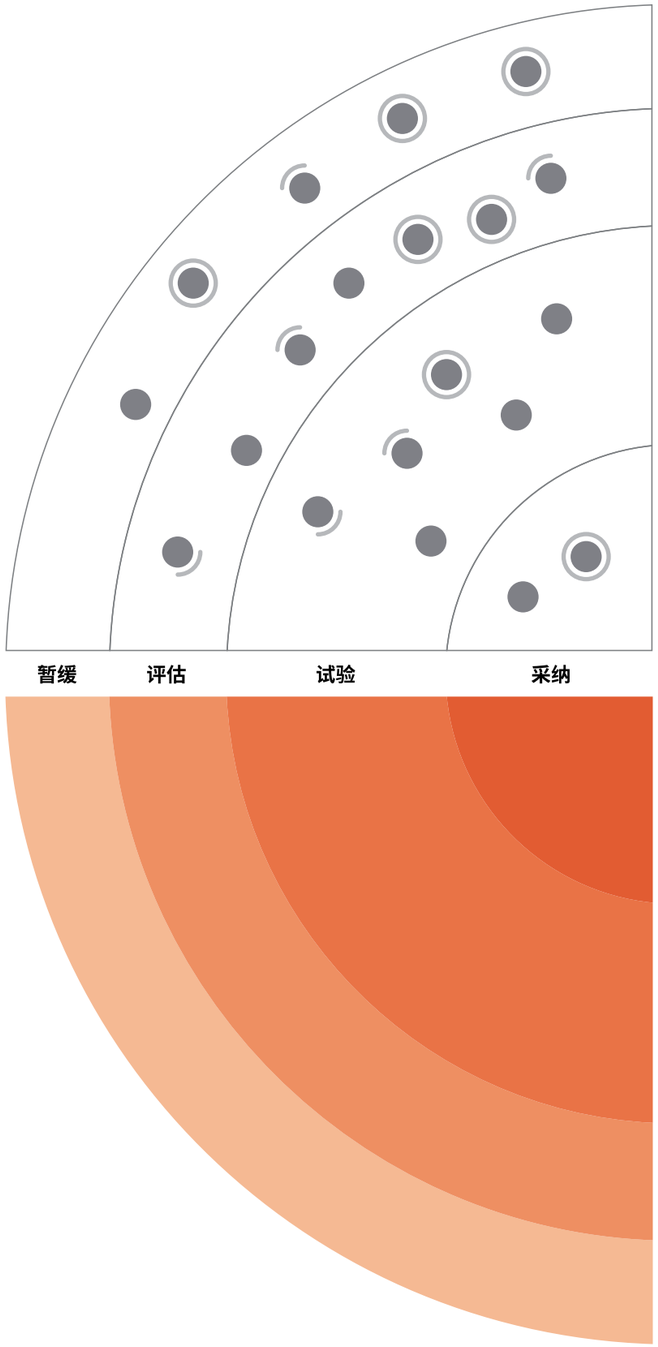
这个雷达以独特的形式记录技术顾问委员会的讨论结果，从首席技术官到开发人员，雷达为各路利益相关方提供价值。这些内容只是简要的总结，我们建议你探究这些技术以了解更多细节。这个雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言及框架。如果雷达技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。

想要了解更多技术雷达相关信息，请点击：
thoughtworks.com/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

GraphQL 浮夸风

我们看到 GraphQL 在很多团队中的采纳率激增，同时其支撑生态也在蓬勃发展。它解决了现代分布式架构（如微服务）中的一些共性问题：当开发人员把系统分解成很多小块时，他们通常还要把信息重新聚合起来才能解决业务需求。GraphQL 提供了一些功能，可以方便地解决这类日渐普遍化的问题。就像所有强大的抽象一样，它提供的是一种折衷方案，团队要认真考虑，以避免长线上的负面影响。比如，我们已经看到有团队通过聚合工具暴露了过多的底层实现细节，导致架构出现了不必要的脆弱性。当团队试图借助聚合工具来创建规范化的、通用的、中心化的数据模型时，就会把短线上的便利变成长线上的麻烦。我们鼓励团队使用 GraphQL 及其迅速成长的周边工具，但是，要小心别过度追求技术通用性，不要试图用一项技术解决很多问题。

与浏览器的斗争仍在继续

网页浏览器原本是被设计用来浏览文档的，但现在主要用来承载应用程序，这种抽象的不匹配一直困扰着开发人员。为了克服这种不匹配所带来的诸多问题，开发人员一直在重新审视和挑战那些公认的用于浏览器测试、状态管理和构建快速且丰富的浏览器应用程序的方法。我们在技术雷达上可以看到这一类的趋势。第一，自从2017年 Redux 作为管理 React 应用状态的默认方法被移到“采纳”环以来，我们看到开发人员要么仍在尝试其他的方法 Recoil，要么推迟对状态管理库的选型决策。第二，人们对 Svelte 越来越感兴趣，而它正在挑战虚拟 DOM 的概念，后者则正是 React 和 Vue.js 等流行的程序开发框架所遵循的概念。第三，用于处理浏览器端测试的新工具不断涌现：Playwright 是改进 UI 测试的又一个新尝试，而 Mock Service Worker 则是一种将测试与后端交互分离的新方法。第四，平衡开发人员的开发效率与应用性能一直都是我们需要面对的一个挑战，浏览器定制的腻子脚本的目的就是改变这个权衡的范围。

可视化一切

这一期技术雷达中，有几个条目来自不同技术领域但却拥有一个共性，即可视化。你会发现很多关于基础设施、数据科学、云资源，以及很多其他极富创新性的可视化工具，其中不乏一些可以有效可视化复杂抽象的方法。也有一些交互式的数据可视化工具和控制面板工具，如 Dash, Bokeh 和 Streamlit，还有一些基础设施的可视化工具，例如微服务架构中的服务网格可视化工具 Kiali。随着开发人员的生态环境变得越来越复杂，一幅能清晰地解释问题的图像对减轻大家的心智负担无疑是大有裨益。

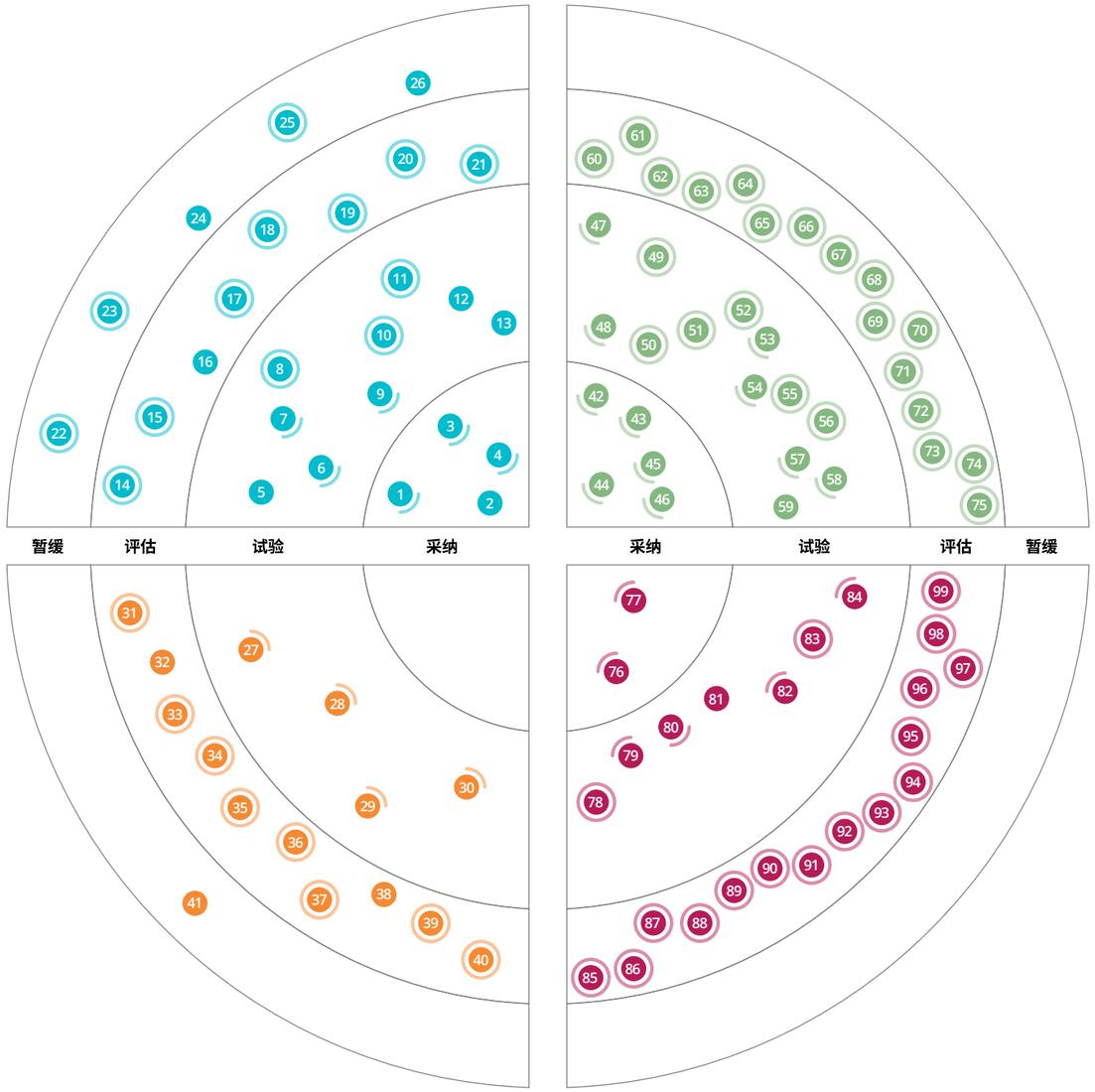
基础设施即代码的青春期

随着组织看到自动化基础设施所带来的好处，管理基础设施即代码变得越来越普遍。这为创新型的工具和框架的创建者们提供了反馈。诸如 CDK 和 Pulumi 之类的工具，提供了远远超过第一代工具的功能。其改进如此之大，以至于我们相信基础设施即代码已经进入了积极与消极因素共存的“青春期”。我们惊喜地看到在所有象限中，都有相关雷达条目，从积极的方面反映了相关生态系统日益成熟。但是，我们还讨论了该领域因为缺乏成熟模式而面临的挑战，以及许多公司在尝试用最佳方式利用此功能时所面临的挑战。所有这些都表明，该领域在持续增长，但尚未成熟。我们希望基础设施社区，继续从软件设计中汲取教训，尤其要关注创建松散耦合的可部署基础设施。

编程大众化

让非程序员能够执行以往只有程序员才能做到的任务，我们围绕这个促进编程大众化的工具和技术进行了一些讨论。而诸如 IFTTT 和 Zapier 之类的解决方案在该领域已长期流行。我们发现，人们开始越来越多地使用诸如 Amazon Honeycode 这样的低代码环境，以创建简单的业务应用程序。尽管此类工具提供了适合其目的的编程环境，但将其产出移至规模化的生产环境时仍会遇到挑战。开发人员长期以来一直设法利用电子表格向导工具，在特定领域和传统编码环境之间找到折衷方案。越来越多的现代工具的问世，在更广泛的领域重新激起了大家的讨论。但取舍的原则，依旧未变。

The Radar



技术

采纳
1. 依赖漂移适应度函数
2. 将运行成本实现为架构适应度函数
3. 安全策略即代码
4. 定制化服务模板

试验
5. 机器学习的持续交付
6. 数据网格
7. 声明式数据管道定义
8. 图表即代码
9. Distroless Docker images
10. 事件拦截
11. 并行核对
12. 使用原生的远程工作方法
13. 零信任架构

评估
14. 限界低代码平台
15. 浏览器定制的腻子脚本（polyfills）
16. 去中心化身份
17. 由 Kube 管理的云服务
18. 开放应用程序模型
19. 安全区域
20. 回旋实验
21. 可验证凭证

暂缓
22. Apollo Federation
23. 披着API网关外衣的企业服务总线
24. 用于业务分析的日志聚合
25. 微前端的无序
26. 生产化的笔记本

平台

采纳

试验
27. Azure DevOps
28. Debezium
29. Honeycomb
30. JupyterLab

评估
31. Amundsen
32. AWS云开发工具包
33. Backstage
34. Dremio
35. DuckDB
36. K3s
37. Materialize
38. Pulumi
39. Tekton
40. 基于IP协议栈的信任

暂缓
41. Node 泛滥

工具

采纳
42. Airflow
43. Bitrise
44. Dependabot
45. Helm
46. Trivy

试验
47. Bokeh
48. Concourse
49. Dash
50. jscodeshift
51. Kustomize
52. MLflow
53. Pitest
54. Sentry
55. ShellCheck
56. Stryker
57. Terragrunt
58. tfsec
59. Yarn

评估
60. CML
61. Eleventy
62. Flagger
63. gossm
64. Great Expectations
65. k6
66. Katran
67. Kiali
68. LGTM
69. Litmus
70. Opacus
71. OSS Index
72. Playwright
73. pnpm
74. Sensei
75. Zola

暂缓

语言 & 框架

采纳
76. Arrow
77. jest-when

试验
78. Fastify
79. Immer
80. Redux
81. Rust
82. single-spa
83. Strikt
84. XState

评估
85. Babylon.js
86. Blazor
87. Flutter Driver
88. HashiCorp Sentinel
89. Hermes
90. io-ts
91. Kedro
92. LitElement
93. Mock Service Worker
94. Recoil
95. Snorkel
96. Streamlit
97. Svelte
98. SWR
99. Testing Library

暂缓

TECHNOLOGY RADAR

技术

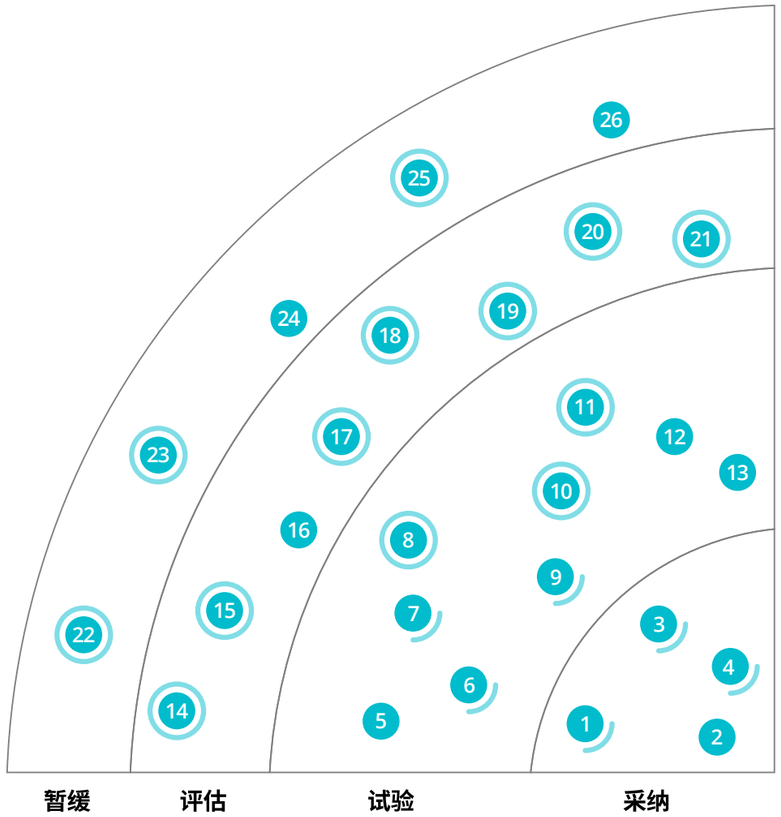
技术

依赖漂移适应度函数
采纳

演进式架构借用自进化计算而引入的适应度函数，可以客观地展示应用程序及架构是否正在偏离期望的指标，实际上是可以集成到发布流水线中的测试。依赖漂移适应度函数追踪应用程序一个主要指标，即应用依赖的库、API或环境组件的新鲜度，并可以将过时需要更新的依赖标记出来。随着 Dependabot、Snyk 这类用于检测依赖漂移的工具日趋成熟，我们可以轻松地在软件发布流程中加入依赖漂移适应度函数，以保证应用程序依赖的更新。

将运行成本实现为架构适应度函数
采纳

对于今天的组织来说，自动化评估、跟踪和预测云基础设施的运行成本是必要的。云供应商精明的定价模型，以及基于定价参数的费用激增，再加上现代架构的动态本质，常常导致让人吃惊的运行成本。例如，无服务架构基于API访问量的费用，事件流方案中基于流量的费用，以及数据处理集群中基于运行任务数量的费用，它们都具有动态的本质，会随着架构演进而产生改变。当我们的团队在云平台上管理基础设施时，将运行成本实现为架构适应度函数是他们的早期活动之一。这意味着我们的团队可以观察运行服务的费用，并同交付的价值进行对比；当看到与期望或可接受的结果之间存在偏差时，他们就会探讨架构是否应该继续演进了。对运行成本的观察和计算需要被实现为自动化的函数。



安全策略即代码
采纳

随着技术格局逐渐复杂化，诸如安全性的问题需要引入更多的自动化和工程实践。在系统构建过程中，我们需要将安全策略——即保护系统免受威胁和破坏的规则和程序，纳入考虑中。例如，访问控制策略定义，并强制谁在什么情况下可以访问哪些服务和资源；相反，网络安全策略应动态限制特定服务的流量速率。我们的一些团队在安全策略即代码上有着丰富的经验。当我们说…即代码时，不仅意味着将这些安全策略写入文件中，还需要将其应用到诸如在代码中采用版本控

采纳
1. 依赖漂移适应度函数
2. 将运行成本实现为架构适应度函数
3. 安全策略即代码
4. 定制化服务模板

试验
5. 机器学习的持续交付
6. 数据网格
7. 声明式数据管道定义
8. 图表即代码
9. Distroless Docker images
10. 事件拦截
11. 并行核对
12. 使用原生的远程工作方法
13. 零信任架构

评估
14. 限界低代码平台
15. 浏览器定制的腻子脚本（polyfills）
16. 去中心化身份
17. 由 Kube 管理的云服务
18. 开放应用程序模型
19. 安全区域
20. 回旋实验
21. 可验证凭证

暂缓
22. Apollo Federation
23. 披着API网关外衣的企业服务总线
24. 用于业务分析的日志聚合
25. 微前端的无序
26. 生产化的笔记本

技术

Data mesh 标志着关于我们如何管理大量分析数据的范式转移，这会尝试去解决过去中心化分析数据管理中很多已知的挑战。但是，支持 data mesh 的开源和商业工具之间的差距还比较大。

(数据网格)

制、在流水线中引入自动验证、在环境中自动部署并观察监控其性能等实践中。基于我们的经验以及现有工具（包括开放策略代理）和平台（如提供了灵活策略定义和实施机制，以支持安全策略即代码实践的 Istio）的成熟度，我们强烈建议在你的环境中使用此技术。

定制化服务模板

采纳

自从定制化服务模板在雷达中出现以来，这个模式已经被广泛采用以帮助组织向微服务过渡。伴随着可观察性工具、容器编排和服务网格边车的不断进步，服务模板可以通过精心挑选的默认值，减少服务与基础设施配合所需的大量设置，从而帮助快速建立新服务。对定制化服务模板应用产品管理原则也取得了成功。以内部开发者作为客户，定制化服务模板可以帮助开发者将代码发布到生产环境，并提供合适的可观察性以进行操作。定制化服务模板带来的另一个好处，是可以作为轻量级的治理机制，对技术选型的默认项进行集中管理。

机器学习的持续交付

试验

大约十年前，我们引入了持续交付(CD)，将其作为我们交付软件解决方案的常规方式。如今解决方案在不断增多，其中也包括机器学习模型，并且我们发现在这种解决方案上，持续交付实践仍然适用。我们将之称为机器学习的持续交付(CD4ML)。尽管 CD 的原则仍保持不变，但用于实现训练、测试、部署和监测模型的端到端过程，其实践和工具需要进行一些修改。例如:版本控制不仅要考虑代码的版本控制，还要考虑数据、模型及其参数的版本控制；测试金字塔扩展为包含模型偏差、公平性、数据和特征验证；部署过程必须考虑如何针对当前的冠军模型来提升和评估新模型的性能。当业界正在为 MLOps 这个新的流行词庆祝时，我们反而认为 CD4ML 才是我们实现端到端（可靠地发布以及持续改进机器学习模型，由想法到生产）过程的整体方法。

数据网格

试验

在管理大量分析数据方面，Data mesh 标志着架构和组织范式的一种可喜转变。该范式建立在四个原则之上：(1)数据所有权和架构的面向领域去中心化;(2)将面向领域的数据视为产品;(3)将自助数据基础设施作为平台，支持自治且面向领域的数据团队;(4)联合控制以实现生态系统和互操作性。尽管这些原则很直观，并且只是试图解决以前集中分析数据管理的许多已知挑战，它们仍胜过了现有的分析数据技术。在现有工具之上为多个客户端构建数据网格后,我们学到了两件事:(a)要加速数据网格的实现，在开源或商业工具上仍存在着巨大的差距（例如,我们目前为客户自定义构建的基于时间的多语言数据，实现通用访问模型）;(b)尽管存在差距,使用现有技术作为基本构建块仍是可行的。

自然，技术匹配是实现企业基于数据网格的数据策略的主要组成部分。不过，要想成功，就需要进行组织结构调整，将数据平台团队分离开来，为每个领域创建数据产品负责人的角色，并引入必要的奖励机制，让领域将其分析数据作为产品，对其负责，并分享出去。

声明式数据管道定义

试验

许多数据管道都是在一个巨大的、或多或少是由 Python 或 Scala 编写的命令式脚本中定义的。该脚本包含各个步骤的逻辑以及将这些步骤链接在一起的代码。在 Selenium 测试中遇到类似的情况时，开发人员发现了 Page Object 模式，此后许多行为驱动开发(BDD)框架都实现了将步骤定义与整合代码进行分离。一些团队正在尝试将同样的思想引入数据工程。单独的声明式数据管道定义(也许是用 YAML 编写的)只包含声明和步骤顺序。它声明输入和输出数据集，在需要更复杂逻辑时引用脚本。A La模式是一个相对较新的工具，它采用 DSL 方法来定义管道，不过 airflow-declarative 工具似乎是这个领域中最有前景的工具，它是一个将 YAML中定义的有向无环图转换为 Airflow 任务调度的工具。

图表即代码

试验

我们看到越来越多的用于创建软件架构和图表即代码 的工具。相较于使用其他更重量级的工具，这些工具可以更方便的做版本控制，还可以从多个源创建领域特定语言。我们很喜欢这类工具，例如 Diagrams，Structurizr DSL，AsciiDoctor Diagram，还有诸如 WebSequenceDiagrams，PlantUML 等系列

产品，当然还有久负盛名的 Graphviz。现在创建一个 SVG 也变得相当简单了，如果能自己动手很快地写一个小工具来做这个，倒也不失为一个不错的选择。例如，本期雷达的某位作者就自己动手写了一个小的 Ruby 脚本用来快速地创建 SVG。

Distroless Docker images
试验

当为我们的应用构建 Docker 镜像的时候，我们常常会考虑两件事情：镜像的安全性和大小。通常情况下我们使用容器安全扫描工具来检测和修复常见的漏洞和风险，以及使用 Alpine Linux 来解决镜像大小和分发性能问题。我们现在已经获得了有关 distroless Docker images 的更多经验，并准备推荐这种方法作为容器化应用程序的另一重要安全预防措施。Distroless Docker images 通过移除完整的操作系统发行版来减少占用空间和依赖。此技术可减少安全扫描噪声和应用程序攻击面，需要修补的漏洞较少，此外，这些较小的镜像更有效。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 或者仅仅使用多阶段 Dockerfiles 创建简单的应用程序镜像。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 Distroless 容器的调试版本，包括 busy box shell。Distroless Docker image 是 Google 率先提出的技术，根据我们的经验，仍然主要限于 Google 生成的镜像。我们希望这项技术能够超越这一生态系统。

事件拦截
试验

随着越来越多的公司从遗留系统中迁移出来，我们觉得有必要强调一种从这些系统中获取数据的新机制，它可以作为变动数据捕获（CDC）的替代方案。Martin Fowler 早在 2004 年就描述了事件拦截。在现代术语中，它涉及到在进入系统时将请求分流，以便逐步构建一个替代系统。这通常是通过复制事件或消息来实现的，但是 HTTP 请求分流也同样有效。例如在将事件写入大型机之前在销售点系统处将事件分流，又如在将支付事务写入核心银行系统之前对其进行分流。这两种情况都会导致部分遗留系统的逐步替换。我们认为，这种从源头获取状态更改，而不是使用 CDC 进行后期处理来重新创建状态更改的技术，一直以来都被忽视了，这也是我们在本期技术雷达中强调它的原因。

并行核对
试验

大规模替换遗留代码始终是一项艰巨的工作，而且经常受益于执行并行核对（Parallel run with reconciliation）。实际上，该技术依赖于通过旧代码和新代码执行相同的生产流程，从旧代码返回响应，比较结果从而对新代码产生信心。尽管这是一种古老的技术，但近年来，我们在持续交付实践（如金丝雀发布和特性切换）的基础上看到了更健壮的实现，并通过添加额外的实验和数据分析层来比较实时结果来扩展它们。我们甚至已经使用这种方法来比较跨功能的结果，例如响应时间。尽管我们已结合定制工具多次使用该技术，但我们还是要感谢 GitHub 的Scientist 工具，该工具对应用程序的关键部分进行了现代化改造，现已移植到多种语言。

使用原生的远程工作方法
试验

随着新冠疫情的蔓延，至少在目前，高度分散的团队似乎将成为“新常态”。在过去的六个月中，我们掌握了许多关于有效远程工作的知识。从积极的方面来说，良好的视觉工作管理和协作工具使得和同事们远程协作比以往更容易。例如，开发人员可以借助 Visual Studio Live Share 和 GitHub Codespaces 来促进团队协作并提高生产力。远程工作的最大弊端可能是筋疲力尽：许多人一整天被安排了大量的“背对背”视频通话，这种缺点带来的风险也开始显现。尽管在线可视化工具使得协作变得更容易，但也有可能会构建出复杂的巨型图，而这些图最终会难以使用；并且工具扩散的安全性方面也需要小心管理。我们的建议是记住后退一步，和你的团队沟通，评估可行与不可行的方法，并根据需要改变流程和工具。

零信任架构
试验

企业中计算和数据的结构不断变化：从单一应用程序到微服务；从集中式数据湖到数据网格；从本地托管到聚合云。与此同时，随着连接设备的激增，保护企业资产的方法在很大程度上仍保持不变。凭借对网络外围的高度依赖和信任——通过加强企业虚拟墙，使用专用链路和防火墙配置，以及替换不再适

技术

我们看到能帮助你通过代码创建软件架构和其他图表的工具正在涌现。它们的好处在于方便的版本控制，以及从多个来源生成DSL的能力。

(图表即代码)

在从遗留系统迁移时，事件拦截是捕捉数据变化很重要的一种选择。比起通过 CDC 尝试去重新创建并进行后处理的过程，直接从来源获得状态的变化，一直被忽视了。

(事件拦截)

技术

Low-code 和 no-code 平台可以在非常有限的领域内解决非常特定的问题。尽管如此，我们还是对它的普适性保持怀疑。

(限界低代码平台)

用于当今现实的静态和繁琐的安全过程的方法，企业继续进行大量投资，以保护其资产。这种持续的趋势迫使我们再次强调“零信任架构”。

零信任架构是安全体系结构及策略的一种模式转变。它基于这样的假设：网络边界不再代表安全边界，不应仅基于物理或网络位置授予用户或服务隐式信任。用于实现零信任架构各个方面的资源，工具以及平台的数量持续增长，其中包括：以最小特权为基础，执行安全策略即代码，尽可能细化原则，并持续监控和自动缓解威胁；使用服务网格来实施应用到服务和服务到服务的安全控制；使用二进制鉴证以验证二进制文件的来源；除传统加密外，还包括安全区域，以强制实施数据安全性的三大支柱：传输，静态和内存。有关该主题的有用介绍，请参阅 NIST ZTA 刊物和有关 BeyondProd 的 Google 白皮书。

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。

浏览器定制的腻子脚本（polyfills）

评估

腻子脚本在 Web 演进的过程中非常有用，它为那些尚未实现某些现代特性的浏览器提供了替代功能。但是，Web 应用通常会把这些腻子脚本发布到并不需要它们的浏览器中，这就导致了不必要的下载和解析开销。情况已经越来越明朗，因为现在只剩下少数几个渲染引擎，而大多数腻子脚本只针对其中之一：IE11 使用的 Trident 渲染引擎。此外，它的市场份额正在不断减少，其技术支持也将在一年内结束。所以，我们建议你使用浏览器定制的腻子脚本，仅向特定浏览器发布必要的腻子脚本。该技术甚至已经由 Polyfill.io 实现成了服务。

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。

由 Kube 管理的云服务

评估

云提供商已开始通过自定义资源定义（CRD）逐渐支持 Kubernetes 样式的 API 来管理其云服务。在大多数情况下，这些云服务是基础架构的核心部分，我们已经看到团队使用诸如 Terraform 或 Pulumi 之类的工具进行配置。有了这些新 CRD，如用于 AWS 的 ACK；

用于 Azure 的 Azure Service Operator 和用于 GCP 的 Config Connectors，你就可以使用 Kubernetes 来整备和管理这些云服务。这些由 Kube 管理的云服务的优点之一是应用程序和基础设施的声明状态可以用相同的 Kubernetes 控制平面来实现。缺点是它将 Kubernetes 集群与基础设施紧密结合在一起，因此我们正在仔细评估它，你也应该这样做。

开放应用程序模型
评估

我们以前多次谈到创建平台工程产品团队来支持公司其他产品团队有许多好处，但实施起来确实很难。在基础设施即代码的世界中，业界似乎仍在寻找正确的抽象。尽管诸如 Terraform 和 Helm 之类的工具，已经朝着正确的方向迈进，但其重点仍然是管理基础设施，而不是应用程序开发。当然，业界还是存在一些向“基础设施即软件”方向的转变，比如涌现出 Pulumi 和 CDK 等新工具。而 开放应用程序模型(OAM) 则试图对该领域进行标准化。通过使用组件、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。OAM 是否能被广泛采用还有待观察，但是我们建议关注这个有趣且有用的想法。

安全区域
评估

安全区域，也称为可信执行环境 (TEE)，是指一种隔离具有较高安全级别的环境 (处理器，内存和存储)，并且仅提供与其周围的不受信任执行上下文进行有限信息交换的技术。例如，硬件和系统级别的安全区域可以创建并存储私钥，并使用它们执行如加密数据或验证签名等操作，而无需私钥离开安全区域或将其加载到不受信任的应用程序内存中。安全区域提供了一系列有限的指令来执行受信任的操作，并隔离不受信任的应用上下文。

长久以来，这项技术一直得到许多硬件和系统供应商 (包括 Apple) 的支持，并且已有开发人员在物联网和边缘应用中使用该技术。然而，直到最近它才在企业和基于云的应用中获得关注。云提供商已经开始引入机密计算功能，如基于硬件的安全区域：Azure 机密计算基础架构允许启用 TEE 的虚拟机，并通过 Open Enclave SDK 开源库进行访问以执行受信操作。同样地，仍处于测试阶段的 GCP 机密虚拟机和 Compute Engine 允许使用在内存中进行数据加密的虚拟机，AWS Nitro Enclaves 紧随其后，即将发布其预览版。随着基于云的安全区域和机密计算的引入，我们现在可以在数据保护的两个支柱：存储的数据保护，传输的数据保护上添加第三个支柱：内存的数据保护。

尽管仍处于企业安全区域的初级阶段，我们还是建议你考虑此技术，同时了解已知可能危及底层硬件提供商的安全区域漏洞。

回旋实验
评估

使用 A/B 测试进行对照实验是揭示有关产品开发决策的好方法。但是，当我们无法让参与 A/B 测试的两个小组之间彼此独立时，这个方法就失效了，也就是说，将某人添加到”A”小组中会影响“B”小组，反之亦然。解决此问题空间的一种技术是回旋实验。这里的核心概念是我们在特定区域中以交替的时间段在实验的“A”和“B”模式之间来回切换，而不是在同一时间段内同时运行。然后，我们比较两个时段之间的客户体验和其他关键指标。我们已经在某些项目中尝试了此方法，并且取得了不错的效果——它是我们的实验工具栏中一款很好的工具。

可验证凭证
评估

凭证在我们生活中无处不在，例如护照、驾照和学历证书。但是，当今大多数数字凭证都是来自信息系统的简单数据记录，易于修改和伪造，并且经常暴露出不必要的信息。近年来，我们已经看到逐步成熟的 可验证凭证 解决了这一问题。W3C 标准将其定义为一种加密安全、尊重隐私和机器可验证的手段。与

技术

今天绝大多数的电子凭证都是来自信息系统的简单数据记录，易于修改和伪造，并经常暴露不必要的信息。最近这些年，我们看到能解决这个问题的可验证凭证变得越来越成熟。

(可验证的凭证)

技术

无论你怎么称呼它，把业务逻辑放入中心化工具，都是在创建架构耦合，降低透明度并增加厂商绑定，却没有明显的好处。

(披着API网关外衣的企业服务总线)

我们使用物理凭据时的经验类似，该模型以凭证持有者为中心：用户可以将可验证的凭证放入自己的数字钱包中，并在没有凭证发行人许可的情况下随时向其他人展示。这种去中心化的方法还使用户能够更好地管理自己的信息并有选择地公开，从而大大改善了数据隐私保护。例如，借助零知识证明技术，你可以构建可验证的凭证，无需透露自己的生日即可证明你是成年人。社区围绕可验证凭证开发了许多用例。我们已参考COVID-19凭证计划（CCI）实施了自己的 COVID 健康认证。尽管可验证凭证不依赖于区块链技术或去中心化身份，但该技术在实践中通常与 DID 结合使用，并将区块链用作可验证的数据注册表。许多去中心化身份的框架也嵌入了该技术。

Apollo Federation

暂缓

我们首次在技术雷达中介绍 GraphQL 时，曾提醒误用它会导致反模式，从长远来看弊大于利。尽管如此，我们发现团队对 GraphQL 越来越感兴趣，因为它能够 聚合来自不同资源的信息。这次我们想提醒你谨慎使用 Apollo Federation 和它对公司统一数据图的强大支持。即便乍看之下，有跨组织的普适概念这种想法是具有吸引力的，但是我们必须考虑之前业界做过的类似尝试——如 MDM 和规范数据模型等，这些尝试暴露了这种方法的缺陷。挑战会是巨大的，特别是当我们发现所在的领域要创建一个独特统一的模型非常复杂的时候。

披着API网关外衣的企业服务总线

暂缓

长期以来，我们都反对使用 中心化的企业服务总线，并且将“智能端点和哑管道”定义为微服务架构的一个核心特性。遗憾的是，我们观察到传统的企业服务总线正在沉渣泛起，披着API网关外衣的企业服务总线 正卷土重来，这自然会对过度庞大的API网关起到推波助澜的作用。不要被市场所惑，无论它顶着什么名号，只要是将业务逻辑（包括编排和转换）放到一个中心化的工具中，都必然会增加架构的耦合度，降低系统的透明度，增加对供应商的绑定，而且还没有任何明显的好处。API 网关仍然可以作为对通用关注点的一个非常有用的抽象，但是我们始终相信，业务逻辑应该由 API 提供，而不是 API 网关或是企业服务总线。

用于业务分析的日志聚合

暂缓

几年前，出现了新一代的日志聚合平台，该平台能够存储和搜索大量的日志数据，用来发掘运营数据中的趋势和洞见。这种工具有很多，Splunk 是其中最著名的。由于这些平台可在整个应用程序范围内提供广泛的运营和安全可见性，因此管理员和开发人员越来越依赖于它们。当干系人发现可以使用日志聚合进行业务分析时，他们便开始乐于此道。但是，业务需求可能会很快超越这些工具的灵活性和可用性。旨在提供技术可观察性的日志通常很难对用户有深刻的理解。我们更喜欢使用专门为用户分析而设计的工具和指标，或者采用更具事件驱动性的可观察性方法，其中收集、存储业务和运营事件的方式可以用更多专用工具来重现和处理。

微前端的无序

暂缓

自从我们2016年首次引入微前端以来，这个理念已经变得越来越受欢迎，并获得了主流的认可。但是正如其他名称比较易记的新技术一样，微前端偶尔也会被误用或滥用。尤其值得注意的是，在人们倾向于将一系列相互竞争的技术、工具或框架混合使用在一个页面中时，往往会拿微前端来做挡箭牌，从而导致微前端的无序。而这其中则以多个前端框架的混用尤甚。例如，在单页面应用中混合使用 React.js 和 Angular。虽然这种做法在技术上是有可能的，但如果不是作为某个经过深思熟虑的过渡策略的一部分，那这种做法就是非常不可取的。团队之间还需要保持样式技术(例如：CSS-in-JS 或 CSS modules) 和组件集成方式(例如：iFrames 或 web components)的一致性。此外，在状态管理、数据

获取、构建工具、分析等其他方面，组织还需要决定到底应该保持一致的标准化方式，还是将这些问题都交由团队自己做决定。

生产化的笔记本

暂缓

在过去的几十年里，最初由 Wolfram Mathematica 引入的计算笔记已经发展到可以支持科学研究、探索和教育工作流程了。自然，它们还支持数据科学工作流，并且诸如 Jupyter notebooks 和 Databricks notebooks 已经成为了一个很好的工具，它们提供了简单且直观的交互计算环境，能够结合代码来分析富文本数据，并将其可视化来讲述数据故事。这些笔记本是作为提供现代科学交流和创新的最终媒介而设计的。不过，在近几年，我们发现一种趋势：笔记本成为运行生产质量类型的代码媒介，其中这些代码通常用于驱动企业运营。我们看到笔记本平台供应商宣传他们的探索笔记本在生产中的使用。这是一个好的期望——对数据科学家来说，简易化编程却没有得到很好的实现，并且牺牲了可扩展性、可维护性、弹性以及一个长线产品代码所需支持的所有其他品质。因此我们不推荐生产化的笔记本，而是鼓励对数据科学家赋能，使其能够使用正确的编程框架构建预生产代码，从而简化持续交付工具以及端到端机器学习平台的抽象复杂性。

技术

我们看到了一个趋势：借口使用微前端在单个页面中混合一些彼此竞争的技术、工具和框架。这在技术上当然可能，但我们永远不会建议这么做。

(微前端无序)

TECHNOLOGY RADAR

平台

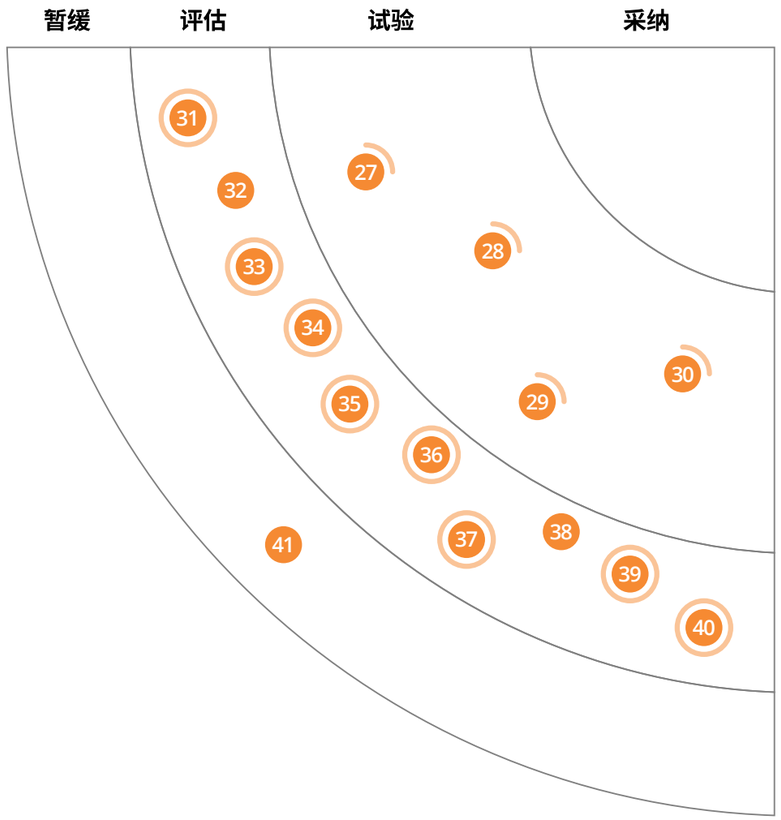
平台

Azure DevOps
试验

Azure DevOps 服务包含 Git 仓库托管、CI/CD 流水线、自动化测试工具、待定项管理工具以及生成物 (artifact) 仓库等在内的一系列管理服务。在这个平台上，我们的团队积累了许多经验，并取得了不错的成绩——这也意味着 Azure DevOps 的逐步成熟。在灵活性上我们对它喜爱有加：它允许你使用来自不同供应商的服务。例如，你可以在 Azure DevOps 流水线服务上使用外部 Git 仓库。尽管我们的团队对 Azure DevOps Pipelines 尤为感兴趣，但不得不说，所有的 Azure DevOps 服务都提供了良好的开发者体验，有助于我们的团队交付价值。

Debezium
试验

Debezium 是一个变更数据捕获 (Change Data Capture, CDC) 平台，可以将数据库变更流式传输到 Kafka 的 topics。CDC是一种流行的技术，具有多种应用场景，例如：将数据复制到其他数据库、输送数据给分析系统、从单体中提取数据至微服务以及废除缓存。Debezium 对数据库日志文件中的变更做出反应，并具有多个CDC连接器，适用于多种数据库，其中包括Postgres、MySQL、Oracle 和 MongoDB。我们在许多项目中都使用了 Debezium，它对我们来说非常有效。



Honeycomb
试验

Honeycomb 是一种可观测性服务，它可以从生产系统中提取丰富的数据，并可以通过动态采样对其进行管理。开发人员可以记录大量丰富的事件，然后决定如何对它们进行切片和关联。理性预测生产系统出了哪些问题的时代已经过去，在当今的大型分布式系统中，这种交互式方法非常有用。Honeycomb 团队正在积极开发多种语言和框架的插件，已经支持的有Go、Node、Java 和 Rails，其他新功能正在迅速添加。另外，其简化过的定价模型也让它更具吸引力。我们团队喜欢它。

采纳

试验
27. Azure DevOps
28. Debezium
29. Honeycomb
30. JupyterLab

评估
31. Amundsen
32. AWS云开发工具包
33. Backstage
34. Dremio
35. DuckDB
36. K3s
37. Materialize
38. Pulumi
39. Tekton
40. 基于IP协议栈的信任

暂缓
41. Node 泛滥

平台

JupyterLab 的交互式环境是 Jupyter Notebook 的一次演进，在一系列新的特性中，包括对于原有拖放 cell 和 tab 自动完成功能的扩展。

(JupyterLab)

Backstage 是来自 Spotify 的开源平台，可以创建开发者门户，这样能标准化你的团队正在使用的工具和技术的数量，并防止你的软件生态陷于碎片和复杂。

(Backstage)

JupyterLab
试验

在上一期雷达中，JupyterLab 还处于评估象限，作为项目 Jupyter 基于 web 的用户界面，现在它已经成为许多数据从业者的首选。JupyterLab 的使用正在迅速超越 Jupyter Notebooks，且终将取而代之。如果你仍在使用 Jupyter Notebooks，去尝试一下 JupyterLab 吧。JupyterLab 的交互环境是对 Jupyter Notebook 的改进：通过单元格拖拽、tab 自动补全等新特性对原有的功能进行了扩展。

Amundsen
评估

数据科学家将大量时间花在数据发现上，这意味着在这一领域能够提供帮助的工具势必会令人兴奋。尽管 Apache Atlas 项目已经成为了元数据管理的实际工具，但数据发现仍然不是那么容易完成的。Amundsen可以与 Apache Atlas 协同部署，为数据发现提供更好的搜索界面。

AWS云开发工具包
评估

对于我们许多团队来说，Terraform 已成为定义云基础设施的默认选择。但是，我们的一些团队一直在尝试使用 AWS云开发工具包 (AWS CDK)，并对其爱不释手。他们尤其喜欢该工具能使用一流的编程语言，而不是配置文件，从而可以利用现有的工具、测试方法和技能。像类似的工具一样，确保部署易于理解和维护也需要花费心思。该工具包目前支持 TypeScript、JavaScript、Python、Java、C#和.NET。AWS 和 HashiCorp 团队最近发布了 Terraform云开发工具包预览版，以生成 Terraform 配置，并用于 Terraform 平台的整备。我们将继续观察 AWS CDK。

Backstage
评估

组织正在寻求通过开发人员门户或平台来支持和简化开发环境。随着工具和技术数量的增加，为了让开发人员能够专注于创新和产品开发而不必为重新发明轮子陷入困境，某种形式的标准化对于保持一致性来说越来越重要。集中的开发人员门户可以轻松提供服务发现和最佳实践。Backstage 是一个由 Spotify 提供的创建开发人员门户的开源平台。它基于软件模板、统一的基础设施工具以及一致且集中的技术文档。它的插件化架构允许对组织的基础设施生态系统进行扩展和适配。

Dremio
评估

Dremio 是一个云数据湖引擎，支持针对云数据湖存储的交互式查询。通过使用 Dremio，你无需再为了满足预测性能而将数据提取和转存到一个专门的数据仓库，因此也无需再专门管理那些数据管道了。Dremio 为数据湖中的数据创建虚拟的数据集，并为消费者提供统一的视图。Presto 使存储与计算层分离的技术得到了普及，而 Dremio 则通过提升性能和优化运营成本使这一技术得到了更进一步的推广。

DuckDB
评估

DuckDB 是一个嵌入式列式数据库，可用于数据科学与数据分析。在将数据转移到服务器之前，分析师花费大量时间在本地清洗数据和可视化数据。尽管数据库已经存在了几十年，但大多数数据库都是为客户端-服务器用例设计的，因此不适合本地交互式查询。为了解决这一问题，分析人员通常会使用内存数据处理工具，例如：Pandas 或 data.table。尽管这些工具是有效的，但它们能分析的范围只限于内存恰好能容纳的数据量大小。我们认为DuckDB 用嵌入式列式引擎巧妙地填补了工具方面的空白，该引擎针对本地、大于内存的数据集进行了优化。

K3s
评估

K3s 是一个轻量级的用于物联网和边缘计算的 Kubernetes 发行版。K3s 被打包成一个单独的二进制文件，对于操作系统的依赖性微乎其微，这使得它非常易于运维和使用。K3s 使用 sqlite3 而非 etcd 作为默认的存储后端。由于所有相关的组件都运行在同一个进程里，这使得 K3s 的内存占用非常低。通过剥离不相关的第三方存储驱动和云提供商，K3s 的二进制文件得以控制得非常小。在资源受限的环境中，K3s 是一个值得考虑的非常不错的选择。

Materialize
评估

Materialize 是一种具备以下优势的流式数据库——无须配置复杂的数据管道，就可进行增量计算。其配置过程十分简单，只须通过标准 SQL 视图来描述计算，然后将 Materialize 连接到数据流即可。底层的差异数据流引擎能够执行增量计算，以最小的延迟提供一致且正确的输出。与传统数据库不同，定义 Materialize 的视图没有任何限制，并且计算是实时执行的。

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。

Tekton
评估

Tekton 是一个诞生不久的 Kubernetes 原生平台，用于管理持续集成和交付(CI/CD)管道。它不仅在 Kubernetes 上安装和运行，并且还将其 CI/CD 管道定义为 Kubernetes自定义资源。这意味着这些管道现在可以由原生的 Kubernetes 客户端(CLI 或 api)控制，并且可以利用底层资源管理特性(如回滚)。管道的声明格式很灵活，并且允许定义有条件的工作流、并行执行路径以及操作最终任务进行清理等特性。因此，Tekton 可以支持具有回滚、金丝雀发布等功能的复杂混合部署工作流。Tekton 是开源的，同时也作为GCP 管理服务被提供出来。虽然文档还有待改进，社区也在扩大，不过我们已经成功地将 Tekton 用于 AWS 上的生产工作负载。

基于IP协议栈的信任
评估

个人和组织如何在网络上建立数字化的信任，这一持续的挑战催生了关于如何证明身份、如何分享和验证建立信任所需的属性以及如何安全交易的新方法。我们的技术雷达介绍了一些开启数字信任新时代的基础技术，例如去中心化身份和可验证凭证。

但是，如果没有将技术栈进行标准化的治理，以实现互操作性，那么就不可能实现这种全球规模的变化。作为Linux基金会的下属组织，新的基于IP协议栈信任基金会已经着手实现这一点。TCP/IP 的标准化成就了互联网的成功，从而实现了数十亿设备之间的互操作。该组织从中获得启发，正在定义具有4层的基于 IP 协议栈的信任的技术和治理标准。基于IP 协议栈的信任标准，包括公共工具，例如去中心化的标识符、去中心化的身份验证、代理（如数字钱包）的标准化协议、通信与数据交换协议（如用于发布和验证可验证凭证的数据流）以及应用生态系统（例如教育、金融、医疗保健等）。如果要重新考虑设计身份系统，以及如何建立与现有生态系统的信任，建议研究基于 IP 协议栈的信任标准，及其支持工具 Hyperledger Aries。

Node 泛滥
暂缓

技术都有被滥用的趋势，尤其是广为流行的技术。比如现在所见到的“Node泛滥”现象，就是一种随意或误用 Node.js 的趋势。其中有两点很突出。首先，我们经常听到这样的说法——应该使用Node.js，这样只用一种编程语言，就能完成前后端的所有代码。对此，

平台

K3s 是一个 Kubernetes 的轻量级分发版本，专为 IoT 和边缘计算打造，它移除了跟这些场景并不相关的第三方存储驱动和云服务商。

(K3s)

Materialize 是一款流式数据库，可以让你不需要复杂的数据流水线，即可完成增量计算。

(Materialize)

平台

这个四层技术和治理协议栈的目标在于，为去中心化的身份管理实现高度互操作性的形式建立了一条基线。

(基于 IP 协议栈的信任)

Node.js 非常流行。但这并不意味着它适用一切——比如它对于需要重度计算的负载来说，它会是个糟糕的选择。我们应该当心不加区分地为错误的理由使用Node.js的趋势。

(Node 泛滥)

我们还是坚持认为多语言编程是一种更好的方法，尽管我们也将JavaScript作为一等语言。其次，我们经常听到团队将性能作为选择 Node.js 的理由。这种观点已经被大量比较合理的基准数据所否定，但其来源还是有历史原因的。当初，Node.js 变得流行时，还是首个使用非阻塞编程模型的主流框架。该模型让 Node.js 非常适合IO密集型任务（我们在2012年的Node.js文章中提到了这一点）。由于单线程的特点，Node.js 从来就不是计算密集型任务的理想选择。而现在，其他平台已经拥有功能强大的非阻塞框架（其中一些已经配备既优雅又现代的 API）。所以性能已不再是选择 Node.js 的理由。

TECHNOLOGY RADAR

工具

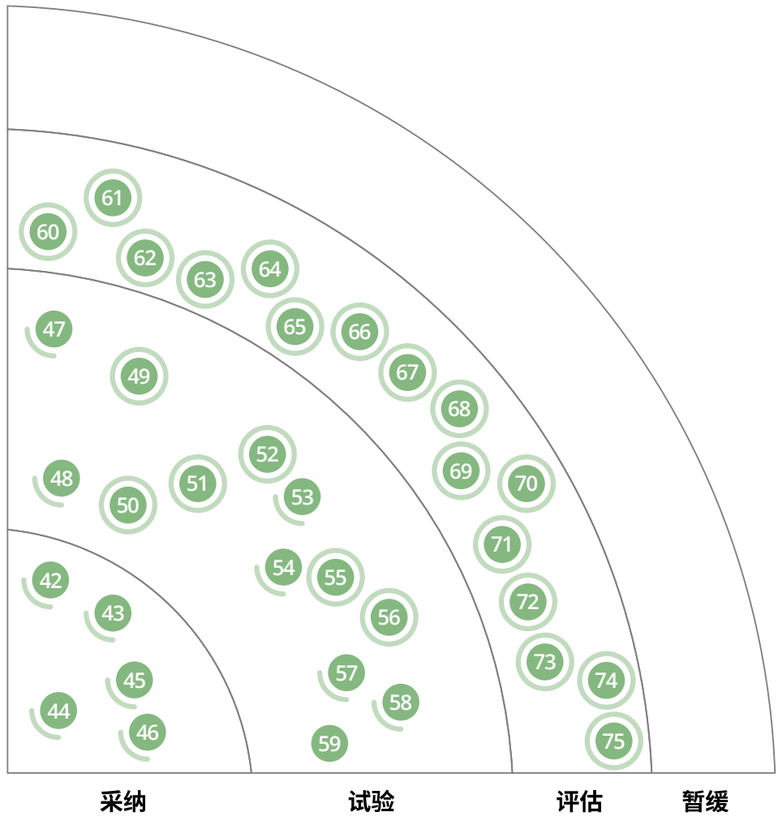
工具

Airflow
采纳

Airflow 仍然是我们广泛采用的最喜欢的开源工作流管理工具，用于构建作为有线无环图（DAGs）的数据处理流水线。这是一个蓬勃发展的领域，开源工具有 Luigi 和 Argo，厂商工具则有 Azure Data Factory 或者 AWS Data Pipeline。然而 Airflow 特别之处在于它对工作流的程序化定义，而非低代码配置文件，以及对自动化测试的支持，开源并支持多平台，对数据生态丰富的集成点还有广泛的社区支持。不过在像数据网格这样的去中心化数据架构中，Airflow 的劣势在于它是一个中心化的工作流编排。

Bitrise
采纳

Bitrise，一款用于移动应用的领域特定 CD 工具，仍然是移动开发工作流中很有用的一部分。每个团队都应该去使用它。Bitrise 能够构建，测试和部署移动应用，从开发者的笔记本电脑一直到应用商店的发布。它易于配置，并提供了一整套预构建的步骤，来满足大多数移动开发的需要。



Dependabot
采纳

在所有帮助保持依赖更新的可选工具中，Dependabot 一直是我们认为可靠的默认选择。Dependabot 跟 GitHub 的集成平滑，并能自动发送你的pull request，更新依赖到最新的版本。它能在整个组织级别启动，这样所有团队接收到这些 pull request 要容易得多。即便你没有在使用 GitHub，也仍然可以在构建流水线中使用 Dependabot库。

采纳
42. Airflow
43. Bitrise
44. Dependabot
45. Helm
46. Trivy

试验
47. Bokeh
48. Concourse
49. Dash
50. jscodeshift
51. Kustomize
52. MLflow
53. Pitest
54. Sentry
55. ShellCheck
56. Stryker
57. Terragrunt
58. tfsec
59. Yarn

评估
60. CML
61. Eleventy
62. Flagger
63. gossm
64. Great Expectations
65. k6
66. Katran
67. Kiali
68. LGTM
69. Litmus
70. Opacus
71. OSS Index
72. Playwright
73. pnpm
74. Sensei
75. Zola

暂缓

工具

Trivy 是用于容器的漏洞扫描器，它是独立类库，这样很方便配置并在本地运行扫描。

(Trivy)

jscodeshift 可以为维护大规模 JavaScript 代码库减轻痛苦。我们发现在维护设计系统时它的帮助很大。

(jscodeshift)

如果选择替代品，你可以考虑 Renovate，它支持更多的服务，包括 GitLab，Bitbucket 以及 Azure DevOps。

Helm
采纳

Helm 是用于 Kubernetes 的包管理器。它附带一个专为 Kubernetes 应用甄选过的仓库，维护于 Charts 的官方库中。之前我们提到过 Helm，现在 Helm 3已经发布，其中最显著的变化是 Helm 2的服务器端组件 Tiller 的移除。去掉 Tiller 的设计好处在于，你只能从客户端对 Kubernetes 集群作出修改，也就是说，你只能依据作为 Helm 命令行用户的权限来修改集群。我们已经在许多客户项目中使用了 Helm，它的依赖管理、模板以及钩子机制都极大简化了 Kubernetes 的应用生命周期管理。

Trivy
采纳

用来创建和部署容器的流水线，应该包含容器安全扫描这个步骤。我们的团队尤其喜欢 Trivy ——一个针对容器的漏洞扫描器。在这个领域的工具中，我们尝试过 Clair 和 Anchore Engine。跟 Clair 不一样，Trivy 不止会检查容器，而且会检查代码库中的依赖。同时，由于它是一个独立的二进制包，所以更容易在本地设置和运行。Trivy 的其他好处还有，它是开源软件，并支持 distroless containers 容器。

Bokeh
试验

Bokeh 是 Python 中最重要的库之一，通过 JavaScript 在浏览器中的渲染，它可以用于科学绘制和数据可视化。与创建出静态图像的桌面工具相比，这样的工具更易于在 web 应用探索中重用代码。Bokeh 尤其擅长这一点。这个库已经足够成熟，功能齐全。我们喜爱 Bokeh 之处在于：它保持对于作为展示层工具的专注，不会越界到比如数据聚合（参照 ggplot）或者 web 应用开发（参照 Shiny 或者 Dash）。所以当分离关注点对你来说很重要时，使用 Bokeh 就是件很愉悦的事情了。Boken 提供了web UI 小部件，并能运行于服务器模式，但你可以伺机使用或者放弃这些特性。Bokeh 很灵活，使用方式很直白，它也没有那么多依赖（比如 pandas 或者 notebooks）。

Concourse
试验

要实现可持续跨多环境构建和部署生产软件的持续交付流水线，需要一种将构建流水线和工件视为一等公民的工具。当我们初次接触 Concourse 时，就喜欢上了它简单灵活的模型，基于容器的构建原理以及它迫使你定义流水线即代码的事实。自那以后，Concourse 的可用性得到了改善，其简单的模型也经受住了时间的考验。我们的许多团队以及客户已经成功地将 Concourse 长时间用于大型流水线设置中。我们还经常借助 Concourse 的灵活性在例如硬件集成测试需要本地设置这种情况下随地工作。

Dash
试验

这一期雷达引入了一些新的工具，它们可以创建出帮助终端用户可视化并与数据交互的 web 应用。虽然还有更多简单的可视化库比如 D3，但它们在构建独立的可以操作既有数据集的分析应用上，还是减少了可观的工作量。来自 Plotly 的 Dash 在数据科学家中获得越来越多的关注，它可以用 Python 创建出功能丰富的分析应用。Dash 增强了 Python 数据类库，就像 Shiny 之于 R。这些应用有时会被认为是仪表盘，但它们可能涉及的功能，要远远超过这个名词所暗含的部分。Dash 尤其适合构建可伸缩的，随时可上线的应用，这跟同门类中另一个工具 Streamlit 不同。当你需要为商业用户呈现更复杂的分析功能时，请考虑使用 Dash，如果只是少量甚至零代码的方案，你可以选择 Tableau。

jscodeshift
试验

维护大规模的 JavaScript 代码库从来不是一件容易的事情，而迁移重大的变更更是极具挑战。在简单的场景中，带有重构能力的 IDE 也许能帮得上忙。但是，如果代码库依赖广泛，每次想要做出重大的变更时，你都不得不遍历客户端代码库，才能做出合适的更新。

这需要人工的监管并手工完成。jscodeshift，一个可以重构 JavaScript 和 TypeScript 的工具，能帮助减轻这种痛苦。它能把你的代码分析成抽象语法树（AST），并提供 API 通过不同的变换（也就是在既有的组件上添加、重命名以及删除属性）操作这棵树，然后把这棵树导出成最终源代码。jscodeshift 还附带一个简单的单元测试程序，它能用测试驱动开发的方法编写迁移 codemods。我们还发现 jscodeshift 对于维护设计系统尤其有效。

Kustomize
试验

Kustomize 是一个管理和定制 Kubernetes 清单文件的工具。它能让你在把 Kubernetes 基础资源应用到不同环境之前，选择和修补它们。它现在已经获得了 kubectl 的原生支持。我们很喜欢它，因为它可以帮助你的代码遵守 DRY 原则。与 Helm（想干的事情太多——包管理、版本管理等等）相比，我们发现 Kustomize 遵循Unix哲学：把一件事情做好，以及每个程序的输出都可以成为另一个程序的输入。

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。

Pitest
试验

传统的测试方法通常聚焦于评估我们的生产代码是不是在做该做的事情。然而，我们也可能在测试代码中犯错，比如引入了不完整或者无用的断言，导致盲目的自信。这就是变异测试的由来；它会评估测试自身的质量，发现很难意识到的临界情况。我们的团队使用 Pitest 已经有一阵子了，现在推荐在 Java 项目中使用它，用于衡量测试套件的健康程度。简而言之，变异测试会给生产代码引入一点变化，然后再次执行相同的测试；如果测试仍然是绿色的，意味着测试不够好仍然需要改进。如果你在使用 Java 以外的语言，那 Stryker 是个好的选择。

Sentry
试验

Sentry 是一款跨平台应用监控工具，并聚焦在错误报告。像 Sentry 这样的工具，区分像 ELK Stack 那样传统的日志解决方案之处在于，它们重点关注发现、研究以及修复错误上。Sentry 已经面世有一阵子了，它支持多个语言和框架。我们在许多项目中使用过 Sentry，它在跟踪错误、发现某个提交是否真实修复了问题、以及警告我们有没有出现回归问题等方面帮助很大。

ShellCheck
试验

尽管基础设施领域的工具已经得到了极大的改进，在某些情况下编写一些 shell 脚本仍然是有价值的。但是由于 shell 脚本的语法非常晦涩难懂，再加上我们现在对编写shell 脚本疏于练习，我们已经开始乐于使用 ShellCheck 来简化 shell 脚本的编写。ShellCheck 可以用于命令行，或者作为构建的一部分，甚至作为很多流行的集成开发环境的扩展来使用。它的 Wiki 页面包含了数百个 ShellCheck 可以识别的问题的详细描述，而且绝大多数的工具和集成开发环境在发现问题的时候，都提供了非常方便地方式来访问该问题对应的 Wiki 页面。

Stryker
试验

Stryker 是变异测试领域中一个较新的条目。与 Pitest 相似，Stryker 可以让你评估测试质量。我们已经在 JavaScript 项目上非常成功地使用了它，但是它也支持 C# 和 Scala 项目。Stryker 的用户友好性以及高度可定制性

工具

Kustomize 是一款可以管理和定制 Kubernetes 清单文件的工具，可以让你在应用到不同的环境之前，选择和修补你的基础 Kubernetes 资源。

(Kustomize)

Sentry 是个跨平台应用监控工具，尤其关注错误报告。它帮助了我们跟踪错误，确认某个提交有否修复了问题，并在回归问题出现时警告我们。

(Sentry)

工具

Flagger 是个有用的工具，可以调整部分流量到服务的某个新版本上——这在使用服务网格和API网关时尤其方便。

(Flagger)

很强，这使得我们能提高代码覆盖率以及对我们为客户交付的应用程序的信心。

Terragrunt
试验

我们已经广泛地使用 Terraform 来创建和管理云基础设施。根据我们的经验，在建立较大规模的基础设施时，代码往往会被拆分为若干个模块，并通过不同的方式被引入到基础设施的创建中，但是这种做法缺乏灵活性，进而会导致无法避免的代码重复，并最终会使团队停滞不前。我们通过使用 Terragrunt 来解决这个问题，它是基于 Terraform 的一个很薄的包装层，它实现了 Yevgeniy Brikman 的 Terraform: Up and Running 倡导的实践。我们发现 Terragrunt 非常有用，它鼓励多环境的版本化模块和可复用性。生命周期钩子作为另一个非常有用的特性提供了更多的灵活性。在打包方面，Terragrunt 与 Terraform 具有相同的局限性：即没有一个合适的方式来定义包以及包与包之间的依赖关系。但是我们可以使用模块并指定一个与 Git 标签相关联的版本号来解决这个问题。

tfsec
试验

安全是每个人都关心的问题，而防微杜渐永远都好过亡羊补牢。在基础设施即代码领域，Terraform 已经成为管理云环境的不二之选，而且我们现在还有了 tfsec，一个可以用来扫描 Terraform 模板并发现潜在安全问题的静态分析工具。我们的团队在 tfsec 的使用上已经积累了非常成功的经验。tfsec 非常易于安装和使用，这使得它对于任何决心通过规避安全风险避免漏洞产生的开发团队而言，都是最佳的选择。针对不同的云提供商 tfsec 都预设了扫描规则，包括 AWS 和 Azure。对使用 Terraform 的团队而言，tfsec 实在是一个福音。

Yarn
试验

Yarn 仍然是许多团队在包管理器上的默认选择。我们对 Yarn 2的面世感到兴奋，它是一次全新的发布，包括大量的更新和改进。除了可用性优化和工作区的改进外，Yarn 2还引入了 zero-installs 的概念，它可以让开发者在克隆项目后直接运行项目。但是，Yarn 2 也带来一些破坏性的变化，这让升级过程并不简单直白。而且它默认是即插即用（PnP）环境，同时不支持 React Native PnP 环境。当然，团队可以选择退出 PnP 环境，或者停留在 Yarn 1。但他们应该意识到，Yarn 1目前已经处于维护模式。

CML
评估

在之前的雷达中，我们提到将机器学习的持续交付作为一种技术，而在这一期中，我们想重点推出一个极具前景的新工具——持续机器学习（或者叫 CML），它的创造者也创造了 DVC。CML 致力于把持续集成和持续交付的最佳工程实践引入到 AI 和 ML 的团队，同时帮助你在传统的软件工程技术栈之上，组织你的 MLOps，而不是创建单独的 AI 平台。我们对他们能优先支持 DVC 感到欣慰，认为这是新工具正在急速发展的好兆头。

Eleventy
评估

只要场景允许，我们一直信赖使用静态网站生成器来避免复杂性，并提升性能。虽然 Eleventy 出现已经有年头了，但它日渐成熟，最近重获我们的关注。而我们之前的喜好比如 Gatsby.js 出现了一些伸缩性的问题。Eleventy 学习起来很快，可以很容易构建出网站。我们还喜欢它的模板功能，可以方便地创建出语义标签（自然可访问性也更高），以及它对翻页简单又智能的支持。

Flagger
评估

服务网格和 API 网关为流量路由到不同的微服务提供一个方便的方法，只要这些微服务都实现了相同的 API 接口。Flagger 利用这个特性，可以做到动态调整路由到某个新版本服务的部分流量。这对于金丝雀发布或者蓝绿部署是一项通用的技术。Flagger 和各种流行的代理（包括 Envoy 和 Kong）结合使用，可以逐步增加对于某个服务的请求，并报告负载的指标，从而为新的发布提供快速反馈。我们很高兴 Flagger 简化了这个极具价值的实践，从而可以被广泛采纳。虽然 Flagger 由 Weaveworks 赞助，但你可以独立使用 Flagger，无需捆绑使用 Weaveworks 的其他工具。

gossm
评估

当你想要连接到 AWS 上的某个服务器实例时，我们通常推荐要经由一个堡垒机，而不是直接连接。然而，仅仅因此而预置一台堡垒机的过程会让人崩溃掉，这也是为什么AWS的系统会话管理器提供了管道来更容易地连接到你的服务器。Gossm 是一个开源的 CLI 工具，它可以更方便地使用会话管理器。Gossm 让你通过 ssh 或者 scp 这样的工具，在终端直接利用会话管理器的安全机制和 IAM 策略。它还有一些 AWS CLI 不具备的能力，包括服务器发现和 SSH 集成。

Great Expectations
评估

随着 CD4ML 的兴起，数据工程和数据科学的运维方面获得了更多的关注。自动化数据治理是发展的结果之一。Great Expectations 是一款可以帮助你在数据流水线中，编制内建控件用于标记异常和质量问题的框架。就像运行在构建流水线中的单元测试，Great Expectations 在数据流水线的执行过程中作出断言。这不仅对于为数据流水线实现某种 Andon，或是确保基于模型的算法保持在训练数据决定的操作范围内，都有帮助。像这样的自动化控件可以帮助分发以及民主化数据访问和保管。Great Expectations 还配有一个探查工具，帮助理解特定数据集的质量，并设置合适的约束。

k6
评估

我们对 k6 的出现感到很兴奋，它是性能测试生态环境中比较新的一款工具，尤其注重开发者体验。k6 命令行运行器执行 JavaScript 编写的脚本，并让你配置执行时间和虚拟用户的数目。它的命令行有一系列高级特性，比如可以在测试执行完成前，让你看到当前的统计数据，动态伸缩最初定义的虚拟用户数量，甚至暂停和继续一个运行中的测试。命令行输出提供了一套带有转换器的可定制指标，能让你在 Datadog 和其他观察工具中可视化结果。为你的脚本添加 checks，可以很容易将性能测试集成到你的CI/CD流水线中去。如果要加速性能测试，可以看看它的商业版本 k6 Cloud，它提供了云伸缩以及额外的可视化能力。

Katran
评估

Katran 是一款高性能的 layer 4 负载均衡器。它并不适合所有人，但如果你需要 layer 7负载均衡器（比如 HAProxy 或者 NGINX）的替代品，或者你想要伸缩负载均衡器到两台及更多的服务器上，那我们推荐你评估一下 Katran。相对于 L7 负载均衡器上的循环 DNS 技术，或者网络工程师通常用于解决类似挑战的 IPVS 内核模型，我们把 Katran 看作一个更灵活和有效的选择。

Kiali
评估

考虑到服务网格越来越多被用来部署大量的容器化微服务，我们期待看到能自动化并简化此类架构风格相关的管理工具的出现。Kiali 就是这样的工具。Kiali 提供了一个图形化用户界面，用于观察和控制使用 Istio 部署的服务网络。我们发现 Kiali 对可视化网络中的服务拓扑结构，并理解它们彼此之间的路由流量，尤其有帮助。比如，当和 Flagger 结合使用时，Kiali 可以显示出那些路由到某个金丝雀服务发布的请求。我们特别喜欢 Kiali 的一个能力，它能让我们向某个服务网格人工注入网络错误，以测试面临网络中断时的弹力。因为配置以及在复杂的微服务网格中运行错误测试带来的复杂度，这个实践经常会被忽略掉。

LGTM
评估

编写安全的代码十分重要，但是开发人员还有很多其他事情需要考虑，不能把时间全花在这里。LGTM 不仅为开发人员提供了一道安全防护网，也是一个安全代码实践的知识库。这是一个专注于安全的静态代码分析工具，以 CodeQL 查询语言实现了（部分开放源代码的）安全编码规则。LGTM 适用于 Java、Go、JavaScript、Python、C#及C/C++，并可以将白盒安全检查集成到持续集成流水线中。LGTM 与 CodeQL 都来自于 Github 安全实验室。

工具

Great Expectations 框架提供了自动的数据治理，可以让你编制内建的控件，在数据流水线中标记异常或者质量问题。

(Great Expectations)

这是个静态代码分析工具，聚焦于由一套安全编码规则所支持的安全性上。

(LGTM)

工具

Sensei 可以让创建和分发安全代码质量指南变得很简单。

(Sensei)

Litmus
评估

Litmus 是一个低门槛的混沌工程工具。它可以用很低的代价往你的 Kubernetes 集群中注入各种各样的错误。除了随机干掉某些 Pod 之外，它提供的众多功能尤其让我们感到兴奋，比如仿真各种网络问题、CPU 问题、内存问题和 I/O 问题等。Litmus 还支持一些量身定制的试验，来仿真 Kafka 和 Cassandra 等常见服务中的错误。

Opacus
评估

差分隐私的概念最初出现在2016年的雷达中。虽然通过系统模型推断查询来破坏隐私的问题在当时可以被识别出来，但因为几乎没有补救措施，它在很大程度上还是个理论性问题。整个行业都缺少能预防它发生的工具。Opacus 是一个 Python 的新库，可以结合 PyTorch 使用，来帮助阻挡某种类型的差分隐私攻击。虽然这是很有前景的进展，但发现正确的模型以及能应用其上的数据集，一直颇具挑战。这个库仍然很新，所以我们拭目以待未来人们对它的接受度会变成什么样。

OSS Index
评估

能够识别出应用系统的依赖是否含有已知漏洞，对于开发团队来说是很重要的事情。OSS Index 可以帮助到这一点。OSS Index是一套免费的开源组件目录，以及设计用来帮助开发者识别漏洞、了解风险并确保软件安全的扫描工具。我们的团队已经通过不同的语言，把这份索引集成到流水线中，比如 AuditJS 和 Gradle plugin。它的运行速度很快，定位漏洞精准，并且几乎没有误报。

Playwright
评估

Web UI 测试一直是一片活跃的疆域。Puppeteer 的部分缔造者在转向微软后，开始将所学投用在 Playwright。它可以让你通过相同的API，为Chromium、Firefox以及 WebKit 编写测试。因为对所有主流浏览器引擎的支持，Playwright 成功获得了一些关注，并作为补丁版本包含在 Firefox 和 Webkit 中。其他工具将如何赶上，让我们拭目以待，毕竟现在 Chrome DevTools Protocol 作为自动操作浏览器的通用 API，正在获得越来越多的支持。

pnpm
评估

pnpm 是一款很有前途的 Node.js 包管理器，它相对于其他包管理器的快速和更高的效率吸引了我们的密切关注。所有依赖被保存在磁盘上的唯一位置，并链接到各自的 node_modeles目录。pnpm 还支持文件级别的增量优化，并提供了可靠的 API 来扩展和定制化，它还支持存储服务器模式，这可以更进一步加速依赖的下载。如果你的组织有大量的项目拥有相同的依赖，也许你需要关注一下 pnpm。

Sensei
评估

来自 Secure Code Warrior 的 Sensei 是一款可以很容易地创建和分发安全代码质量规则的 Java IDE 插件。在 ThoughtWorks，我们通常提倡“工具胜过规则”，也就是说，要更容易地做正确的事情，而不是遵守像检查列表那样的治理规则和过程，而 Sensei 就符合这个哲学。开发者可以创建出很容易跟其他团队成员共享的代码片段。它们被实现成针对 Java AST 的查询，可以很简单，也可以很复杂，这样的例子包括对于 SQL 注入和加密漏洞的警告，以及其他。另外一个我们喜欢的特性是，Sensei 在 IDE 中一旦发现代码变化就开始执行，因此它比其他传统的静态分析工具提供了更快的反馈。

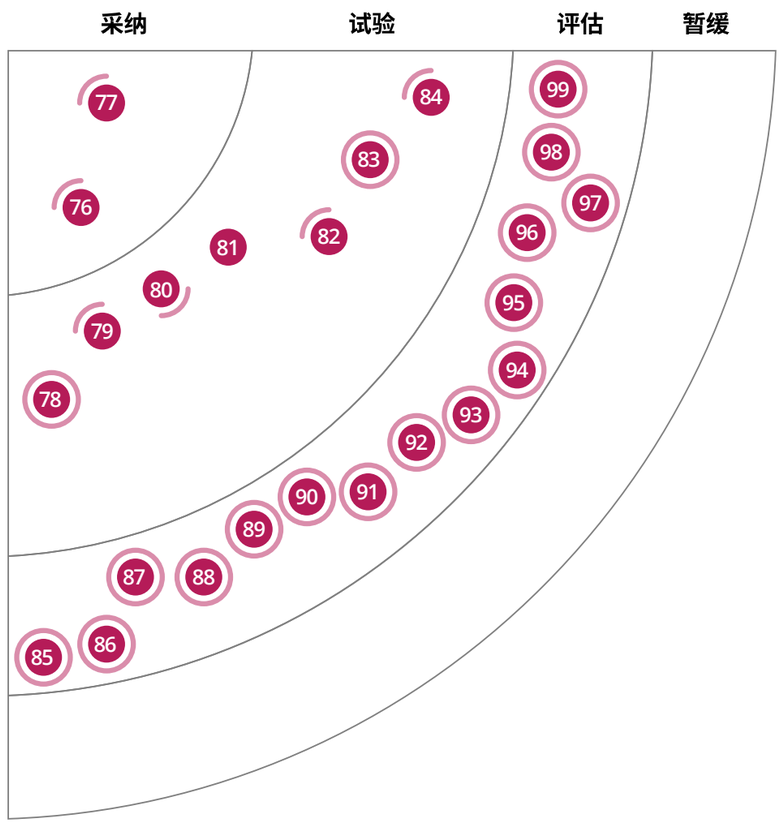
Zola
评估

Zola 是用 Rust 编写的静态网站生成器。它是一个没有依赖的可执行文件，执行速度非常快，并且支持你期望的所有常用功能，例如，支持 Sass、markdown 和热加载。我们已经成功地使用 Zola 构建了静态网站，并欣赏它的易用性。

TECHNOLOGY RADAR

语言&框架

语言&框架



Arrow
采纳

Arrow 的推广语是“Kotlin标准库的函数式伴侣”。事实证明，其所提供的现成可用的更高级别抽象的程序包非常有用，以至于我们的团队在使用 Kotlin 时，都将 Arrow 视为明智的默认选择。最近，为准备1.0版的发布，Arrow 团队进行了几处更改，包括添加了新模块，删除了一些旧模块，并将一些功能标记为过时。

jest-when
采纳

jest-when 是一个轻量级的 JavaScript 库，为 Jest 提供匹配模拟函数入参的能力。Jest 是全栈测试的好工具，而 jest-when 则可以帮助检查模拟函数接收的参数是否符合期望，这样就能够为依赖较多的模块写出更健壮的单元测试。jest-when 易于使用，并支持多种匹配器。因此在需要匹配模拟函数入参时，我们会默认选择 jest-when。

Fastify
试验

在需要用 Node.js 实现时，我们看到团队非常高兴能使用 Fastify。这个 web 框架提供了方便的请求-响应验证处理，支持 TypeScript，以及为团队提供更轻松的开发体验的插件生态系统。虽然它在 Node.js 生态中是个很好的选择，我们仍然坚持之前的建议：不要陷入 Node 过载的场景中。

Immer
试验

随着单页面 JavaScript 应用越来越复杂，以可预测的方式进行状态管理就显得愈发重要。不可变性（Immutability）可以帮我们确保应用具有一致的表现，不幸的是，JavaScript 没有提供内置的具有深层不变性的数据结构（参见 ES 记录与元组提案）。Immer（在德语中是永远的意思）是一个极小的包，它可以让你用更加便利的方式处理不可变状态。它基于写时复制（copy-on-write）机制进行工作，具有最小化的 API，而且只操作普通的 JavaScript 对象和数组。这意味着其数据访问是无缝的，不需要做大规模重构就能把不可变性引入到现有代码库中。目前，我们的很多团队都在自己的 JavaScript 代码库中使用它，相对于 Immutable.js，我们更喜欢它一些，这就是把它移入“试验”中的原因。

采纳
76. Arrow
77. jest-when

试验
78. Fastify
79. Immer
80. Redux
81. Rust
82. single-spa
83. Strikt
84. XState

评估
85. Babylon.js
86. Blazor
87. Flutter Driver
88. HashiCorp Sentinel
89. Hermes
90. io-ts
91. Kedro
92. LitElement
93. Mock Service Worker
94. Recoil
95. Snorkel
96. Streamlit
97. Svelte
98. SWR
99. Testing Library

暂缓

语言&框架

我们不再认为 Redux 是 React 应用中状态管理的缺省办法。它仍然是个有价值的框架，但相对于替代品，它会导致更冗长和难懂的代码。

(Redux)

XState 是一个简单的 JavaScript 和 TypeScript 框架，可以创建有限状态机并将其可视化为状态图。

(XState)

Redux
试验

Redux 已被移回试验环，因为我们不再将其视为 React 应用程序默认的状态管理方式。经验表明，在许多情况下 Redux 框架仍然具有一定的价值。但是与其他方法相比，Redux 会导致代码冗长难读。而引入 Redux Sagas 通常更会加剧这个问题。相对的，React 最新版本中的功能已经可以有效地管理状态，而无需引入其他框架。但需要着重强调的是，当简单的状态管理解决方案开始变得复杂时，仍然可以考虑使用 Redux，或者是 Facebook 最近发布的 Recoil。

Rust
试验

Rust 编程语言持续流行，并连续五年被开发人员们评为 Stack Overflow 的“最受喜爱”语言。我们当然也喜欢它。它是一种快速，安全且富有表现力的语言，随着它的生态系统发展，其实用性也不断提高。例如，Rust 开始用于数据科学和机器学习，并能显著提高性能。同时，用 Rust 编写的面向流的低延迟数据库 Materialize 也应运而生。

single-spa
试验

single-spa 是个 JavaScript 框架，它把多个微前端集成为一个前端应用。虽然我们告诫提防微前端混乱，借口使用微前端来混合和匹配多个框架，但 single-spa 恰恰干的就是这个。我们明白存在一些正当的场景，比如在有必要集成多个框架时，你需要升级多个微前端中某个框架到新版本。single-spa 一直是我们团队在集成微前端时的默认框架选择，他们发现它可以跟 SystemJS 工作得很好，并且能管理单个依赖的不同版本。

Strikt
试验

Kotlin 的生态在持续生长，更多的类库开始充分利用 Kotlin 语言的特性，去换掉他们的 Java 替代品。Strikt 是一个可以让你以非常流畅的风格编写测试断言的断言库。它使用 Kotlin 的 blocks 和 lambdas，帮助你的测试不那么冗长，以保持可读性。Strikt 还支持创建自定义断言，这会让你的测试更特定于领域。

XState
试验

在之前的雷达中，我们曾经提及多个状态管理的类库，但 XState 在其中显得与众不同。它是个简单的 JavaScript 和 TypeScript 框架，可以创建有限状态机并可视化为状态图。它可以跟最流行的响应式 JavaScript 框架 (Vue.js, Ember.js, React.js 以及 RxJS) 集成，并基于 W3C 标准来创建有限状态机。另外一个值得留意的特性是它可以序列化状态机的定义。在其他的上下文中 (尤其在编写游戏逻辑时) 创建有限状态机时，我们发现一件很有帮助的事情，是 XState 对状态以及可能的转换的可视化能力，通过它的 visualizer 实现起来是如此容易。

Babylon.js
评估

在几年前我们写到超越游戏的VR时，我们没有预见到 VR 解决方案会以多快以及多深的程度进入到除了视频游戏之外的领域。事后看来，我们当然看到了一些兴趣和采纳的增长，但人们对它的理解却比我们预期的要慢得多。原因之一可能是工具。Unity 和 Unreal 是两个用于开发 VR 应用的成熟又强大的引擎。我们还特别提到 Godot。然而，这些引擎跟大多数 web 和企业团队熟悉的那些工具很不同。随着我们继续探索，我们意识到基于 web 的 VR 方案已经取得巨大进展，其中对 Babylon.js 我们有相当积极的经验。Babylon.js 是用 TypeScript 编写并在浏览器中渲染出它的应用，这为许多开发团队提供了熟悉的开发体验。此外，Babylon.js 也是开源软件，成熟而且资金充足，这让它足具吸引力。

Blazor
评估

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。我们看到一个有趣的新选择 —— Blazor，它可以使用 C# 来构建交互式 Web UI。我们青睐这个开源框架，是因为它能让我们在浏览器中基于 WebAssembly 运行 C# 代码，从而利用 .NET 标准运行时环境和生态系统以及它的各种自定义库。此外，只要需要，它还可以在浏览器中与 JavaScript 代码进行双向互操作。

Flutter Driver
评估

Flutter Driver 是 Flutter 应用的集成测试库。通过 Flutter Driver，你可以在真实设备或模

拟器上测试和驱动测试套件。我们的团队一直在编写单元测试和 widget 测试，以确保 Flutter 应用的大部分业务功能已被实现。不过，对于测试实际用户交互方面，我们仍在评估。

HashiCorp Sentinel
评估

尽管我们是定义安全策略即代码的积极倡议者，但这个领域的工具一直很有限。如果你在使用 HashiCorp 产品（比如 Terraform 或者 Vault ），并且不介意为企业版本付费，那你就可以使用 HashiCorp Sentinel。事实上，Sentinel 是一门完整的编程语言，用来定义和实现基于上下文的策略决策。比如，在 Terraform 中，它可以用来在应用基础设施变更前，测试是否存在策略违规。在 Vault 中，Sentinel 可以用来定义对 API 的细粒度访问控制。这样的方法提供了类似高级编程语言提供的封装、可维护性、可读性和扩展性，自然相比较传统的声明式安全策略而言更具吸引力。Sentinel 和 Open Policy Agent 属于同一类型的工具，但它是专利所有，闭源，并且只能工作于 HashiCorp 产品中。

Hermes
评估

Hermes 是一个经过优化的 JavaScript 引擎，可在 Android 端快速启动 React Native 应用程序。JavaScript 引擎（例如 V8）具有即时（JIT）编译器，可在运行时对代码进行分析以生成优化的指令。而 Hermes 采用了另一种方法，将 JavaScript 代码提前编译（AOT）为优化的字节码。这样做的结果是，你可以获得更小的 APK 图片大小，更少的内存消耗和更快的启动时间。我们正在一些 React Native 应用中认真评估 Hermes，建议你也这样做。

io-ts
评估

我们在使用 TypeScript 时，很喜欢强类型带来的安全性。但是，将数据带入类型系统（比如调用后端服务读取数据）时，可能会引发运行时错误。io-ts 可以解决这个问题。io-ts 的编码和解码函数，将编译时类型检查与运行时消费外部数据结合在一起。同时，io-ts 也可以用作自定义的类型守卫。我们认为这是一个绝妙的解决方案。

Kedro
评估

过去我们讨论过在数据科学项目中应用好的工程实践的改进 工具。Kedro 是另一个很好的补充。它是用于数据科学项目的开发工作流框架，为构建生产用数据和机器学习管道带来了标准化的方法。我们很欣赏其对软件工程实践和良好设计的关注，它强调测试驱动开发、模块化、版本控制以及良好的代码实践，比如将凭证排除在代码库之外。

LitElement
评估

自我们于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，它减少了对许多常见引用文件的需要，并且使编写 Web 组件变得更加容易。我们已经在项目中使用它取得了早期的成功，并且很高兴看到这项技术的成熟。

Mock Service Worker
评估

Web 应用，特别是企业内部应用，通常分为两个部分。用户界面和小部分业务逻辑运行在浏览器中，而大部分业务逻辑、认证和持久化工作运行在服务器中。这两部分一般会通过在 HTTP 上传输 JSON 进行通讯。但请不要误认为这些端点（endpoint）是真正的 API，不，它们只是当应用要穿越两个运行环境时的实现细节而已。与此同时，它们也提供了一个合适的“接缝”，以便独立测试这些“零件”。当测试 JavaScript 部分时，可以通过像 Mountebank 这样的工具在网络层对其服务端进行打桩（stub）和模拟（mock）。还有一种方式是在浏览器中拦截各种请求。我们很喜欢这种由 Mock Service Worker 带来的方式，因为 Service Worker 是前端开发人员比较熟悉的抽象层。这种方法可以简化设置，并且执行得也更快。不过，由于这些测试没有测到真正的网络层，所以你还要实现一些端到端测试，才能得到一个健康的测试金字塔。

Recoil
评估

越来越多正在使用 React 的团队，都开始重新评估他们对于状态管理的选项，这也是我们在重新考虑 Redux 时提到的。现在，React 的缔造者 Facebook，发布了 Recoil，一个用于管理状态的全新框架。它源自一个要应付大量数据的内部应用。即便现在对 Recoil 没

语言&框架

Sentinel 是个完整的编程语言，可以定义和实现基于上下文的政策决定。

(HashiCorp Sentinel)

Kedro 是一款用于数据科学的开发工作流框架，它为构建可随时上线的数据和机器学习流水线提供了标准化的方法。

(Kedro)

语言&框架

Snorkel 由斯坦福大学创建，可以让我们以编程地方式为用于训练机器学习算法的大量数据集打标签。

(Snorkel)

有充分的实践经验，我们仍然能预见它的潜力和前景。API 简单易学，感觉像是惯用的 React。不像其他方法，Recoil 为应用间共享状态提供了一个有效而灵活的办法：它支持从派生数据和查询动态地创建状态，并在不损害代码分割的情况下，实现整个应用范围内的状态监控。

Snorkel
评估

现代机器学习模型非常复杂，并且需要大量标记训练数据集进行学习。Snorkel 始于斯坦福 AI 实验室，开发者们意识到手动标记数据非常昂贵，而且通常不可用。Snorkel 使我们可以通过创建标记函数的方式自动对训练数据进行标记。Snorkel 采用监督式学习技术来评估这些标记函数的准确性和相关性，然后重新赋权并组合输出标签，从而生成高质量的训练标签。由此，Snorkel 的创建者们推出了一个名为 Snorkel Flow 的商业平台。尽管 Snorkel 本身已不再被积极研发，但它在使用弱监督方法标记数据方面的思想仍具有重要意义。

Streamlit
评估

Streamlit 是 Python 编写的开源应用框架，数据科学家用其来构建好看的数据可视化应用。Streamlit 专注于快速原型设计，并且支持各种不同的可视化库(包括 Plotly 和 Bokeh)，因此在Dash等竞品中脱颖而出。对于需要在实验周期中快速展示的数据科学家来说，Streamlit 是一个可靠的选择。我们在一些项目中使用它，并且只需要花费很少的工作量就能把多个交互式可视化放在一起。

Svelte
评估

我们不断看到新的前端 JavaScript 框架出现，其中 Svelte 作为一个前途无量的新组件框架脱颖而出。与其他使用虚拟 DOM 的框架不同，Svelte 将代码编译为单纯的无框架 JavaScript 代码，这些代码可以直接操作更新 DOM。不过，它只是一个组件框架；如果你打算构建功能丰富的应用程序，可以考虑将 Sapper 与 Svelte 一起进行评估。

SWR
评估

SWR 是用于获取远程数据的 React Hooks 库，它实现了 stale-while-revalidate HTTP 缓存策略。SWR 首先从缓存（过时的）中返回数据，然后发送获取请求（再验证）并最终用更新的响应数据刷新数值。组件因此持续而且自动地获得一个数据流，先是过时的，然后是刷新过的。我们的开发者在使用 SWR 时获得了很好的开发体验，并且因为数据总是显示在屏幕上，从而显著提升用户体验。然而，我们提醒团队，只有当应用程序返回过时数据是合适的时候，才能使用 SWR 缓存策略。要注意，HTTP 通常要求缓存要用最新的响应返回给请求，只有在需要非常慎重的场景下，才会允许返回过时的响应数据。

Testing Library
评估

Testing Library 是用于在众多框架（如 React，Vue，React Native 以及 Angular）中测试应用程序的软件包集合。这组库通过鼓励测试用户行为，而非实现细节的方式（如在特定时间点中 UI 存在的元素），来帮助你以用户为中心的方式测试 UI 组件。这种思维方式的好处之一在于测试更加可靠，这也是我们所认为的主要区别。我们建议在任意框架中测试 Web 程序时，都使用该系列工具。尽管仅有 React Testing Library 和 Angular Testing Library 的直接使用经验，我们仍对 Testing Library 印象深刻。

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25年以来，ThoughtWorks已经从一个小团队，成长为现在拥有超过7，000人，分布于全球14个国家、拥有43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/cn/radar

#TWTechRadar