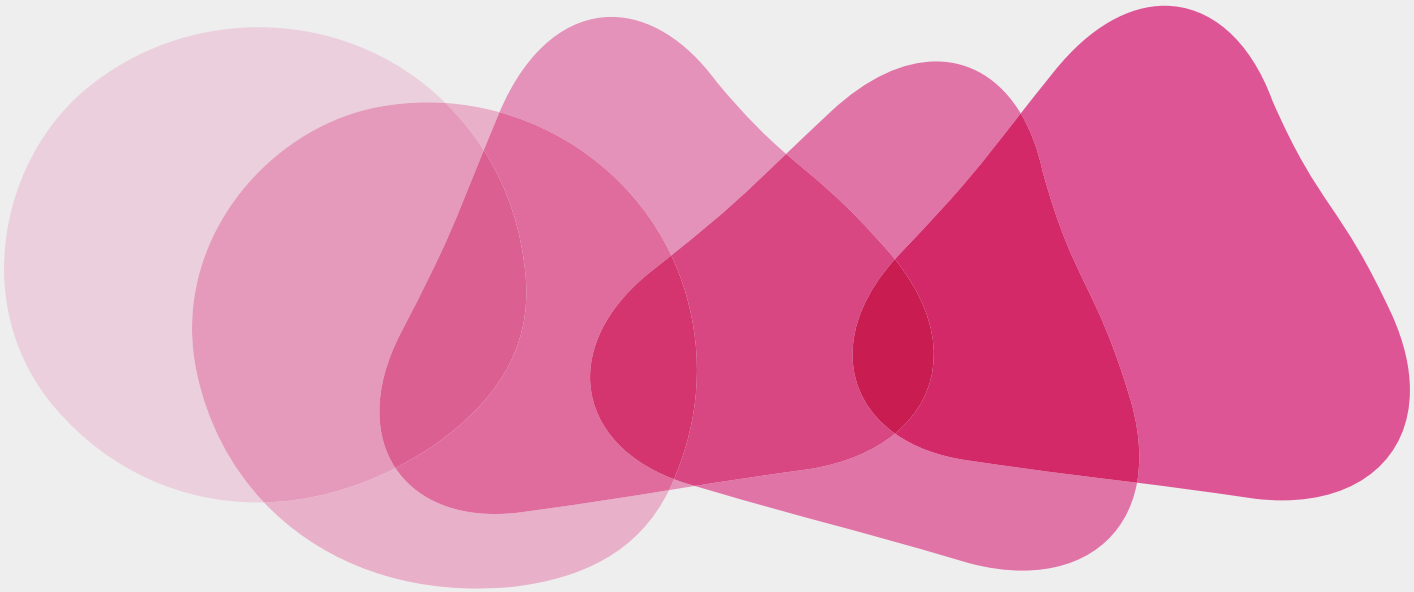
ThoughtWorks®

TECHNOLOGY RADAR *VOL.19*

洞察构建未来的技术和趋势



thoughtworks.com/cn/radar

#TWTechRadar

贡献者

技术雷达由 ThoughtWorks 技术顾问委员会筹备，其人员组成为：

Rebecca Parsons (CTO) | Martin Fowler (首席科学家) | Bharani Subramaniam | Camilla Crispim | Erik Doernenburg
Evan Bottcher | Fausto de la Torre | 徐昊 | Ian Cartwright | James Lewis | Jonny LeRoy
Ketan Padegaonkar | Lakshminarasimhan Sudarshan | Marco Valtas | Mike Mason | Neal Ford
王妮 | Rachel Laycock | Scott Shaw | 刘尚奇 | Zhamak Dehghani

技术雷达中国区技术咨询顾问组：

包欢	边晓琳	陈璐	樊卓文	韩盼盼	黄进军	库天锡	李旭飞	梁凌锐
刘先宁	马伟	闵锐	阮焕	汪志成	王柏元	王瑞鹏	魏喆	邬倩
吴邦	吴兵华	伍斌	向博	徐瑾	徐培	鄢倩	严嘉阳	杨乐涵
杨政权	姚琪琳	易维利	袁慎建	张凯峰	张扬	张羽辰		

本期技术雷达是基于ThoughtWorks技术顾问委员会在2018年10月亚特兰大会议上的讨论所得出的。

最新动态

本期精彩集锦

粘性十足的云平台

云提供商知道他们正在严峻的市场中进行竞争。为了获胜，他们需要吸引用户注册并长期留住他们。因此，为了保持竞争力，他们在新增产品特性上你争我抢，使得彼此不相上下。这一点可以从本期雷达试验环中以下云提供商的排名看出：AWS、Google Cloud Platform 和 Azure 。然而，一旦用户注册，这些云提供商就倾向于与用户建立尽可能高的粘性，以阻止他们使用其他提供商的服务。这通常表现为其云服务会与其服务和工具套件紧密集成。用户只有继续使用其云服务，才能获得更好的开发者体验。通常在用户决定是否将其部分或全部工作负载移动到另一朵云上，或发现云服务的使用和账单多到失控时，就能明显感觉到这种粘性。我们鼓励客户使用 架构适应度函数度量成本 的技术来监控运维成本，并将其作为衡量云供应商粘性的指标。或者使用Kubernetes和容器，并通过运用基础设施即代码来提升工作负载的可移植性，降低切换到另一个云提供商的成本。在本期雷达中，我们还会介绍两个新的云基础设施自动化工具：Terragrunt和Pulumi。虽然我们支持通过粘性的高低来评估云提供商的新产品，但提醒你不要落入只使用通用云服务功能的陷阱。根据我们的经验，创建和维护与云无关的抽象层的开销，会超出退出某个特定云提供商的花费。

挥之不去的企业级应用反模式

无论技术如何快速变化，一些企业仍然想方设法地重新实现过去的反模式。雷达中的许多暂缓条目都在揭穿一些“新瓶装旧酒”的老把戏。比如：用Kafka重新创建ESB的反模式、分层式微服务架构、Data-hungry packages、过度庞大的API网关、Low-code 平台和一些其他有害的旧实践。一如既往的根本问题，是如何在隔离和耦合之间取得平衡：我们隔离组件，使其在技术角度便于管理。但是我们也需要协调组件，使其有助于解决业务问题。这就产生了某种形式的耦合。因此，上述旧模式就不断重新冒出来。新的架构和工具为解决这些问题提供了适当的方法，但这需要刻意去理解如何正确地使用它们，而不仅仅是使用崭新的技术去重新实现旧模式。

持之以恒的工程实践

随着技术创新步伐加快，新技术的发展呈现出一种从爆发到沉淀不断循环的模式。每当能够颠覆我们对软件开发固有认知的新技术出现时，都会引起业界的争相追捧，容器化、响应式前端和机器学习都是很好的例子。这时技术处在爆发阶段。然而，只有在明确如何与长期以来的工程实践（持续交付、测试、协作等）相结合之后，新技术才能真正的发挥功效，并进入沉淀阶段，为下一次爆发性扩张打下坚实的基础。在沉淀阶段，我们尝试在新技术的背景下应用实践，比如进行全面的自动化测试以及创建脚本代替重复操作，通常也会创造出新的开发工具。虽然表面看来技术创新是行业发展的唯一驱动力，但事实上，创新与持之以恒的工程实践相结合才是我们不断进步的基础。

速度 = 距离 / 时间

通常，我们会选取本期雷达中部分共性条目的精彩集锦展现在雷达主题中，但本主题涉及自技术雷达诞生以来出现过的所有条目。我们发现（并通过一些调研证明）雷达条目停留在雷达上的时间正在缩减。当我们在10年前启动技术雷达时，如果某个条目在雷达上的位置不再移动，它依然会保留两期（大约一年）时间，之后才会自动移出雷达。然而正如标题中的公式所说，速度 = 距离 / 时间：软件开发生态系统中的变化一直在持续加速。在时间保持不变（依然是每年发布两次）的前提下，雷达中技术创新所跨越的距离明显地增大了。这为精明的读者提供了显著的证据：技术变革的步伐正在不断加快。我们不仅看到雷达中的各个象限在加速变化，也看到了客户对新兴的及多样化的技术选择所表现出的兴趣。因此我们将改变传统的默认模式：雷达不再默认保留其上的条目，它们是否出现在雷达上完全取决于它们当前的价值。我们在深思熟虑后做出了这项改变，并认为只有这样才能更好地跟上技术生态系统中前所未有的狂热变化节奏。

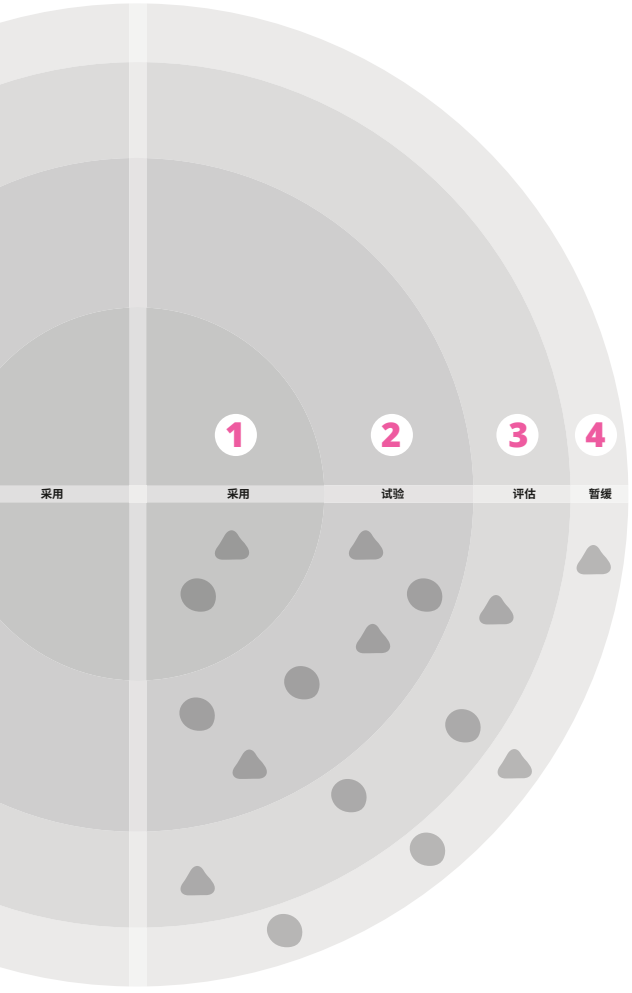
关于技术雷达

ThoughtWorks人酷爱技术。我们对技术进行构建、研究、测试、开源、记述，并始终致力于对其进行改进-以求造福大众。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(TAB)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，为从开发人员到CTO在内的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。

这个雷达是图形性质的，把各种技术项目归类为技术、工具、平台和语言及框架，如果某个条目可以出现在多个象限，我们选择看起来最合适的象限。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。

要了解关于雷达的更多背景，请点击：thoughtworks.com/cn/radar/faq



雷达一览

1 采用

我们强烈主张业界采用这些技术。如果适合我们的项目，我们就毫不犹豫地使用。

2 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。

3 评估

值得研究一番的技术，以确认它将对你产生何种影响。你应该投入一些精力来确定它是否会对你所在的组织产生影响。

4 暂缓

别用这项技术启动任何新项目。在已有项目上使用它没有坏处，但是想在新开发的项目上使用这个技术的话需要三思而行。

三角形图标 三角形图标表示新出现或位置发生过显著变化的条目

圆形图标 圆形表示没有变化的条目

我们感兴趣的技术实在太多，远不是如此大小的文档能合理容纳的。如果一个图标在技术雷达上都没有移动，我们就把它略去。以减少混乱，并为新条目腾出空间，但并不表示我们不再关心它。

THE RADAR

技术

采用
1. Event Storming

试验
2. 1% canary *NEW*
3. Bounded Buy *NEW*
4. Crypto shredding *NEW*
5. Four key metrics *NEW*
6. Multi-account cloud setup *NEW*
7. Observability as code *NEW*
8. Risk-commensurate vendor strategy *NEW*
9. Run cost as architecture fitness function *NEW*
10. Secrets as a service *NEW*
11. Security Chaos Engineering
12. Versioning data for reproducible analytics *NEW*

评估
13. Chaos Katas *NEW*
14. Distroless Docker images *NEW*
15. Incremental delivery with COTS *NEW*
16. Infrastructure configuration scanner
17. Pre-commit downstream build checks *NEW*
18. Service mesh

暂缓
19. "Handcranking" of Hadoop clusters using config management tools *NEW*
20. Generic cloud usage
21. Layered microservices architecture *NEW*
22. Master data management *NEW*
23. Microservice envy
24. Request-response events in user-facing workflows *NEW*
25. RPA *NEW*

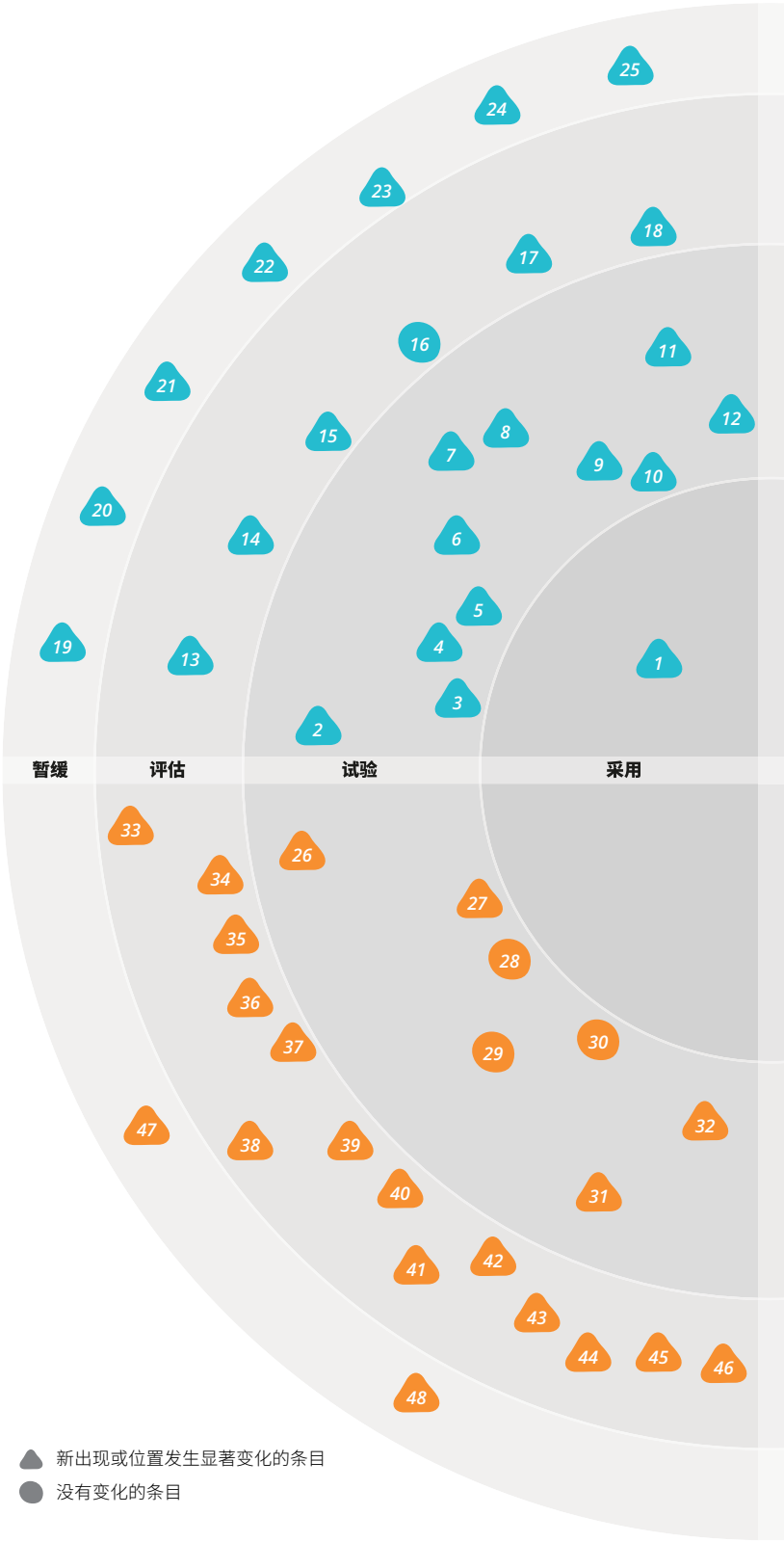
平台

采用

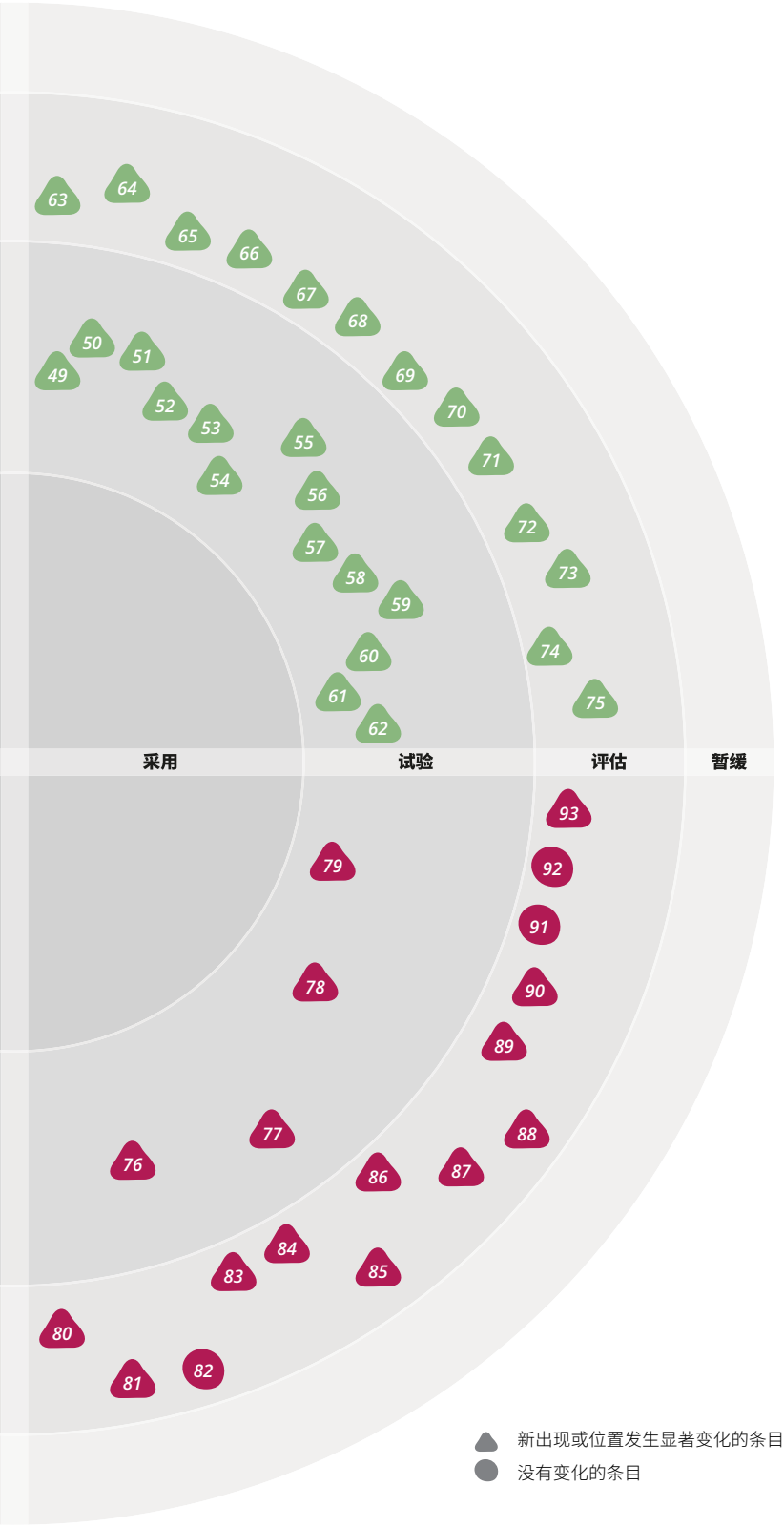
试验
26. Apache Atlas *NEW*
27. AWS
28. Azure
29. Contentful
30. Google Cloud Platform
31. Shared VPC *NEW*
32. TICK Stack

评估
33. Azure DevOps *NEW*
34. CockroachDB *NEW*
35. Debezium *NEW*
36. Glitch *NEW*
37. Google Cloud Dataflow *NEW*
38. gVisor *NEW*
39. IPFS *NEW*
40. Istio *NEW*
41. Knative *NEW*
42. Pulumi *NEW*
43. Quorum *NEW*
44. Resin.io *NEW*
45. Rook *NEW*
46. SPIFFE *NEW*

暂缓
47. Data-hungry packages *NEW*
48. Low-code platforms *NEW*



THE RADAR



工具

采用

试验

49. acs-engine *NEW*
50. Archery *NEW*
51. ArchUnit
52. Cypress
53. git-secrets *NEW*
54. Headless Firefox
55. LocalStack *NEW*
56. Mermaid *NEW*
57. Prettier *NEW*
58. Rider *NEW*
59. Snyk *NEW*
60. UI dev environments *NEW*
61. Visual Studio Code
62. VS Live Share *NEW*

评估

63. Bitrise *NEW*
64. Codefresh *NEW*
65. Grafeas *NEW*
66. Heptio Ark *NEW*
67. Jaeger *NEW*
68. kube-bench *NEW*
69. Ocelot *NEW*
70. Optimal Workshop *NEW*
71. Stanford CoreNLP *NEW*
72. Terragrunt *NEW*
73. TestCafe *NEW*
74. Traefik *NEW*
75. Wallaby.js *NEW*

暂缓

语言&框架

采用

试验

76. Jepsen
77. MMKV *NEW*
78. MockK *NEW*
79. TypeScript

评估

80. Apache Beam *NEW*
81. Camunda *NEW*
82. Flutter
83. Ktor *NEW*
84. Nameko *NEW*
85. Polly.js *NEW*
86. PredictionIO *NEW*
87. Puppeteer *NEW*
88. Q# *NEW*
89. SAFE stack *NEW*
90. Spek *NEW*
91. troposphere
92. WebAssembly
93. WebFlux *NEW*

暂缓

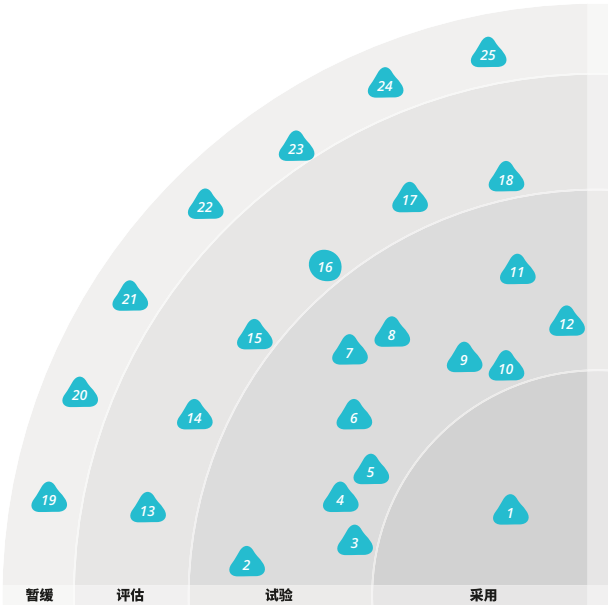
技术

采用
1. Event Storming

试验
2. 1% canary *NEW*
3. Bounded Buy *NEW*
4. Crypto shredding *NEW*
5. Four key metrics *NEW*
6. Multi-account cloud setup *NEW*
7. Observability as code *NEW*
8. Risk-commensurate vendor strategy *NEW*
9. Run cost as architecture fitness function *NEW*
10. Secrets as a service *NEW*
11. Security Chaos Engineering
12. Versioning data for reproducible analytics *NEW*

评估
13. Chaos Katas *NEW*
14. Distroless Docker images *NEW*
15. Incremental delivery with COTS *NEW*
16. Infrastructure configuration scanner
17. Pre-commit downstream build checks *NEW*
18. Service mesh

暂缓
19. "Handcranking" of Hadoop clusters using config management tools *NEW*
20. Generic cloud usage
21. Layered microservices architecture *NEW*
22. Master data management *NEW*
23. Microservice envy
24. Request-response events in user-facing workflows *NEW*
25. RPA *NEW*



快速市场响应能力是组织进行微服务转型的主要驱动之一。然而只有沿长期业务领域边界对服务（及其支持团队）进行清晰划分时，这种期望才可能实现。否则，现实需求只有在跨组织和跨服务的通力合作才下能完成，这自然会在规划产品路线图时产生冲突。良好的领域模型设计是解决此问题的方案，**事件风暴（EVENT STORMING）**也迅速成为我们最喜爱的方法之一，它使我们能够迅速识别问题领域中的关键概念，并用最好的方式与各方利益相关人制定解决方案。

快速反馈是我们构建软件的核心价值之一。很多年来，我们使用金丝雀发布来鼓励对于新版本软件要尽早反馈，同时对选定用户做增量发布以降低风险。与这个技术相关的问题之一就是如何划分用户。对非常小的一部分用户（比如1%）作金丝雀发布是变更的催化剂。从一小部分用户开始可以让团队逐渐熟悉这项技术，而快速捕获用户反馈可以让不同团队观察新发布的影响，从中学习并按照需要调整方向——这是工程师文化里很宝贵的变化。我们称之为万能的**1% 金丝雀（1% CANARY）**。

大多数没有足够资源进行软件定制化开发的组织，通常会选择一些“开箱即用”的或基于SaaS平台的解决方案来直接满足需求。我们建议这些组织首先要设计出清晰的目标能力模型，然后使用被称为限界购买的策略，即只选择模块化、解耦的，且只包含于单一业务能力（Business Capability）的限界上下文（Bounded Context）中的厂商产品。
(Bounded Buy)

大多数没有足够资源进行软件定制化开发的组织，通常会选择一些“开箱即用”的或基于SaaS平台的解决方案来直接满足需求。但是最近我们注意到，这些解决方案的范围正在急剧扩张，与业务的各个环节纠缠在一起。这种扩张模糊了集成边界，导致越来越难以估计软件变更的复杂度，变更速度也越来越慢。为了降低这种风险，我们建议这些组织首先要设计出清晰的目标能力模型，然后使用被称为**限界购买的策略(BOUNDED BUY)**，即只选择模块化、解耦的，且

只包含于单一业务能力(Business Capability)的限界上下文(Bounded Context)中的厂商产品。我们应该将这种对模块化和独立交付能力的要求,加入对供应商选择的验收标准中去。

对敏感数据保持适当的控制是相当困难的,尤其是在出于对数据备份和恢复的目的而将数据复制到主数据系统之外的时候。**密钥粉碎(CRYPTO SHREDDING)**是通过故意覆盖或删除用于保护该数据的加密密钥来使敏感数据无法读取的做法。例如,可以使用随机密钥对数据库中客户个人详细信息表的所有记录进行一对一加密,然后使用另一张表来存储密钥。如果客户行使了"被遗忘的权利",我们可以简单地删除相应的密钥,从而有效地"粉碎"加密数据。当我们有信心对小规模加密密钥集合维持适当控制,但对较大数据集的控制信心不足时,此项技术非常有效。

DevOps状态报告和Accelerate的核心点都支持了软件交付性能的四个关键指标:前置时间,部署频率,平均恢复时间(MTTR)和变更失败百分比。

(Four key metrics)

2014年首次发布的DevOps状态报告指出,高效团队创造了高效的组织。最近,该报告背后的团队编写了Accelerate一书,描述了他们在报告中使用的科学方法。两份材料的核心点都支持了软件交付性能的**四个关键指标(FOUR KEY METRICS)**:前置时间,部署频率,平均恢复时间(MTTR)和变更失败百分比。作为帮助许多组织转型的咨询公司,反复使用这些指标测量,可以帮助组织确定他们是否在提高整体效能。每个指标都创造了一个良性循环,并使团队专注于持续改进:缩短交付周期,减少浪费的活动,从而使你可以更频繁地部署;部署频率迫使你的团队改进他们的实践和自动化流程;通过更好的实践,自动化和监控可以提高你从故障中恢复的速度,从而降低故障频率。

自助使用、按需付费是当代云计算重要的特性(和优势)。但是,当使用单个账号去部署大规模的服务时,事情就变得非常复杂。我们必须为该账户设置使用规则和流程,往往这些流程包含大量的审批步骤,从而降低我们的效率。**设置多云账号(MULTI-ACCOUNT CLOUD SETUP)**是一个不错的方案,我们可以申请多个账号,并且为每个团队指定一个账号。当然这个方案增加了其他方面的复杂度,例如共享计费、不同 VPC 之间的通信、与云供应的关系管理等。但是,也会提高我们的开发效率与安全性,审计只用于单服务的账号相对容易,在发生数据泄漏时,我们所受到的影响也相对较小。设置多云账号可以降低我们对云供应商的粘性,因为单服务账号能提供清晰的边界,所以便于整体迁移到另一个云提供商。

可观测性是运转分布式系统与微服务架构必不可少的一部分。我们强烈建议使用代码来配置可观测性生态系统,称为可观测性即代码,并且采取基础设施即代码的方式搭建其基础设施。

(Observability as code)

可观测性是运转分布式系统与微服务架构必不可少的一部分。我们依赖不同的系统输出来推断分布式组件的内部状态,比如分布式追踪、日志聚合、系统指标等,进而诊断问题所在,并找到根本原因。可观测性生态系统的一个重要方面就是监控——可视化以及分析系统的输出——并且在检测到异常时报警。传统的监控报警配置,都是通过图形界面的操作完成。这种方法导致控制面板页的配置不可重复,从而无法持续测试和调整报警,来避免报警疲劳或错过重要的报警,进而偏离组织的最佳实践。我们强烈建议使用代码来配置可观测性生态系统,称为**可观测性即代码(OBSERVABILITY AS CODE)**,并且采取基础设施即代码的方式搭建其基础设施。因此,在选择提供可观测性的工具时,要选择支持通过代码版本控制进行配置,并能通过基础设施持续交付流水线执行API或命令行的产品。可观测性即代码,是基础设施即代码中经常被遗漏的一部分,我们认为这一点非常重要,需要明确指出。

通常,为了将风险外包给供应商,企业在其最关键和风险最高的系统上往往锁定某个特定供应商,以求风险最低。不幸的是,这会让企业可以选择的解决方案变得更少,灵活性也更低。相反,企业应该在业务风险最高的地方保持最大的供应商独立性。我们看到一种新的**风险相称的供应商策略(RISK-COMMENSURATE VENDOR STRATEGY)**正在兴起,它鼓励在高度关键系统中维持其供应商的独立性。而那些相对不太重要的业务可以利用供应商提供的成熟解决方案,这可以让企业更容易承受失去该供应商所带

来的影响。随着主流云提供商扩展其服务范围，这种权衡变得愈发重要。例如，在开发工作开始时，使用AWS Secret Management Service可以提高效率，并且还有易于集成AWS云生态的好处。但相比于自己实现密钥管理解决方案，例如使用Vault时，AWS的方案难以迁移到不同的云供应商。

随着软件架构及其业务的演进，我们理应密切关注应用的运行成本，但发现并非所有的组织都如此。尤其是在使用无服务器架构时，开发者们认为无服务器架构会更便宜，因为他们只需按消耗的计算时间付费。然而几家主要的云服务提供商在热门的无服务器函数上定价十分精明，虽然无服务器在快速迭代上很有优势，但与专属云（或内部私有云）相比，它的开销可能随着使用量迅速增长。我们建议团队将应用的**运行成本纳入架构适应度函数（RUN COST AS ARCHITECTURE FITNESS FUNCTION）**来考量，这意味着：追踪并权衡应用的运行成本与交付价值；当它们之间产生较大出入时，就需要考虑改进软件架构了。

我们长期以来都警示人们，要抵制在代码仓库中保存密钥信息的诱惑。在往期雷达中，我们曾推荐了将代码和密钥管理解耦，但如今我们发现了一系列提供**密钥即服务（SECRETS AS A SERVICE）**的好工具。通过这些工具，系统能够从外部服务中获取秘密信息，而不是使用硬编码或将其配置在运行环境中的方式。例如来自HashiCorp的Vault这样的工具帮助你将密钥与应用程序分开管理，同时有助于你强制实施一些安全政策，例如周期性密钥轮换。

虽然在本期雷达中大部分条目都是全新的，但我们认为依然值得继续为**安全混沌工程（SECURITY CHAOS ENGINEERING）**的实用性提名。本期雷达中我们将其移动到了试用阶段，因为使用此技术的团队确信他们的安全策略足以应对常见的安全故障模式。不过，在使用这种技术时需要谨慎，我们不希望我们的团队对安全问题变得不再敏感。

当涉及到大规模数据分析或机器智能问题，基于不同数据集合和参数进行可重复性分析就变得非常有价值。

(Versioning data for reproducible analytics)

当涉及到大规模数据分析或机器智能问题，基于不同数据集合和参数进行可重复性分析就变得非常有价值。

为了实现可重复性分析，数据和模型（包括算法方案，参数和超参数）需要进入版本控制。因为数据量的缘故，**可重复性分析的版本化数据（VERSIONING DATA FOR REPRODUCIBLE ANALYTICS）**比起版本化模型更加棘手。一些像DVC这样的工具采用类似git工作流的方式，让用户提交和推送数据文件到远程的云存储，从而实现数据的版本化。这极大方便了协作者拉取特定版本的数据来进行可重复性分析。

Chaos Katas将Kata的方法论与Chaos Engineering的相关技术（具体指在受控环境中模拟故障和停机）进行结合，对工程师进行系统化教学和培训。.

(Chaos Katas)

CHAOS KATAS是一项为基础设施和平台工程师提供技能培训和提升的技术。它将Kata的方法论与Chaos Engineering的相关技术（具体指在受控环境中模拟故障和停机）进行结合，对工程师进行系统化教学和培训。这里的Kata是指触发受控故障的代码模式，它允许工程师发现问题，恢复故障，开展事后分析并找到根本原因。工程师通过重复执行Kata能帮助他们真正掌握新的技能。

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。通过这项技术，镜像占用的空间只包含应用本身以及它的资源和语言运行时依赖，而不包含操作系统。

(Distroless Docker Images)

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。在此期雷达中，我们欣喜地发现了一个由Google开发，用来解决容器安全性和大小问题的技术，名叫**DISTROLESS DOCKER IMAGES**。通过这项技术，镜像占用的空间只包含应用本身以及它的资源和语言运行时依赖，而不包含操作系统。它的优势包括减少其它因素对应用安全扫描的干扰、减小安全攻击面、降低漏洞修复开销，以及减小镜像大小以获得更高性能。Google 已经为不同的语言发布了一系列的distroless

容器镜像。你可以通过 Google 的构建工具Bazel来创建 distroless 应用镜像，它支持使用Distroless的语法规则或者简单地采用多阶段Dockerfiles的方式来创建Distroless 容器。值得注意的是 Distroless 容器默认情况下不提供 shell 来进行调试，但你能很方便地在网上找到包含 busybox shell 的 distroless 容器可调试版本。

作为敏捷领域的先驱和领导者，ThoughtWorks一直以实际行动践行增量交付实践，同时建议我们的客户从“增量交付”的视角来审视他们已有的软件。但是这个过程通常很困难，因为大多数供应商都采用一次性交付的方式，这就会涉及到大量的数据迁移。然而最近我们成功的实践了**与供应商一起增量交付(INCREMENTAL DELIVERY WITH COTS (commercial off-the-shelf))**，以增量的方式向较小的用户群发布特定的业务流程。我们建议您评估是否可以将此实践应用于您选择的供应商软件，以帮助减少一次性交付的风险。

为了降低风险，采用云策略时大多数组织都默认采用严格的、集中式的配置管理方式，但这也导致了严 重的生产力瓶颈。另外一种做法则是允许团队自己管理自己的配置，并使用一种工具来确保配置的安全性。
(Infrastructure configuration scanner)

很长时间以来，我们一直在建议交付团队对整个技术栈负责，其中也包括对基础设施负责。这意味着，在以安全可靠、合规的方式配置基础设施这方面，交付团队需要承担起 更多的责任。为了降低风险，采用云策略时大多数组织都 默认采用严格的、集中式的配置管理方式，但这也导致了严重的生产力瓶颈。另外一种做法则是允许团队自己管理自 己的配置，并使用**INFRASTRUCTURE CONFIGURATION SCANNER**这种方式来确保配置的安全性。Watchmen是个很有意思的工具，它旨在为由交付团队自主拥有和运营 AWS 账户配置提供基于规则驱动的扫描。Scout2是另一个配置扫描的例子，它可以提供安全合规的支持。

在更复杂的架构和部署中，可能并不能容易立即发现某个依赖正在检入的代码的构建已经损坏。开发人员在尝试修复已损坏的构建时，会发现自己就像在打移动靶，因为该构建会不断被其上游依赖所触发。**下游构建的提交前检查(PRE-COMMIT DOWNSTREAM BUILD CHECKS)**是一个很简单的技术：用一个提交前（pre-commit）或推送前（pre-push）脚本来检查这些下游构建的状态，并事先警告开发人员某个构建已经坏掉了。

随着大型组织向拥有和运营自己的微服务的更自主的团队过渡，他们如何在不依赖集中托管基础架构的情况下确保这些服务之间的必要一致性和兼容性?为了有效地协同工作，即使是自主的微服务也需要与一些组织标准保持一致。**SERVICE MESH** 提供一致的发现、安全性、跟踪、监控和故障处理，而无需共享API网关或ESB等设施。典型的实现是将每个服务进程和轻量级反向代理进程一起部署，反向代理进程可能在单独的容器中。这些代理与服务注册表，身份提供者，日志聚合器和其他服务进行通信。服务互操作性和可观测性是通过此代理的共享实现而不是共享运行时实例获得的。我们提倡采用去中心化的微服务管理方法已经有一段时间了，并很高兴看到这种一致模式的出现。Linkerd 和 Istio 等开源项目将逐步成熟，这使得 service mesh 更容易实现。

当组织选择 vanilla Hadoop 或Spark 发行版而不是某个供应商的发行版时，他们必须决定如何配置和管理集群。有时，我们会看到**使用配置管理工具进行“手摇式”启动的 Hadoop 集群("HANDCRANKING" OF HADOOP CLUSTERS USING CONFIG MANAGEMENT TOOLS)**，这些工具有 Ansible，Chef 等。虽然这些工具非常适合整备不可变的基础设施组件，但是对于管理有状态的系统，它们并不太管用，而且尝试使用这些工具管理和演进集群通常会导致大量的工作。所以，我们建议使用 Ambari 等工具来配置和管理有状态的Hadoop或Spark群集。

我们越来越多地看到组织准备使用“任意云”，并不惜一切代价避免被单一供应商锁定。
(Generic cloud usage)

主流云供应商在定价和快速发布新特性方面的竞争日益激烈，这使得消费者难以抉择和保持粘性。我们越来越多地看到组织准备使用“任意云”，并不惜一切代价避免被单一供应商锁定。当然，这就导致了只使用**通用云服务功能(GENERIC CLOUD USAGE)**。我们发现，有的组织对云的使用仅限于所有供应商都有的特性，而忽略供应商提供

的独特能力；有的组织为了保持云无关性，斥巨资开发复杂且难以维护的抽象层。供应商锁定的问题是真实存在的。对此，我们建议采用多云策略，评估从一个云到另一个云的功能迁移成本及工作量，抵制使用特定云特性的好处。我们推荐将应用程序置于广泛采用的 Docker 容器中以提高工作负载的可移植性；使用开源的安全和身份协议以轻松迁移工作负载的身份；使用风险相称的供应商策略，仅在必要时维持云独立性；在合适之处使用 Polycloud 混合匹配不同供应商的服务。简而言之，请用合理的多云策略，避免只使用通用云服务。

微服务架构的一个显著特征是系统组件和服务是围绕业务能力进行组织的。无论系统规模大小，微服务都需要将系统功能和信息进行有意义的分组和封装，以便拆分后的微服务能彼此独立地交付业务价值。而以前的服务架构方式会根据技术特性组织服务。我们观察过很多采用**分层式微服务架构 (LAYERED MICROSERVICES ARCHITECTURE)**的组织，他们在某些方面存在着明显的矛盾。这些组织都陷入了以技术角色为主来划分服务的误区，比如，用户体验 API、进程 API 或系统 API等。我们会很自然地将技术团队按层划分，这会导致想要交付任何有价值的业务变更，都需要缓慢而昂贵的多团队合作。请务必小心这种分层方式带来的影响，我们更推荐根据业务能力来划分服务和团队。

主数据管理 (MASTER DATA MANAGEMENT (MDM)) 是一个典型的企业“银弹”解决方案：它声称可以将表面上相关的问题一次性解决，但它创建集中的单点来进行统一的变更、协调、测试和部署，极大地降低了组织响应业务变化的能力。在将MDM方案集成到不同消费和生产系统的过程中，组织往往尝试捕捉到所有的“主”数据并将它们映射到MDM中，这让整个实施过程变得漫长而复杂。

微服务已成为现代云计算系统中的领先的架构模式，但我们依旧认为团队在使用该架构时应谨慎。**MICROSERVICE ENVY**特指那些盲目追赶微服务潮流的现象，很多团队在实践微服务时并没有简化其系统架构，大多数的实践方案只是将一些简单的服务聚合在一起而已。目前，Kubernetes等平台简化了复杂的微服务系统的部署问题，其他服务提供商们也正在推进他们的微服务治理方案，这些强大的工具都可能裹挟团队走上微服务之路。但请千万谨记，微服务是通过开发复杂度来换取运维复杂度，并需要自动化测试、持续交付和DevOps文化提供坚实的支撑。

我们经常看到**在面向用户的工作流中使用请求 — 响应事件 (REQUEST-RESPONSE EVENTS IN USER-FACING WORKFLOWS)**的系统设计。这样一来，要么UI被阻塞，要么用户就必须等页面收到响应消息后重新加载。做出这类设计的主要依据往往是为了性能或是为了用统一的方式来处理后端之间的同步和异步通信。我们认为这样做会在开发、测试和运维上所增加不必要的复杂度，远远超过了采用这种统一方式带来的好处，我们强烈建议直接使用同步HTTP请求来处理后端服务之间的同步通信，而不必改成事件驱动的设计。如果做得好，使用HTTP通信很少会成为分布式系统的瓶颈。

这种仅关注自动化业务流程而不解决底层软件系统或功能的方法的问题在于，引入额外的耦合会使底层系统更改起来更加困难。
(RPA)

机器人流程自动化（**RPA**）是许多数字化转型计划的关键部分，因为它有望在不必对底层架构和系统进行现代化改造的情况下节省成本。这种仅关注自动化业务流程而不解决底层软件系统或功能的方法的问题在于，引入额外的耦合会使底层系统更改起来更加困难。这也会让未来任何解决遗留IT环境的尝试都变得更加困难。很少有系统能够忽视变化，因此RPA的进展需要与适当的遗留系统现代化战略相结合。

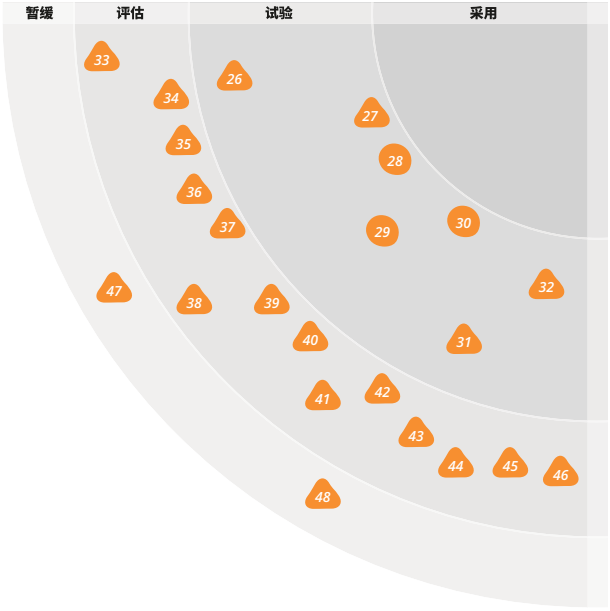
平台

采用

试验
26. Apache Atlas *NEW*
27. AWS
28. Azure
29. Contentful
30. Google Cloud Platform
31. Shared VPC *NEW*
32. TICK Stack

评估
33. Azure DevOps *NEW*
34. CockroachDB *NEW*
35. Debezium *NEW*
36. Glitch *NEW*
37. Google Cloud Dataflow *NEW*
38. gVisor *NEW*
39. IPFS *NEW*
40. Istio *NEW*
41. Knative *NEW*
42. Pulumi *NEW*
43. Quorum *NEW*
44. Resin.io *NEW*
45. Rook *NEW*
46. SPIFFE *NEW*

暂缓
47. Data-hungry packages *NEW*
48. Low-code platforms *NEW*



APACHE ATLAS 是一款用于满足企业数据治理需求的元数据管理框架。Atlas支持元数据类型建模、数据资产分类、数据来源追踪和数据发现。

(Apache Atlas)

随着企业数据需求的不断增长和多样化，对元数据管理的需求也在不断地增长。**APACHE ATLAS** 是一款用于满足企业数据治理需求的元数据管理框架。Atlas支持元数据类型建模、数据资产分类、数据来源追踪和数据发现。但是，在搭建元数据管理平台的时候，我们也必须小心避免重蹈主数据管理的覆辙。

我们在7年前首次将**AWS**移到“采纳”环，其服务的广度、深度和可靠性从那时起就已获得长足进步。然而，我们现在又将AWS移回“试验”环。这不是因为其产品存在缺陷，而是因为其竞争对手。它的 GCP 和 Azure 这两个竞争对手已经相当成熟，这使得选择云提供商变得越来越复杂。所以我们会把“采纳”环留给该领域未来的大赢家。多年来，AWS一直是该领域的默认选择。但是考虑到组织自身的地理位置和监管范围，以及同云提供商之间战略的一致性（或者缺乏一致性），当然还包括组织最重要的需求和云供应商差异化产品之间的契合度。所以，我们认为组织是时候应该在云供应商之间做出权衡了。

Microsoft已经在稳健地改进**AZURE**，如今大型云提供商Amazon、Google和Microsoft在核心云体验上并没有太大的差别。这些云提供商似乎都在追求其他方面的差异性，比如功能、服务和成本结构。Microsoft 对欧洲公司在法务上的需求表现出了真正的兴趣。对此，他们有一个细致且合理的策略，比如提供了像Azure Germany和Azure Stack这样各具特色的产品。这个策略为欧洲公司在预测GDPR以及美国可能对立法所做的更新中提供了几分把握。

Headless CMS（Content Management Systems, 内容管理系统）正在成为数字化平台的常见组件。**CONTENTFUL** 是一个现代化的 headless CMS。我们的团队已经成功把它集成到开发工作流中。我们特别喜欢其“API 优先”的特点，及其CMS as Code的实现。它支持强大的内容建模原语代码和内容模型演化脚本，并允许将其视为其他数据存储的

schema，并将演进式数据库设计实践应用到 CMS 开发中。我们所喜欢的其他特性包括：默认包含两个 CDN以提供多媒体资源和 JSON文档，本地化的良好支持和与Auth0集成的能力（尽管需要做出一些努力）。

随着**GOOGLE CLOUD PLATFORM** (GCP)在可用地理区域和服务成熟度方面的扩展，全球的客户在规划云技术策略时可以认真考虑这个平台了。与其主要竞争对手Amazon Web Services相比，在某些领域，GCP 所具备的功能已经能与之相媲美。而在其他领域又不失特色——尤其是在可访问的机器学习平台、数据工程工具和可行的“Kubernetes即服务解决方案”（GKE）这些方面。在实践中，我们的团队对GCP工具和API良好的开发者体验也赞赏有嘉。

按照传统的方式，我们需要反复为每个团队配置VPC、子网、安全组和网络访问控制列表，与此相比，我们更推崇Google的共享VPC（SHARED VPC）这一概念。共享VPC使组织、项目、VPC和子网成为网络配置中的一级实体。
(Shared VPC)

我们在大大小小的组织中积累了丰富的公共云经验，一些公有云模式也随之出现。其中一种模式是将组织级别管理的虚拟私有云，拆分为众多较小的由团队自治的子网。这个想法与设置多云账号类似，它有助于将基础设施按照团队边界进行分隔。按照传统的方式，我们需要反复为每个团队配置VPC、子网、安全组和网络访问控制列表，与此相比，我们更推崇Google的**共享VPC（SHARED VPC）**这一概念。共享VPC使组织、项目、VPC和子网成为网络配置中的一级实体。VPC可由组织的系统管理员管理，然后管理员可以将子网管理的权限委派给项目，项目与VPC中的子网一一对应。这样的方式简化配置，同时使安全性和访问控制更加透明。

TICK Stack 是一些开源组件的组合。这些组件组合成了一个平台，以便人们存储、可视化和监控诸如指标和事件这样的时间序列数据。这些组件包含了 Telegraf（用于收集和报告指标的服务器代理）、InfluxDB（高性能的时间序列数据库）、Chronograf（提供平台用户界面），以及 Kapacitor（能对来自 InfluxDB 的数据进行处理、流传输和批处理操作的数据处理引擎）。与基于拉模式的 Prometheus 不同，该平台是基于推模式来收集数据的。InfluxDB是该平台的核心组件，它是目前最优秀的时间序列数据库。该技术栈目前由InfluxData提供支持。虽然需要升级到该平台的企业版才可以使用数据库集群等功能，但将该平台用做监控仍然是相当不错的选择。我们当前在一些生产环境中使用了这一技术栈，并获得了很好的体验。

已取代 Visual Studio Team Services 的 **AZURE DEVOPS** 服务，包括一组托管服务，如Git仓库、CI和CD流水线以及制品库等。当使用Azure DevOps服务快速启动项目时（即在 Azure 平台上管理、构建和发布应用程序），我们获得了良好的体验。但同时也遇到了一些挑战，如该平台对CI和CD的“流水线即代码”缺乏全面的支持，构建代理在启动时速度很慢，构建和发布被分离到不同的流水线上来进行等，另外在使用该平台时还遭遇了几次停机。我们希望Azure DevOps服务能够随着时间的推移得以改进，以便为在Azure上托管应用程序的开发人员提供良好的体验，并与其他Azure服务无缝集成。

CockroachDB是一个开源分布式数据库，其设计思路源自白皮书Spanner：谷歌的分布式数据库。它提供分布式事务和地理分区的功能，并支持SQL。
(CockroachDB)

COCKROACHDB是一个开源分布式数据库，其设计思路源自白皮书Spanner：谷歌的分布式数据库。在CockroachDB中，数据自动按照区间进行切分，通常以64MB为单位，被分布到集群中的不同节点上。每一个区间都有一个共识组。由于使用了Raft共识算法，所以这些数据总是能够保持同步。凭借这种独特的设计，CockroachDB提供分布式事务和地理分区的功能，并支持SQL。不像依赖TrueTime（用原子时钟进行线性化）的 Spanner，CockroachDB使用 NTP 进行时钟同步，并且提供序列化功能（作为默认隔离级别）。如果所处理的是适合单个节点的结构化数据，那么可以选择传统的关系型数据库。但如果数据需要跨节点进行容量伸缩、保持一致并且能够在系统故障时保存下来，那么我们建议可以仔细研究一下CockroachDB。

我们一直在寻找这个领域的工具或平台（包括在之前的技术雷达中讨论过的 Bottled water），而 Debezium 是一个极佳的选择。它使用了基于日志的CDC方法，意味着能以对数据库日志文件的变更进行响应的方式进行工作。

(Debezium)

DEBEZIUM是一个change data capture (CDC)平台，可以将数据库的变更以流的形式传入 Kafka 主题中。CDC是一种流行的技术，具有多个使用场景，包括将数据复制到其他数据库中，为分析系统提供数据，从单块系统中提取微服务，以及令缓存数据无效等。我们一直在寻找这个领域的工具或平台（包括在之前的技术雷达中讨论过的 Bottled Water），而 Debezium 是一个极佳的选择。它使用了基于日志的CDC方法，意味着能以对数据库日志文件的变更进行响应的方式进行工作。Debezium使用了Kafka连接，这使得它具有高度的容量伸缩性，以及对故障的系统韧性。它拥有包括Postgres、Mysql和MongoDB在内的多个数据库的CDC连接器。目前，我们正在一些项目上使用该平台，并取得了很好的效果。

我们对 **GLITCH** 很感兴趣。这是一个在线协作开发环境，允许用户轻松复制和调整（或“重新混合”）现有的Web应用程序，或者创建自己的Web应用程序。该平台源于“修补匠”精神，对于编程初学者是一个理想的选择，同时也能够支持更加复杂的应用的开发。它主要关注JavaScript和 Node.js，对其他语言也提供有限的支持。通过集成实时编辑、托管、共享和自动化版本控制等功能，Glitch提供了一种令人耳目一新的独特的协作编程方式。

谷歌云数据流（GOOGLE CLOUD DATAFLOW）在传统的ETL场景中非常有用。它可以从数据源读取数据、转换数据并将转换后的数据存储到接收器中。整个过程的配置和容量伸缩都是由该平台进行管理。该平台支持Java、Python和Scala，并提供连接各种数据源的包装器。然而，该平台的当前版本并不允许用户添加其他库，所以可能会导致它不适合进行某些类型的数据操作。同时用户也无法动态地更改数据流的DAG（Directed Acyclic Graph，有向无环图）。因此，如果用户的ETL有基于参数的条件执行流程，就可能需要做一些变通才能使用该平台。

GVISOR是一个容器内的user-space内核。它不允许应用程序访问主机内核的所有功能，只能访问主机内核的表层。不同于现有通过虚拟化硬件实现的沙盒技术 KVM 和 Xen 或者基于规则执行的方式实现的沙盒技术seccomp, SELinux 和 AppArmor，gVisor通过拦截应用程序的系统调用这种特殊的方式来实现容器沙盒，并且不需要虚拟化硬件的转换就能够实现访问层内核。gVisor包含一个名为runsc的 Open Container Initiative（OCI）运行时，它集成了Docker和Kubenets（实验性支持）。gVisor是一个相对较新的项目，所以我们建议您根据自身容器安全状况对其进行评估。

在多数情况下，区块链不适合存储 blob 文件（例如：图像，音频），当人们开发 DApp 时，一种选择是将blob文件存放在一些链下的集中式数据存储中，这种做法通常会导致信任缺失，另一种选择是将它们存储在星际文件系统 **IPFS** 上，这是一种内容可寻址、版本化、点对点的文件系统。它旨在高效地分发大规模数据，并能阻止任何中心化机构删除数据，文件存储在不需要相互信任的对等节点上。IPFS 保存文件的每一个版本，这样你将永远不会丢失重要文件，我们将IPFS视为区块链技术的好的补充。除了区块链应用程序外，IPFS还有一个愿景是对现有的网络基础设施进行去中心化重塑。

构建和运维微服务生态系统的一个老问题，是如何实现一些横切关注点，例如服务发现、服务到服务的安全性、原始服务到其它服务的安全性、可观测性（包括遥测和分布式跟踪）、滚动发布和系统韧性。在过去几年中，service mesh 技术已经成为我们对这个问题的默认答案。Service mesh 以配置为代码的方式，在基础设施层实现了这些横切关注点。策略配置可以一致地应用于整个微服务生态系统。这些策略可以在 serivice mesh 流量内外（通过将 service mesh 代理作为网关的方式）以及每个服务的流量（通过将相同的 service mesh 代理作为 sidecar 容器的方式）上强制执行。在密切关注不同开源 service mesh 项目 Linkerd 的进展的同时，我们已经成功地在生产环境中使用了 **ISTIO** 。Istio易于配置的操作模型令人惊叹。

作为应用开发者，我们喜欢专心解决核心业务问题，而让底层平台来处理那些枯燥且困难的任务（如部署、容量伸缩及应用程序管理）。虽然无服务器架构往这个方向迈出了一步，然而大多数流行的产品都会与某个专有实现绑在一起。而这意味着供应商绑定。**KNATIVE**试图以开源无服务器平台的方式来解决此问题。它良好地集成了流行的Kubernetes生态系统。利用 Knative，可以对随时请求的计算进行建模（其间可以从一些框架中进行选择，如 Ruby on Rails、Django和Spring 等），可以订阅、交付和管理事件，可以集成用户所熟悉的 CI和CD 工具，可以从源代码构建出容器。该平台提供一组中间件组件，来构建以源代码为中心且基于容器（能够实现资源伸缩性）的应用。这使得它成为一个颇有吸引力的平台，当有无服务器需求时，值得对其进行评估。

它是云基础设施自动化领域很有前途的新星。其优势在于，允许使用 TypeScript/JavaScript、Python 和 Go 语言（无需YAML配置）来编写配置。

(Pulumi)

我们对 **PULUMI** 非常感兴趣。它是云基础设施自动化领域很有前途的新星。其优势在于，允许使用 TypeScript/JavaScript、Python 和 Go 语言（无需YAML配置）来编写配置。Pulumi 专注于云原生架构（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。

在区块链技术领域，Ethereum(以太坊)是一个领先的开发者生态系统。我们看到了一些新兴的解决方案，它们旨在将Ethereum这项技术传播到一些企业环境中。这些企业通常需要网络权限和交易隐私管理，另外还需要更高的吞吐量和更低的延迟。**QUORUM** 就是其中的一个解决方案。Quorum最初由J.P. Morgan开发，其定位是“企业版的Ethereum”。与创建了新的以太坊虚拟机（EVM）的 Hyperledger Burrow 节点不同，Quorum的代码源自以太坊官方客户端的一个分支，所以能与以太坊一起进化。在保留了以太坊账本的大多数功能的前提下，Quorum将共识协议从工作量证明机制更改为更高效的协议，并增加了私有交易支持。使用Quorum，开发人员可以使用他们的以太坊知识来构建企业级的区块链应用，这些知识包括 Solidity 语言和 Truffle 合约。然而根据我们的经验，Quorum还没有为应用于企业做好充足的准备；比如：它缺乏针对私有合约的访问控制机制，无法用于负载均衡，并且只支持部分数据库。所有的这些限制都为用户的部署与设计带来显著的负担。我们建议在使用Quorum的时候保持警惕，同时密切关注它的后续发展。

RESIN.IO 是一个物联网（IoT）平台。虽然只做把容器部署到设备中这一件事，但它做得很好。开发人员使用一个软件即服务（SaaS）的门户来管理设备，并为这些设备分发由 Dockerfile 定义的应用。该平台可以为多种硬件类型构建容器，并通过无线的方式部署容器镜像。Resin.io 使用 balena 来管理容器。而balena是一个基于 Mobby 框架的容器引擎，由 Docker 出品。该平台仍在开发中，有些功能尚需完善，也缺少一些特性（比如与私有容器注册服务协同工作）。但是目前的特性集（包括从 Web 门户使用 ssh 访问一个设备上的容器）表明它的未来充满希望。

ROOK是一款运行在Kubernetes集群中的开源云原生存储编排工具。与Ceph集成之后的Rook，能将文件、块和对象存储系统引入到Kubernetes集群中，并能与使用这些存储的其他应用和服务一起无缝地运行。通过使用一些Kubernetes operator，Rook可以在控制层上编排Ceph，这样就可以避免挤占应用程序和Ceph之间的数据通道。存储是云原生计算中的重要组件，虽然Rook现在仍然在CNCF进行孵化，但是我们相信，它将引领我们向着自给自足和跨公有云和本地化部署的可移植性更进一步。

利用谷歌具有开创性的高容量平台的关键组件来构建开源产品，似乎已成为一种趋势。如同HBASE利用了谷歌的BigTable，Kubernetes 利用了谷歌的Borg，**SPIFFE** 正在利用谷歌的LOAS，来将一种称为工作负载标识的关键云原生概念转化为现实。SPIFFE标准由开源软件 SPIFFE Runtime Environment (SPIRE) 提供支持，该软件可自动为软件工作负载提供加密且可证明的身份。虽然SPIRE还没有为在生产环境使用做好准备，但我们看到了其在以下场景中的巨大价值：以平台无关的方式，在现代分布式IT基础设施中的工作负载之间，进行强身份验证。SPIRE支持许多用例，包括身份转换、OAuth客户端身份验证、mTLS“无处不在的加密”和工作负载可观察性。本期雷达所介绍的 Istio 默认就使用SPIFFE。

数据饥饿软件包(DATA-HUNGRY PACKAGES)是一种解决方案，它需要将数据吸附到自身才能运行。在某些情况下，它们甚至可能需要成为这些数据的“主人”。一旦这种软件包拥有了数据，它就成为更新、修改或访问这些数据的唯一方法。这种软件包可能会解决诸如ERP这样特定的业务问题。但是，组织的有关库存或财务的“数据需求”，通常需要复杂的集成和对位于原始范围之外的系统的更改。

低代码平台(LOW-CODE PLATFORMS)使用图形化用户界面与图形化配置来创建应用程序。然而不幸的是，该平台的推广理念却是软件开发不再需要有经验的开发团队。这种理念忽视了这样的事实，即在创建高质量软件所需要的所有实践中，编写代码仅仅是其中的一小部分，而诸如控制源代码、测试和精心设计解决方案这些实践，也同样重要。尽管这种平台有时也是有用的，但我们仍建议要小心对待，特别是当他们肆无忌惮地宣称其能实现更低成本和更高产出的时候。

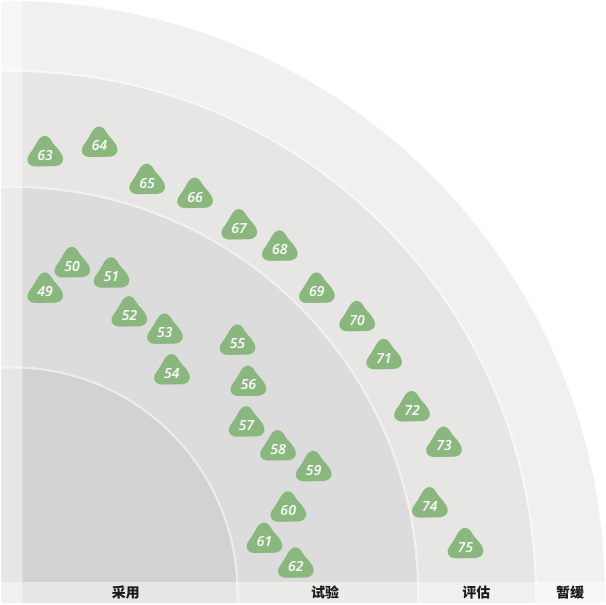
工具

采用

试验
49. acs-engine *NEW*
50. Archery *NEW*
51. ArchUnit
52. Cypress
53. git-secrets *NEW*
54. Headless Firefox
55. LocalStack *NEW*
56. Mermaid *NEW*
57. Prettier *NEW*
58. Rider *NEW*
59. Snyk *NEW*
60. UI dev environments *NEW*
61. Visual Studio Code
62. VS Live Share *NEW*

评估
63. Bitrise *NEW*
64. Codefresh *NEW*
65. Grafeas *NEW*
66. Heptio Ark *NEW*
67. Jaeger *NEW*
68. kube-bench *NEW*
69. Ocelot *NEW*
70. Optimal Workshop *NEW*
71. Stanford CoreNLP *NEW*
72. Terragrunt *NEW*
73. TestCafe *NEW*
74. Traefik *NEW*
75. Wallaby.js *NEW*

暂缓



Azure Container Service Engine (**ACS-ENGINE**) 是用于 Azure Resource Manager (ARM) 的模版生成器。ace-engine 使用一个 JSON 文件进行集群配置，并生成 ARM 所需的一系列文件。该工具可以灵活选用不同的容器编排工具，包括 Kubernetes、DC/OS、OpenShift、Swarm mode 与 Swarm，也可以灵活配置集群的特征及代理。我们已经在多个项目中应用了 acs-engine，并推荐用它来管理 Azure Container Service 中的集群。

我们看到安全工具与现代软件交付过程的集成有了显著进步。**ARCHERY** 是一个开源的安全工具，它的社区活跃，并正在将其与其他工具（包括 Zap）相结合。Archery 主要用于 Web 应用程序，可以轻松地将安全工具集成到构建与部署系统中。也可以通过 Archery 的工作面板，跟踪漏洞及应用程序和网络的安全扫描结果。

ARCHUNIT是一个基于 Java 的测试库，用于检查代码的结构特性，如包和类的依赖关系、注解验证，甚至还能检查代码分层是否一致。我们很喜欢 ArchUnit 的地方是，它可以在现有的测试环境中以单元测试的方式运行，尽管只支持基于 Java 的架构。在CI环境或部署流水线中集成 ArchUnit 测试套件，可以方便地在演进式架构中实现架构适应度函数。

运行端到端测试时经常会遇到一些棘手的问题，比如运行时间过长，测试过于零碎，还需要修复无头模式下运行的测试所导致的 CI 失败。我们的团队借助 **CYPRESS** 很好地解决了性能差、响应时间长、资源加载慢等常见问题。Cypress 是一款很有用的工具，可以帮助开发者构建端到端测试，还可以将所有测试步骤保存为 MP4 视频，便于检查错误。

Git-secrets 是防止将密码或其他敏感信息提交到 git 仓库的小工具，也可以在公开代码库之前使用其扫描全部历史提交。
(git-secrets)

安全性仍然至关重要，而粗心地将安全凭据或其他机密提交到源代码仓库是一个主要的攻击向量。**GIT-SECRETS** 是防止将密码或其他敏感信息提交到 git 仓库的小工具。也可以在公开代码库之前使用 git-secrets 扫描全部历史提交，以确保没有意外地提交凭据。git-secrets 内建支持常见的 AWS 密钥和凭据，也可以为其他的提供商进行快速配置。

Firefox 无头模式(HEADLESS FIREFOX)与用于前端测试的 Chrome 无头模式一样成熟。与 Chrome 无头模式类似，在 Firefox 无头模式下运行浏览器测试时无需渲染 UI 组件，因此大大加快了 UI 测试的速度。

使用云服务时面对的一个挑战是如何在本地进行开发和测试。LOCALSTACK 为 AWS 解决了这个问题。它提供了各种 AWS 服务的本地 测试替身 实现，包括 S3、Kinesis、Dynamodb 和 Lambda 等。
(LocalStack)

使用云服务时面对的一个挑战是如何在本地进行开发和测试。LOCALSTACK 为 AWS 解决了这个问题。它提供了各种AWS服务的本地测试替身 实现，包括S3、Kinesis、Dynamodb 和 Lambda 等。它基于现有的最佳工具如 Kinesalite、Dynalite、Moto 等构建，并增加了进程隔离与错误注入的功能。LocalStack 的使用很简单，并附带了一个简单的 JUnit 运行器以及 JUnit 5扩展。我们在一些项目中使用过 LocalStack，并对它印象深刻。

MERMAID 使用类似 markdown 的标记语言来生成图表。Mermaid 为简化文档编写而生，并且发展迅速。目前已经为许多工具提供了插件支持，比如 Confluence、Visual Studio Code 和 Jekyll。可以试用一下 GitHub 上的在线编辑器。此外，Mermaid 还提供了很好用的命令行，可以使用图表定义文件生成 SVG/PNG/PDF。我们已经在许多项目中使用了 Mermaid，尤其欣赏的是，它可以用 markdown 简洁地描述图形和流程图，同时也可以将图表定义文件存储在代码仓库中。

Prettier 是一个颇有主张的 JavaScript 代码自动格式化工具（也在逐渐支持其它语言）。它通过强制实施自己主张的代码风格，增强了代码的一致性和可读性，并减少了开发人员在格式化上的工作量，以及团队在代码风格大讨论上浪费的工作量。
(Prettier)

PRETTIER 是一个颇有主张的 JavaScript 代码自动格式化工具（也在逐渐支持其它语言）。它通过强制实施自己主张的代码风格，增强了代码的一致性和可读性，并减少了开发人员在格式化上的工作量，以及团队在代码风格大讨论上浪费的工作量。虽然你可能不同意 Prettier 所强制选择的风格，不过我们发现它给团队带来的好处通常会大于这些风格方面的小问题。Prettier 可以与版本管理系统的“提交前钩子”或 IDE 插件一起使用。与任何格式化工具一样，一次性地格式化整个代码库可能会给版本控制历史带来混乱，不过我们觉得这只是一个小问题。我们特别欣赏的是 Prettier 不再使用基于 linter 的方法。借用来自 gofmt 的一句话：不仅验证代码，更保持它始终有效！

技术雷达在2015年引入了 Visual Studio Code，而如今它已不再是唯一基于 .NET Core 的跨平台 IDE。最近，作为 JetBrains 开发的 IDEA 平台的一员，**RIDER** 已经被广泛使用。Rider 的重构功能是基于 ReSharper 实现的，因此那些习惯于 ReSharper 快速灵巧的开发人员对其情有独钟。不仅如此，Rider 还为 .NET 带来了完整的 IDEA 平台，大大提升了开发效率。不管你喜欢哪个平台，都有必要尝试一下 Rider，它现在比 Visual Studio Code 更有优势。活跃的生态和强有力的竞争使这些工具能够持续改进，对整个社区来说是件好事。

SNYK 可以查找、修复及监控 npm 、Ruby 、Python 、Scala 、Golang 、.NET 、PHP 、Java 与 Docker 依赖树中的漏洞。将 Snyk 加入构建流水线后，它会基于一个托管的漏洞数据库，持续地监控和测试你的库依赖树。在发现漏洞时，还可以给出可以解决该安全问题的最小的依赖版本。

随着越来越多的团队接受DesignOps，这个领域的实践和工具也日渐成熟。我们的许多团队都在使用一整套支持UI组件快速迭代的环境（也称为**UI DEV ENVIRONMENTS**），以专注于用户体验设计人员与开发人员之间的协作。目前可用的环境有：Storybook，react-styleguidist，Compositor 及 MDX。这些工具既可以在组件库或设计系统的开发过程中单独使用，也可以嵌入到Web应用项目中使用。如果只是为了向组件中添加新功能，你可以只启动Storybook dev服务器，而不需要启动应用、BFF 或其他服务。

VISUAL STUDIO CODE 是微软推出的免费 IDE 编辑器，可以跨平台使用。我们曾用它同时进行前端 React、TypeScript 和后端 GoLang 的开发，而无需在不同的编辑器之间切换，体验很好。Visual Studio Code 中的工具、语言支持和扩展插件数量都在迅猛增长，也越来越好用。我们要特别推荐在实时协作及远程结对编程时使用 VS Live Share 。固然微软或 Jetbrains 成熟的 IDE 对使用静态类型语言（如 Java 、.NET 或 C++）的复杂项目支持得更好，但我们也发现 Visual Studio Code 正逐渐成为基础设施开发组和前端开发组的首选工具。

VS LiveShare 能够减少远程结对编程时遇到的障碍。我们特别欣赏的是，开发人员可以在使用 Live Share 协作时沿用自己的编辑器配置。

(VS Live Share)

VS LIVE SHARE 是用于 Visual Studio Code 与 Visual Studio 的插件，提供实时合作编辑与调试代码、语音通话、共享终端和暴露本地端口等功能，能够减少远程结对编程时遇到的障碍。我们特别欣赏的是，开发人员可以在使用 Live Share 协作时沿用自己的编辑器配置，包括主题、快捷键和扩展。

构建、测试和部署移动应用，尤其是由一条流水线从代码仓库打通到应用商店的时候，会涉及许多复杂的步骤。Bitrise 配置简单，并预置了一组完整的步骤，可以满足绝大多数移动应用开发所需。

(Bitrise)

构建、测试和部署移动应用，尤其是由一条流水线从代码仓库打通到应用商店的时候，会涉及许多复杂的步骤。虽然这些步骤可以由脚本或普通 CI/CD 工具提供的流水线自动完成，但对于专注移动应用开发，而不需要与后端的构建流水线做集成的小组来说，使用专用工具可以降低复杂度和维护开销。**BITRISE** 配置简单，并预置了一组完整的步骤，可以满足绝大多数移动应用开发所需。

CODEFRESH 是类似于 CircleCI 与 Buildkite 的托管型持续集成服务器。它以容器为中心，并将 Dockerfile 文件和容器托管集群视为一等公民。我们十分欣赏的是，CircleCI 鼓励流水线式的交付方式，并且支持分支与合并。我们的团队对 CircleCI 的前期反馈良好，但还没有验证在大型项目及复杂流水线上的使用效果。

我们一直在寻找一些工具和技术，使大型组织内的交付团队能够独立于其他部门工作，同时满足安全与风险管控的要求。Grafeas就是一个这样的工具。

(Grafeas)

我们一直在寻找一些工具和技术，使大型组织内的交付团队能够独立于其他部门工作，同时满足安全与风险管控的要求。**GRAFEAS** 就是一个这样的工具。组织可以使用它发布软件工件（Docker镜像、库及软件包）的权威元数据，并在构建脚本或其他自动化的合规性控制过程中使用这些元数据。通过访问控制机制，可以将发布审核或漏洞信息的团队与构建、部署软件的团队间的职责划分清楚。请注意，虽然包括 Google 与 JFrog 在内的多个组织已经在工作流程中使用了 Grafeas，但它目前仍处于 alpha 阶段。

Heptio Ark 是用于 Kubernetes 集群和持久化卷的灾难恢复管理工具。Ark 使用一系列检查点备份与恢复集群，配置使用简单。
(Heptio Ark)

HEPTIO ARK 是用于 Kubernetes 集群和持久化卷的灾难恢复管理工具。Ark 使用一系列检查点备份与恢复集群，配置使用简单。使用 Ark 可以显著缩短基础架构发生故障时的恢复时间，还能轻松地将 Kubernetes 资源从一个集群迁移到另一个集群，或者复制生产环境用于测试和排错。Ark 支持主流的云存储提供商（包括 AWS，Azure 和 Google Cloud），并且从版本0.6.0开始，提供了插件系统用于兼容其他备份与卷存储平台。虽然 GKE 等 Kubernetes 托管环境已经提供了这类服务，但如果需要自行运维 Kubernetes，不论是在本地还是云端，都请仔细考虑使用 Heptio Ark 进行灾难恢复。

JAEGER 是一个开源的分布式追踪系统。与 Zipkin 类似，Jaeger 的设计灵感来源于谷歌的 Dapper，并遵循 OpenTracing。Jaeger 的诞生晚于 Zipkin 但普及迅速。这是因为 Jaeger 支持更多种语言的客户端库，并且在 Kubernetes 集群中安装它也很简单。我们已经成功将 Jaeger 与 Istio 配合使用，在 Kubernetes 集群中与 Envoy 集成进行应用程序追踪。我们也很喜欢 Jaeger 的UI。随着 Jaeger 加入 CNCF，我们预计 Jaeger 会在社区工作上投入更多的精力，同时也会与 CNCF 内的其他项目产生更深度的融合。

KUBE-BENCH 是一款基础设施配置扫描工具，基于 K8S 的 CIS 评分自动检查 Kubernetes 配置，涵盖用户身份验证，权限控制和数据安全等方面。我们的团队发现 kube-bench 在识别易受攻击的配置方面很有价值。

OCELOT是基于.NET Core实现的轻量级API网关项目，它可以通过轻松的配置来实现路由转发、请求聚合、服务发现、认证授权、限流熔断、负载均衡等特性，它还集成了Service Fabric、Consul、Eureka等功能。目前Ocelot的功能已经相当完整，它在.NET Core社区的活跃度也很高，.NET社区的开发者已经对 Ocelot 进行了很多扩展，以支持gRPC、Orleans与 WebSocket 等通信协议。尽管市面上不乏优秀的 API Gateway(如 Kong)，但 .NET 社区在构建微服务时还是更青睐 Ocelot。一方面是由于 Ocelot 能够更好的与 .NET生态(如 IdentityServer4)集成，另一方面腾讯已经将其用于生产环境来构建网关，这无疑是给Ocelot在可用性方面注入了一剂强心剂。

在调研用户体验时，往往需要收集并分析数据，以便在设计产品时做出更好的决策。**OPTIMAL WORKSHOP** 是一套数字化的调研工具。它提供了首次点击、卡片分类等功能，可以帮助验证原型以及改进网站导航和信息展示。Optimal Workshop 还可以协助组织远程调研，对于分布式的团队特别有用。

越来越多的项目需要处理非结构化的数据，而从文本中提取出有意义的业务信息是一项关键技术。Stanford Corenlp 是一组基于 Java 的自然语言处理（NLP）工具集。
(Stanford CoreNLP)

越来越多的项目需要处理非结构化的数据，而从文本中提取出有意义的业务信息是一项关键技术。**STANFORD CORENLP** 是一组基于 Java 的自然语言处理（NLP）工具集，支持英语、汉语和阿拉伯语等多种语言的命名实体识别、关系抽取、情感分析与文本分类，也提供了用于标记语料库和训练模型的工具。Stanford CoreNLP 协助我们使用 NLP 领域的最新研究成果来解决各种业务问题。

我们广泛的应用 Terraform 来实现代码化配置（as-code）云基础设施。**TERRAGRUNT** 是 Terraform 的一个轻量级的封装，用来落地《Terraform: Up and Running》书中主张的实践。我们发现 Terragrunt 很有帮助，因为它通过一些便利的特性来促进版本化模块和不同云环境的复用性，这些功能包括递归执行子目录下的代码。我们希望 Terragrunt 能再进化一些，能够原生地支持持续交付的实践，我们希望在持续交付的流水线上所有基础设施的代码都能被打包、版本化并能在不同环境下复用（我们团队已经通过替代方案实现了这些功能）。

我们团队对 **TESTCAFE** 的反响很好。这是一个基于 JavaScript 的自动化浏览器测试工具，可以使用 JavaScript 或 TypeScript 编写测试，并在任何支持 JavaScript 的浏览器中运行测试。TestCafe 提供了开箱即用的并行执行、HTTP请求模拟等有用的功能。TestCafe 使用异步执行模型而无需指定等待时间，有效提升了测试套件的稳定性。

TRAEFIK 是一款开源的反向代理及负载均衡器。如果只是需要具有简单路由功能的边缘代理，而非 NGINX 、HAProxy 这样的重量级产品，就可以考虑 Traefik 。它提供了微服务所必不可少的免重载更新配置、度量、监控与断路器等功能，同时也很好地集成了 Let’ s Encrypt 以支持 SSL。相比于 Traefik ，为了扩展、添加或移除微服务，NGINX 、HAProxy 等工具可能需要额外的工具进行模板化配置；有时还需要重启，给生产环境造成很大麻烦。

我们非常关注测试驱动开发过程中的快速反馈，并一直寻找新方法使它变得更快。**WALLABY.JS** 是一款支持主流编辑器的商用扩展，可以持续运行 JavaScript 单元测试，并在代码旁高亮显示测试结果。它还可以识别及运行每项代码改动所影响的最小测试集，并在代码录入的同时持续运行测试。

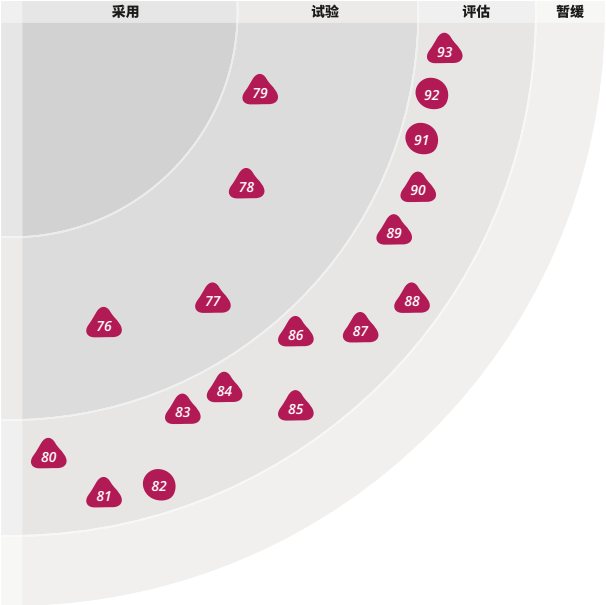
语言&框架

采用

试验
76. Jepsen
77. MMKV *NEW*
78. MockK *NEW*
79. TypeScript

评估
80. Apache Beam *NEW*
81. Camunda *NEW*
82. Flutter
83. Ktor *NEW*
84. Nameko *NEW*
85. Polly.js *NEW*
86. PredictionIO *NEW*
87. Puppeteer *NEW*
88. Q# *NEW*
89. SAFE stack *NEW*
90. Spek *NEW*
91. troposphere
92. WebAssembly
93. WebFlux *NEW*

暂缓



随着 微服务 架构越来越多地被采用，相比以前，我们构建了更多的分布式应用程序。尽管解耦架构带来了许多好处，但证明整个系统正确性所需的工作量和复杂程度正急剧增加。**JEPSEN** 提供了许多我们所需要的工具，来帮助我们验证协调任务调度程序的正确性，测试分布式数据库的最终一致性、线性一致性（Linearizability）和 可串行性（Serializability）。我们在一些项目中使用了 Jepsen，令人惊喜的是，我们可以测试驱动配置，注入和修复故障，验证系统恢复后的状态。

MMKV 是微信开发的开源框架，为移动应用提供高速的键值对存储。它利用 iOS 的内存映射功能来避免直接保存修改，因此性能极高。MMKV 也能够在应用程序异常崩溃时保存并快速恢复数据。

作为原生的开发库，它能帮助开发团队在测试 Kotlin 应用时编写干净、简洁的代码。
(MockK)

MOCKK 是用 Kotlin 编写的模拟库。它的核心理念是像 Coroutines 和 Lambda 表达式一样，为 Kotlin 提供一等公民级别的语言特性支持。不同于 Mockito 或 PowerMock 的蹩脚封装，作为原生的开发库，它能帮助开发团队在测试 Kotlin 应用时编写干净、简洁的代码。

TYPESCRIPT 是一门严谨的语言。一直以来，它不断改进的开发工具和IDE支持给我们留下了深刻的印象。随着基于浏览器的代码库数量不断增长，类型安全变得尤为重要，而使用此 TypeScript 类型定义库，可以让我们在保证类型安全的同时从丰富的 JavaScript 库中受

益。TypeScript 的类型安全特性让我们在使用 IDE 或其他工具进行代码开发时获得了更详尽的上下文，从而保障了代码修改和重构的安全性。作为 JavaScript 的超集，TypeScript 的文档和社区使学习曲线变得平缓。

Apache Beam 是个开源的统一编程模型，用于定义和运行批量/流式的并行数据处理流水线。Beam 提供了可移植的 API 层，可以直接定义流水线，而不依赖于具体的执行引擎（也称为 runner）如 Apache Spark、Apache Flink 或 Google Cloud Dataflow。这些引擎的功能不尽相同，因此提供可移植的 API 是项艰巨的任务。Beam 积极地尝试将各引擎的创新功能加入 Beam 模型，同时通过社区合作的方式，影响这些引擎的产品规划，以达到微妙的平衡。Beam 提供了丰富的内置I/O转换器，以满足大多数数据流水线的需求。如有特殊需求，也可以在 Beam 中实现自定义转换器。可移植 API 以及可扩展 I/O 转换器，是评估是否将 Apache Beam 用于处理数据流水线业务时最值得注意的地方。

我们常常对业务流程模型和标记法（BPMN）工具持怀疑态度，Camunda 简化了测试、版本管理和工作流重构方面的工作。Camunda 还可以与 Spring、Spring Boot 以及其他框架集成，这使它成为了我的不二之选。
(Camunda)

我们常常对业务流程模型和标记法（BPMN）工具持怀疑态度，因为它们普遍表现出低代码环境相关的缺点。然而 OSS BPMN 框架 **CAMUNDA** 不仅提供了一些这方面的创新，还支持在 Java 代码中以库的方式直接集成它的工作流和决策引擎，从而简化了测试、版本管理和工作流重构方面的工作。不仅如此，Camunda 还可以与 Spring、Spring Boot 以及其他框架集成，这使它成为了我的不二之选。

FLUTTER是一个跨平台的框架，可以使用Dart语言编写原生 Mobile 应用。借助 Dart，它能够编译成原生代码，并直接 和目标平台通讯，而不必借助桥接和上下文切换——这些可能导致框架中出现性能瓶颈，就像 React Native 或 Weex 那样。Flutter的热重载（hot-reload）特性让人惊叹，它能在编 码时为你提供超快的视觉反馈。目前，Flutter 仍在 Beta 阶段，不过我们会持续关注它，了解其生态系统的成熟度。

Kotlin 语言已不仅仅适用于移动应用开发。新工具和框架的出现证明了其在 web 应用程序开发上的价值。**KTOR** 就是其中之一。与其它支持 Kotlin 的 web 框架相比，Ktor 本身就是用 Kotlin 编写的，并运用了诸如 coroutines 等语言特性来支持异步非阻塞的实现。除了具有轻量级架构外，它还能灵活的与各种不同的日志、依赖注入和模板引擎工具结合使用，这让 Ktor 成为了我们团队在创建 RESTful 服务时一个有趣的选项。

我们在与团队的交流中了解到 Python 卷土重来并席卷了多个技术领域。事实上，它正成为最常用的编程语言。这一方面得益于数据科学家和机器学习领域推动了它的应用，另一方面我们也看到有团队正将其运用于微服务的构建。**NAMEKO** 就是这样一个超轻量级的微服务框架，也是 Flask的 替代方案。与 Flask 不同的是，Nameko 只包含了 WebSocket、HTTP、AMQP 支持等有限功能。它对可测试性的重视也得到我们的欣赏。如果你不需要 Flask 提供的模板等功能，那么 Nameko 值得一瞧。

POLLY.JS 是个用于测试 JavaScript 网站和应用程序的简单工具。它可以拦截和模拟 HTTP 交互，从而简单快速地测试 JavaScript 代码而无需启动其依赖的服务或组件，我们的团队对这一点情有独钟。

PREDICTIONIO 是开源的机器学习服务框架。无论是普通开发者还是数据科学家，都可以利用它创建用于预测的智能应用。和所有其他智能应用类似，PredictionIO 由三个部分组成：数据收集和存储、模型训练以及模型部署和服务暴露。开发者只需要基于它提供的相应接口实现数据处理逻辑、模型算法和预测逻辑，无需在诸如存储数据以及训练模型之类的事情上投入额外精力。从我们的经验来看，在不要求高并发处理的情况下，PredictionIO 能支持不同大小的数据集。大多数时候我们使用 PredictionIO 来为中小企业构建预测类智能服务，或者在构建复杂定制化预测引擎的过程中进行概念验证。

在之前的技术雷达中，我们提到了使用 Headless Chrome 进行前端测试。如今，随着其他浏览器对 Chrome DevTools 协议（CDP）的引入，出现了一系列针对这些浏览器的自动化和测试相关支持库。即使在无头模式下，CDP 也支持对浏览器的细粒度控制。基于 CDP 协议，一些高级自

动化测试框架被陆续创造出来，而 **PUPPETEER** 正是其中之一。它可以通过单页应用程序驱动 Headless Chrome 模式，从而实现基于时间追踪的性能诊断等功能。我们的团队在使用过程中发现它比基于 WebDriver 的其它工具更快，也更灵活。

量子计算目前已经可供测试，但何时真正到来尚未明确。在硬件到位之前，我们已经可以通过语言和模拟器来实验和学习它，Q#可以让你更加了解它的前景。
(Q#)

量子计算目前已经可供测试，但何时真正到来尚未明确。在硬件到位之前，我们已经可以通过语言和模拟器来实验和学习它。尽管IBM等公司已经取得了不错的进展，我们对微软在 **Q#** 语言及其模拟器（本地32量子比特，Azure云上40量子比特）方面的工作更加关注。如果你想开始了解这项编程的前景，请查看他们在 GitHub 上的范例。

SAFE 技术栈是 Suave、Azure、Fable 和 Elmish 的简称。SAFE 囊括了一系列技术，形成了前后端一致的 Web 开发技术栈。SAFE 在服务端和浏览器端都使用了 F# 语言，因此注重于异步函数式类型安全的编程机制。它不仅提供了一些提高开发效率的功能比如热加载；还允许我们替换技术栈里的某些模块，例如服务器端 Web 框架或云提供商。

新编程语言的广泛使用往往会催生支持成熟工程实践的新工具，例如自动化测试，Kotlin 也不例外。Cucumber、RSpec 和 Jasmine 这些测试工具广为人知，它们采用 Gherkin 语言和需求规范（Specification）来编写测试。在以上几个工具启发下诞生的测试框架 **SPEK** 能让开发团队把像行为驱动开发这样的成熟实践带到 Kotlin 世界来。

在那些使用了AWS CloudFormation（而非Terraform）的项目中，我们尝试了用 **TROPOSPHERE** 作为在AWS上定义基础设施即代码的方式。Troposphere 是一个 Python 库，可以使用 Python 代码生成 JSON 格式的 CloudFormation 描述。我们喜欢 troposphere 是因为它很容易发现 JSON 错误，同时它也有类型检测、单元测试以及 DRY 组合 AWS 资源等功能。

WEBASSEMBLY 将“浏览器作为代码执行环境”向前推进了一大步。它是一种二进制编译格式，几乎以原生速度 跑在浏览器中，已经获得所有主流浏览器的支持并向后兼容。它拓展了编写前端功能的语言范围，早期集中在 C、C++ 和 Rust，并且也可以作为 LLVM 的编译目标。在沙盒中执 行时，它可以和 JavaScript 交互并且共享相同的权限和安全模型。当和Firefox 最新的流式编译器一起使用时，也可 以提升页面初始化速度。尽管还处在早期阶段，但是这个 W3C 标准绝对值得研究。

经过一系列应用的试用，WebFlux 给我们的团队留下了深刻的印象，并汇报说这种响应式（函数式）实现增强了代码的可读性和系统的吞吐量。
(WebFlux)

Spring Framework 5 已发布一年有余，它采用了响应式流 —— 一套非阻塞背压（backpressure）式异步流式处理标准。在传统的 Spring MVC 模块之外，**WEBFLUX** 为在 Spring 生态下编写 Web 应用提供了一个响应式替代品。经过一系列应用的试用，WebFlux 给我们的团队留下了深刻的印象，并汇报说这种响应式（函数式）实现增强了代码的可读性和系统的吞吐量。但他们也确实注意到，采用 WebFlux 需要在思维方式上做出一些重大转变，建议在 WebFlux vs. Spring MVC 的技术选型中考虑这一点。

第一时间获知最新版技术雷达发布消息，以及独一无二的技术研讨会与深度内容。

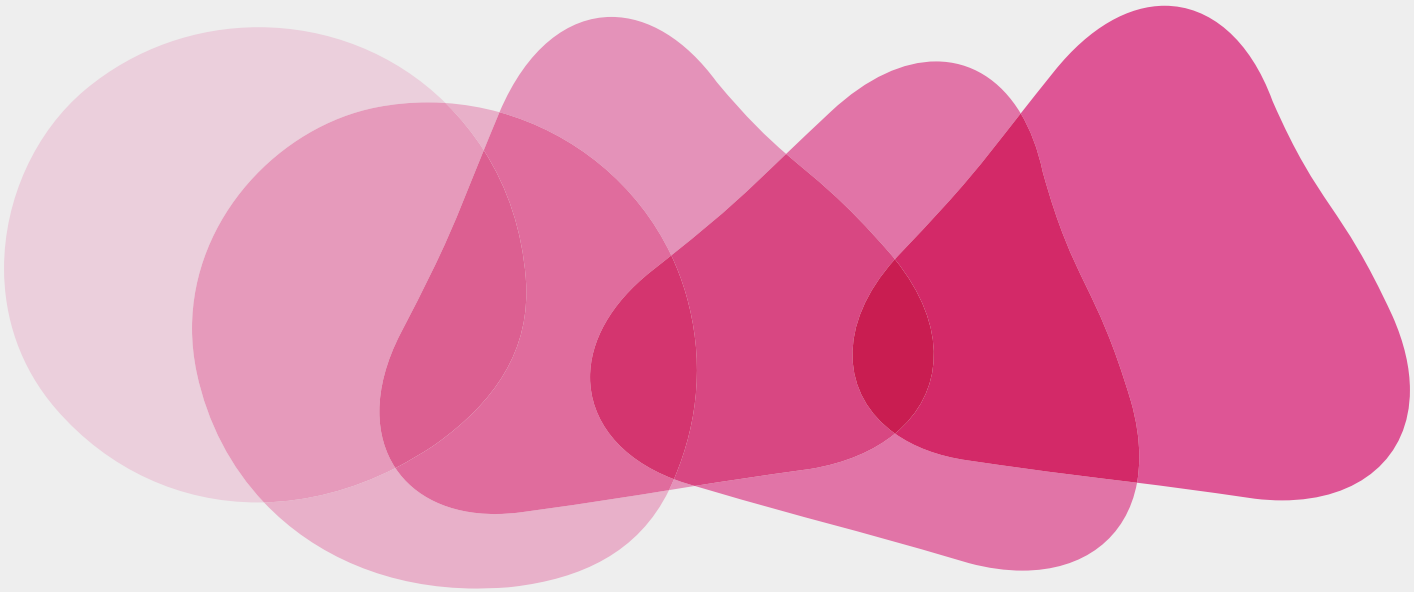
点击订阅

thght.works/Sub-CN

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5000人，分布于全球14个国家、拥有41间办公室的全球企业。这14个国家是：澳大利亚、巴西、加拿大、智利、中国、厄瓜多尔、德国、印度、意大利、新加坡、西班牙、泰国、英国、美国。



thoughtworks.com/cn/radar

#TWTechRadar