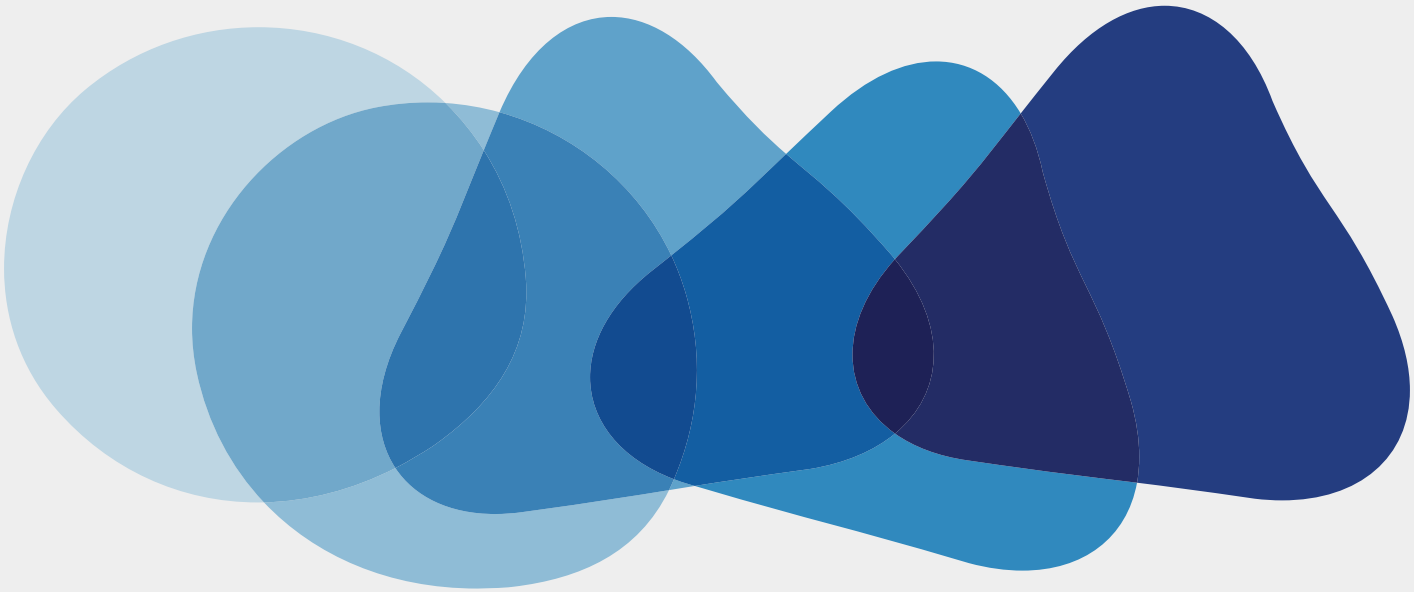
ThoughtWorks®

TECHNOLOGY RADAR *VOL.18*

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 | Rachel Laycock | Scott Shaw | 刘尚奇 | Zhamak Dehghani

技术雷达中国区技术咨询顾问组：

陈莹莹	顾宇	韩盼盼	靳亚堃	李好	刘先宁	马伟	彭洪伟	汪志成
王健	王静源	王瑞鹏	王晓雷	魏喆	伍斌	严嘉阳	杨乐涵	杨璐
姚琪琳	喻晗	袁慎建	张凯峰	张霄翀	张羽辰	赵正阳	郑达夫	

本期技术雷达是基于ThoughtWorks技术顾问委员会在2018年3月悉尼会议上的讨论所得出的。

最新动态

本期精彩集锦

浏览器增强，服务端式微

为了实现应用逻辑，浏览器在持续扩展成为部署目标的能力。当平台能照顾好横切关注点和非功能性需求的同时，我们注意到后端逻辑的复杂性有逐步降低的趋势。WebAssembly的引入为web应用创建逻辑提供了新的语言选择，同时把处理过程更加推向金属侧（以及GPU）。Web Bluetooth让浏览器能够处理那些原本是本地应用才能处理的功能，而且我们看到越来越多像 CSS Grid Layout和CSS Modules这样的开发标准正在替换掉自定义的库。对更好用户体验的追求，正在持续地把功能装进浏览器里，许多后端服务因此变得越来越薄，复杂性也因而降低。

不断蔓延的云环境复杂性

虽然AWS继续凭借令人眼花缭乱的新服务保持领先，但我们逐渐看到 Google Cloud Platform（GCP）和 Microsoft Azure 已经成为可行的替代方案。像 Kubernetes 这样的抽象层以及持续交付和基础设施即代码这样的实践，能支持更容易的演进式变化，从而促进云环境之间的过渡。但随着 Polycloud（这允许组织根据差异化的功能在多个供应商之间进行挑选）以及越来越多的监管和隐私问题的出现，云策略必然会变得更加复杂。例如，许多欧盟国家现在依法对数据所在地做出要求。这使得数据存储的管辖权和其主机的管理策略成为评估云计算环境的新维度。云计算环境的可选范围也在扩大，比如在“函数即服务”和“管理更长寿命集群”这两者之间，就可以选择提供了“容器即服务”（CaaS）的 AWS Fargate 这个有趣的选项。虽然各个组织对云技术的应用日臻成熟，但伴随使用这些新技术构建真实解决方案的，是逐渐蔓延又无法避免的复杂性。

信任团队，但要验证

对于几乎所有的软件开发来说，安全问题仍然是至关重要的。当前我们观察到传统的“全局权限管理”的安全策略正在转变为更为细致的本地化方法。现在许多系统会在更小的域（这里的域是抽象的概念，非windows域或者网域、域名）内管理信任，并在不同系统之间使用一些新的机制创建可传递的信任。其理念正在由“永远不信任域外所有东西”以及“从不验证域内任何东西”转变为“信任但是需要验证域内外任何东西”——也就是说可以假设和系统其它部分有本意良好的互动，但一定要在本地验证这份信任。这使得团队可以对自己的基础设施、设备和应用程序栈有高度的权限控制，从而实现高度可视化，并可以在必要时候提供高级访问护栏。类似 Scout2的工具以及 BeyondCorp 这样的技术反映了关于信任更成熟的视角。我们欢迎这种向本地化管理的转变，特别是当工具和自动化策略可以确保同等或更好的合规性时。

物联网的发展

物联网（IoT）生态系统持续稳步发展，关键成功因素包括安全和成熟的工程实践。我们看到了整个物联网生态系统的增长，从设备上的操作系统到连接标准，尤其是基于云服务的设备管理和数据处理。我们看到了一些成熟的工具和框架，支持良好的工程实践，比如持续交付、部署以及为实现最终广泛使用的大量其他必要实践。除了主要的云服务提供商——包括 Google IoT Core，AWS IoT 和 Microsoft Azure IoT Hub ——像阿里巴巴和阿里云这样的公司也在大力投资物联网 PaaS 解决方案。可以从我们的 EMQ 和Mongoose OS 条目一瞥当今物联网生态系统的主流功能，它们也例证了物联网的良好的发展状态。

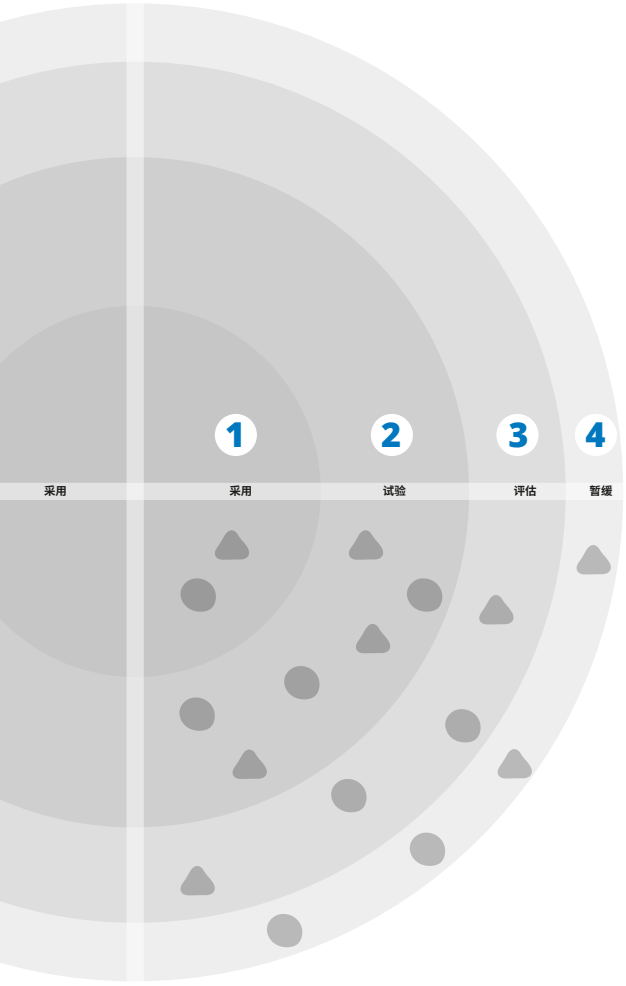
关于技术雷达

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。如果一个图标在一年内的两期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. Lightweight Architecture Decision Records

试验
2. Applying product management to internal platforms
3. Architectural fitness function
4. Autonomous bubble pattern
5. Chaos Engineering
6. Domain-scoped events *NEW*
7. Hosted identity management as a service *NEW*
8. Micro frontends
9. Pipelines for infrastructure as code
10. Polycloud

评估
11. BeyondCorp *NEW*
12. Embedded mobile mocks *NEW*
13. Ethereum for decentralized applications
14. Event streaming as the source of truth
15. GraphQL for server side resource aggregation *NEW*
16. Infrastructure configuration scanner *NEW*
17. Jupyter for automated testing *NEW*
18. Log level per request *NEW*
19. Security Chaos Engineering *NEW*
20. Service mesh
21. Sidecars for endpoint security
22. The three Rs of security

暂缓
23. Generic cloud usage *NEW*
24. Recreating ESB antipatterns with Kafka

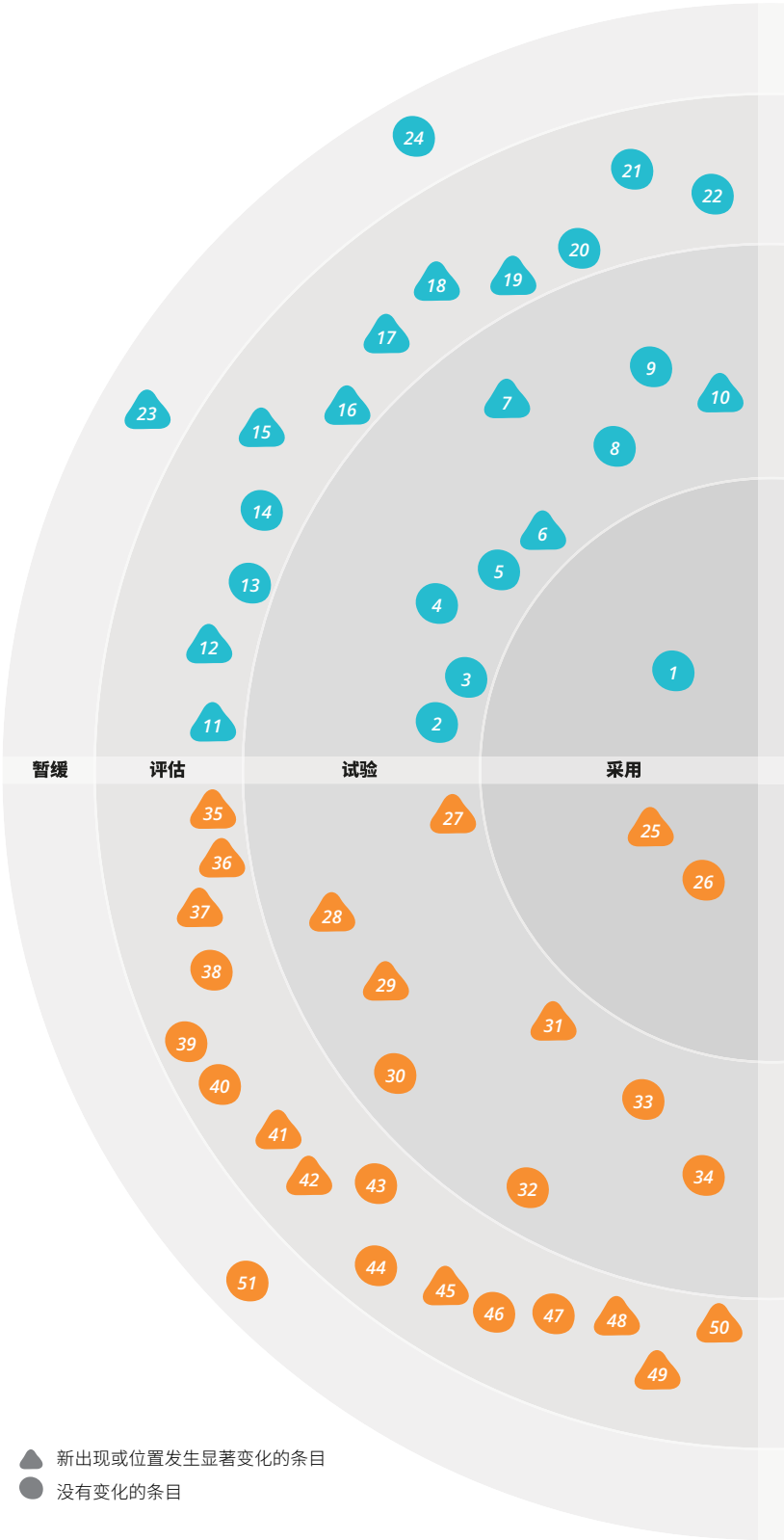
平台

采用
25. .NET Core
26. Kubernetes

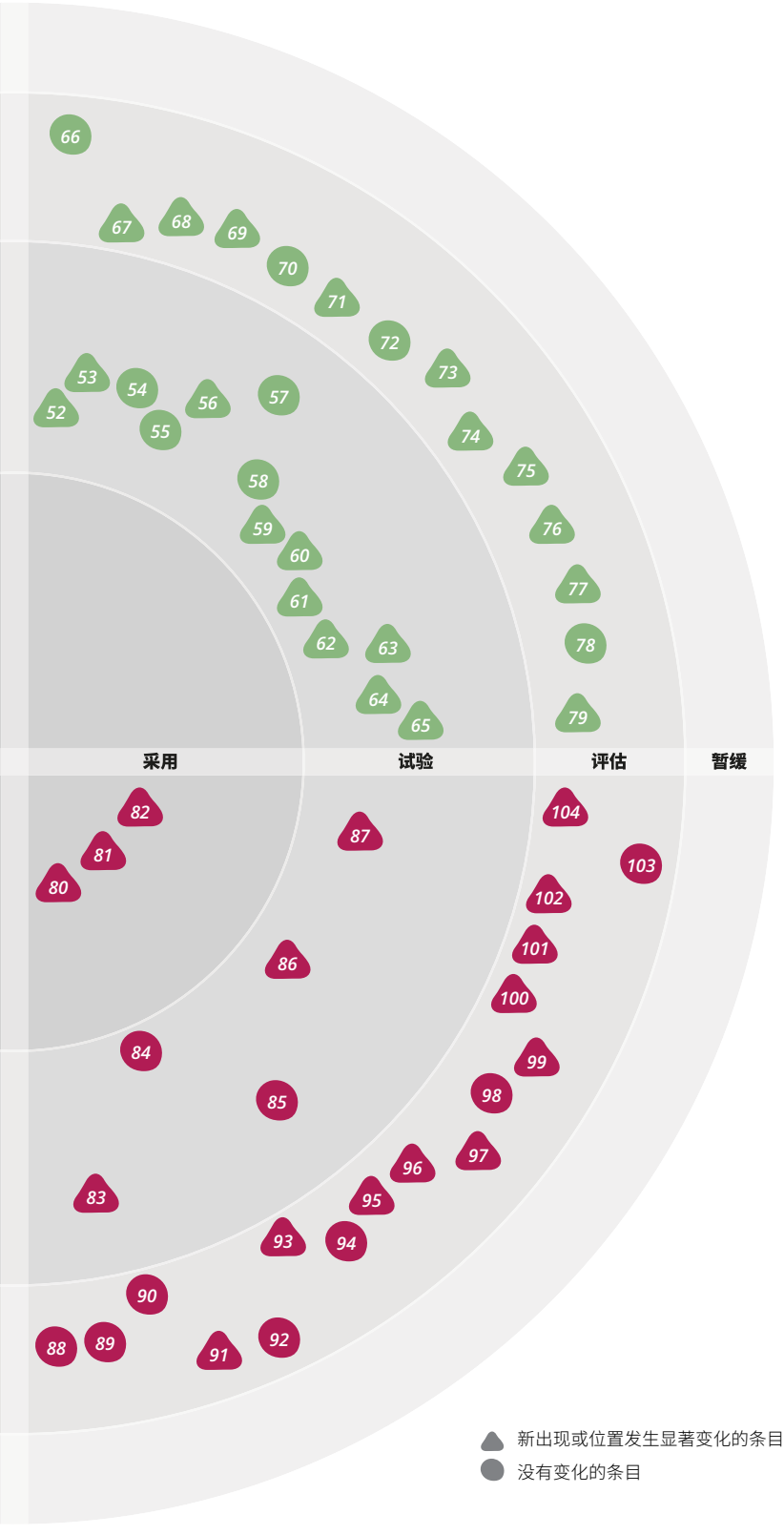
试验
27. Azure
28. Contentful *NEW*
29. EMQ *NEW*
30. Flood IO
31. GKE
32. Google Cloud Platform
33. Keycloak
34. WeChat

评估
35. AWS Fargate *NEW*
36. Azure Service Fabric
37. Azure Stack *NEW*
38. Cloud Spanner
39. Corda
40. Cosmos DB
41. Godot *NEW*
42. Interledger *NEW*
43. Language Server Protocol
44. LoRaWAN
45. Mongoose OS *NEW*
46. Netlify
47. TensorFlow Serving
48. TICK Stack *NEW*
49. Web Bluetooth *NEW*
50. Windows Containers

暂缓
51. Overambitious API gateways



THE RADAR



工具

采用

试验

52. Appium Test Distribution *NEW*
53. BackstopJS *NEW*
54. Buildkite
55. CircleCI
56. CVXPY *NEW*
57. gopass
58. Headless Chrome for front-end test
59. Helm *NEW*
60. Jupyter
61. Kong API Gateway
62. kops
63. Patroni *NEW*
64. WireMock *NEW*
65. Yarn

评估

66. Apex
67. ArchUnit *NEW*
68. cfn-nag *NEW*
69. Conduit *NEW*
70. Cypress
71. Dependabot *NEW*
72. Flow
73. Headless Firefox *NEW*
74. nsp *NEW*
75. Parcel *NEW*
76. Scout2 *NEW*
77. Sentry *NEW*
78. Sonobuoy
79. Swashbuckle for .NET Core *NEW*

暂缓

语言&框架

采用

80. AssertJ
81. Enzyme
82. Kotlin

试验

83. Apollo *NEW*
84. CSS Grid Layout
85. CSS Modules
86. Hyperledger Composer *NEW*
87. OpenZeppelin *NEW*

评估

88. Android Architecture Components
89. Atlas and BeeHive
90. Clara rules
91. Flutter *NEW*
92. Gobot
93. Hyperapp *NEW*
94. PyTorch
95. Rasa *NEW*
96. Reactor *NEW*
97. RIBs *NEW*
98. Solidity
99. SwiftNIO *NEW*
100. Tensorflow Eager Execution *NEW*
101. TensorFlow Lite *NEW*
102. Troposphere *NEW*
103. Truffle
104. WebAssembly *NEW*

暂缓

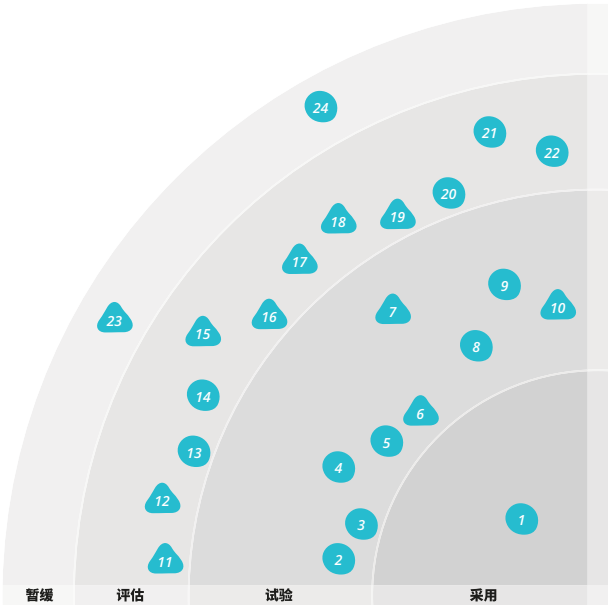
技术

采用
1. Lightweight Architecture Decision Records

试验
2. Applying product management to internal platforms
3. Architectural fitness function
4. Autonomous bubble pattern
5. Chaos Engineering
6. Domain-scoped events *NEW*
7. Hosted identity management as a service *NEW*
8. Micro frontends
9. Pipelines for infrastructure as code
10. Polycloud

评估
11. BeyondCorp *NEW*
12. Embedded mobile mocks *NEW*
13. Ethereum for decentralized applications
14. Event streaming as the source of truth
15. GraphQL for server side resource aggregation *NEW*
16. Infrastructure configuration scanner *NEW*
17. Jupyter for automated testing *NEW*
18. Log level per request *NEW*
19. Security Chaos Engineering *NEW*
20. Service mesh
21. Sidecars for endpoint security
22. The three Rs of security

暂缓
23. Generic cloud usage *NEW*
24. Recreating ESB antipatterns with Kafka



要记住，就像适用于其他软件领域一样，封装也同样适应于事件和事件驱动的体系结构。特别是，我们需要考虑一个事件的范围，以及我们是否期望它在同一个应用程序、同一个领域或整个组织中被消费。**DOMAIN-SCOPED EVENT**将在其发布的同一个领域内被消费，因此我们期望消费者能够访问特定的上下文、资料或引用，进而对事件进行处理。如果这个事件的消费在组织内更广泛地发生，且事件的内容需要有所不同，我们就要注意不要“泄漏”其他依赖领域的实现细节。

身份管理是平台的关键组件之一。外部用户在使用移动应用的时候，需要对其身份进行验证，开发人员需要被授权才能访问基础设施组件，而微服务也需要向彼此证明自己的身份。你应该考虑的是，身份管理是否真的有必要自己来搭建和维护。根据我们的经验，**HOSTED IDENTITY MANAGEMENT AS A SERVICE**（SaaS）这种解决方案更为可取。我们相信，像Auth0和Okta这样的顶级托管商可以在“正常运行时间”和“安全”两方面提供更好的SLA。也就是说，虽然有时候自行搭建和维护身份管理解决方案是一个现实的选择，特别是对于那些有操作规范和资源的企业来说，这个选择更为安全。但大型企业的身份解决方案通常包含更广泛的功能，例如集中授权、治理报告和职责分离管理等等。不过，这些担忧通常与员工身份更相关，特别是那些受遗留系统限制的企业，尤其如此。

组织们越来越习惯**POLYCLOUD**策略，不再把所有业务全“押在”一个服务供应商身上，他们会根据自己的策略，把不同种类的业务分配给不同的供应商。其中一些组织采用了最佳的解决方案，比方说：把标准服务部署在AWS上，把机器学习和面向数据的应用部署在Google，微软 Windows 应用则部署在Azure上。对于部分组织而言，这是一个关乎文化和商业的抉择。比如，零售行业往往不愿意把数据放在Amazon，他们会根据数据的不同分配给不同的供应商。“云不可知论”策略追求的是跨供应商的可移植性，这个代价很大，并且会导致为迎合所有要求刻意而为的决策。与之不同，Polycloud 策略更加注重选择每个供应商所提供的最好服务。

一些企业正在完全消除隐式信任的内部网络，并将所有的通信都视为通过公共互联网传输。

(BeyondCorp)

在上一期技术雷达中，我们讨论了无边界企业的出现。目前，一些企业正在完全消除隐式信任的内部网络，并将所有的通信都视为通过公共互联网传输。Google工程师在**BEYONDCORP**中描述了一套在大型企业内可行的实践方案，这套实践方案包括：受管设备、802.1x网络以及保护单个服务的标准访问代理。

在开发移动应用程序时，我们团队经常发现缺少外部服务器来测试应用程序。建立一个网络模拟器或许可以很好地解决这个问题。开发 HTTP 模拟器并将其编译到应用的二进制文件中进行测试——**EMBEDDED MOBILE MOCKS**，这使得我们可以在断开连接的状态下测试移动应用程序，且无需外部依赖。该项技术可能需要根据移动应用程序使用的网络库以及对底层库的使用情况创建一个专属的库。

在实践微服务的过程中，为了将后端资源进行聚合，我们实践了一个又一个的模式。之前，我们经常使用类似于Netflix Falcor这样的工具帮助我们实现BFF（Backend for Frontend），现在很多项目已经开始使用**GRAPHQL FOR SERVER-SIDE RESOURCE AGGREGATION**。GraphQL可以让客户端直接使用特定的查询语句去访问BFF以获取数据。使用这项技术时，后端服务可以继续暴露 RESTful API，而GraphQL可以轻易的将这些服务所提供的资源聚合在一起，并且对客户端十分友好。我们推荐GraphQL是因为其简化了BFF和其他聚合服务的实现。

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。

我们在一些有趣的报告中发现**JUPYTER FOR AUTOMATED TESTING**。Jupyter这种把代码、注释和输出整合在一个文档中的能力使我们想起了FIT、FitNesse和Concordion。这种灵活的方式对于高度依赖数据或统计分析的测试显得特别有用，比如性能测试。Python确实能够为自动化测试提供很多便利，但是随着测试的复杂性增强，一种能够管理一套记事本的方法将会变得更加有用。

在高度分布式的微服务架构中，其可观察性有一个两难问题 —— 要么记录一切，代价是巨量的存储空间；要么随机抽样记录，代价是有可能丢失某些重要事件。最近我们注意到一项技术，它在这两种方案之间提供了一个折衷方案。通过跟踪请求头中传入的某个参数来**LOG LEVEL PER REQUEST**。使用跟踪框架（可能基于OpenTracing标准），你可以在一次事务中的多个服务之间传递一个相关的ID。还可以在开始事务时注入其它数据（比如期望的日志级别），并且与跟踪信息一起传递它。这样可以确保这些额外数据在系统中总是和相应的单个用户事务一起流动。这在调试时也是个很有用的技巧，因为服务可能会暂停或以逐个事务的方式进行修改。

我们曾故意将误报引入到生产环境网络和其他基础设施——例如构建时的依赖关系中，检查它是否有能力在受控条件下识别安全故障。

(Security Chaos Engineering)

我们在上一期技术雷达里讨论了混沌工程，以及 Netflix 公司的 Simian Army 工具套件。我们已经采用它们来测试生产环境的恢复能力。**SECURITY CHAOS ENGINEERING**扩展了安全技术的范畴。我们曾故意将误报引入到生产环境网络和其他基础设施——例如构建时的依赖关系中，检查它是否有能力在受控条件下识别安全故障。虽然这个技术很有用，但应谨慎使用，以避免团队遇到安全问题。

我们越来越多地看到组织准备使用多个云，但是却不是为了同时享受每个供应商的优势，而是为了避免被单一供应商不惜一切代价地“锁定”。

(Generic cloud usage)

主流云提供商继续快速对其云服务添加新的特性，在Polycloud的旗帜下，我们建议并行使用多个云，基于每个提供商的产品优势来混合匹配多种服务。我们越来越多地看到组织准备使用多个云，但是却不是为了同时享受每个供应商的优势，而是为了避免被单一供应商不惜一切代价地“锁定”。当然，这就导致了**GENERIC CLOUD USAGE**，，也就是仅使用了所有供应商都有的特性。这让我们想起了10年前，当时的公司也是尽量规避使用关系数据库中的许多高级特性，尽量使用通用特性，以保持供应商中立。供应商锁定的问题是真实存在的。然而，不应该用“一刀切”的方法来处理它，我们建议从退出成本的角度来看待这个问题，并将这些问题与使用云特定特征的益处联系起来。

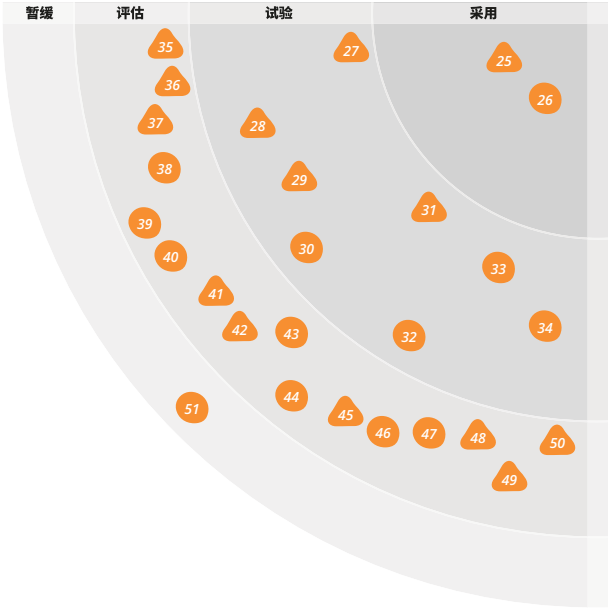
平台

采用
25. .NET Core
26. Kubernetes

试验
27. Azure
28. Contentful *NEW*
29. EMQ *NEW*
30. Flood IO
31. GKE
32. Google Cloud Platform
33. Keycloak
34. WeChat

评估
35. AWS Fargate *NEW*
36. Azure Service Fabric
37. Azure Stack *NEW*
38. Cloud Spanner
39. Corda
40. Cosmos DB
41. Godot *NEW*
42. Interledger *NEW*
43. Language Server Protocol
44. LoRaWAN
45. Mongoose OS *NEW*
46. Netlify
47. TensorFlow Serving
48. TICK Stack *NEW*
49. Web Bluetooth *NEW*
50. Windows Containers

暂缓
51. Overambitious API gateways



我们团队认为**.NET CORE**已经足够成熟，可以成为.NET服务器应用程序的默认平台。开源的 .NET Core 框架支持在 Windows、MacOS 和 Linux 操作系统上使用一流的跨平台工具来开发和部署 .NET 应用程序。微软提供了好用的 Docker 镜像，使得在容器化环境中部署.NET Core 应用程序变得非常简单。其在社区中积极的发展方向和我们项目的反馈表明—— .NET Core是 .NET 应用开发的未来。

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）。

EMQ是一个可伸缩的开源多平台 MQTT代理。为了追求高性能，它使用Erlang/OTP语言编写，能处理数百万的并行连接。它能支持多种协议，包括MQTT、MQTT传感器网络、CoAP以及WebSockets，使其适用于物联网和移动设备。我们已经开始在项目中使用 EMQ，很享受其安装以及使用的便捷性，以及它能将消息路由到不同目的地（包括Kafka和PostgreSQL）的能力，还有它在监控和配置上所采用API驱动的策略。

虽然软件开发生态系统正在将 Kubernetes 作为主要的容器编排平台，但运行 Kubernetes 集群的维护工作仍然很复杂。**GKE**（Google Kubernetes Engine）是一个托管式 Kubernetes 解决方案，用于部署容器化应用程序，以减轻运行和维护 Kubernetes 集群的运维开销。我们的团队有丰富的 GKE 使用经验，已经使用该平台在安装安全补丁、Node 监控和自动修复、管理多集群和多地区网络方面做了很多工作。根据我们的经验，谷歌以“API 优先”的方法暴露平台能力，以及使用如OAuth这样的行业标准进行服务授权，优化了开发人员的体验。然而，重要的是，要考虑 GKE 正处于快速的开发过程中，许多 API 都是以 beta 版发布。尽管其开发者通过抽象尽量不让消费者感觉到底层变化，但这仍然会对消费者有所影响。我们期待看到 Terraform on GKE以及其它类似的工具在基础设施即代码方面的成熟度会得到不断提升。

AWS FARGATE 最近进入了“docker as a service”（docker 即服务）的领域，目前尚只能在美国东一区（us-east-1）使用。当团队使用AWS Elastic Container Service（ECS）时，AWS Fargate 是一个能在其上工作的不错选项，因为开发人员无需管理、整备和配置任何底层的 EC2 实例或集群。Fargate允许把 ECS 或EKS（即针对 Kubernetes 的 ECS）任务作为一个Fargate的类型，并运行在 AWS Fargate 基础设施上。如果你喜欢 AWS Lambda 所提供的业务聚焦功能。但无法把应用部署为一个单一函数时，Fargate 是你最方便的选择。

AZURE SERVICE FABRIC是一个为微服务和容器而构建的分布式系统平台。我们可以因其可靠的服务，将它用作一个 PaaS 平台，也可以因其管理容器的能力，将它作为一个容器编排工具。Service Fabric的独特之处，是构建在其可靠服务之上的编程模型，比如 Reliable Actors。以物联网用例来说，Reliable Actors 提供了一些令人信服的优点——除了 Service Fabric 的可靠性和平台效益之外，还可以获得其状态管理和复制能力。本着继续关注开源软件（OSS）的原则，微软将 Service Fabric 转换为github上的开放开发过程。所有这些都使得Azure Service Fabric 值得一试，尤其是那些在.NET Framework 上进行投资的组织。

微软在全功能的公共云和简单的本地虚拟化之间提供了一个有意思的产品：一个运行 Microsoft Azure Global云的精简版本软件。

(Azure Stack)

相比自我托管的虚拟化解决方案，云计算的优势更加明显。但是由于延迟或监管的原因，数据有时不能轻易地离开组织的规定范围。对于欧洲公司来说，目前的政治气候也引发了更多关于“将数据放在美国管辖范围”的担忧。通过 **AZURE STACK**，微软在全功能的公共云和简单的本地虚拟化之间提供了一个有意思的产品：一个运行Microsoft Azure Global云的精简版本软件。该软件可以安装在诸如惠普和联想这样的预配置通用商品硬件上，从而让企业在本地获得核心的 Azure 体验。默认情况下，Microsoft 和硬件供应商所提供的技术支持是彼此分离的（他们承诺要相互合作），但系统集成商也能提供完整的 Azure Stack 解决方案。

随着 AR 和 VR 持续获得发展的动力，我们得以继续探索一些工具来创造这个身临其境的虚拟世界。在两大游戏引擎之一Unity上的良好体验，让它得以被纳入前几期的技术雷达之中。虽然我们依旧喜欢 Unity，但对**GODOT**这个该领域相对较新的工具也感到很兴奋。Godot 是一个开源软件，虽然不像大型商业引擎那样功能完整，但它的软件设计非常现代化，并且显得不那么杂乱。其所提供的 C# 和 Python 两个版本，降低了游戏行业以外开发者进入的门槛。在今年早些时候 Godot 所发布的 3.0 版，增加了对 VR 的支持，很快也将对 AR提供支持。

大多数人可能都是通过比特币了解到“钱联网”（Internet of money）。事实上，这个想法可以追溯到 Web 的早期阶段。HTTP 甚至为数字支付预留了状态码。

(Interledger)

大多数人可能都是通过比特币了解到“钱联网”（Internet of money）。事实上，这个想法可以追溯到 Web 的早期阶段。HTTP 甚至为数字支付预留了状态码。这个想法中具有挑战性的部分是在不同账本（ledger）中的不同实体之间进行价值转移。区块链技术通过构建分布式共享账本来促成这一想法。目前的挑战是如何实现不同区块链帐本之间的互操作性，以及与传统集中账本的互操作性。**INTERLEDGER**是一种连接不同帐本的协议。该协议使用连接器和加密机制（例如HTLC）在帐本之间路由安全支付的信息。通过其套件加入支付网络并不困难。Interledger 最初由 Ripple 发起，现在由 W3C 社区团体稳步推进。

随着相互连接的嵌入式设备数量的加速增长，以及硬件可以被更广泛地访问，**MONGOOSE OS**为嵌入式软件开发人员弥补了一个引人关注的隔阂，即适合原型开发的 Arduino 固件与裸机微控制器原生 SDK 之间存在的隔阂。作为一款微控制器操作系统，Mongoose OS 包含了一系列支持典型的物联网应用的程序库以及开发框架，并默认与通用的MQTT服务器及诸如Google Cloud IoT Core和AWS IoT这些流行的物联网云平台连接。实际上，谷歌也在其 Cloud IoT Core 中推荐使用Mongoose入门套件。在一个嵌入式项目中建立已连接的工作区时，我们使用了 Mongoose OS，并感受到了无缝连接的体验。在所有的功能中，其在单设备级别上的安全内置方案以及 OTA 固件更新方式，都受到我们的青睐。不过在我们编写本期技术雷达时，Mongoose OS仅支持有限数量的微控制器和板卡，更受欢迎的基于 ARM 的微控制器仍在开发中。

TICK STACK是一个由开源组件组成的平台。使用它就可以轻松地收集、存储、绘制基于时间序列的数据（如度量和事件）来触发告警。TICK Stack 的组件包括：收集和报告各种指标的服务器代理telegraf、高性能时间序列数据库 InfluxDB、平台的用户界面 Chronnograf，以及可以处理来自 InfluxDB 数据库的流式数据和批量数据的数据处理引擎Kapacitor。不像基于“拉”模型的Prometheus，TICK Stack是基于“推”模型来收集数据的。InfluxDB 组件是该系统的核心，同时也是目前最好的时间序列数据库。虽然这套组件栈基于 InfluxData，而且需要使用诸如数据库集群这样的 InfluxData 企业版的功能，但在监控方面它仍然是一个不错的选择。我们正在一些生产环境上使用该平台，并且获得了一些很好的体验。

WEB BLUETOOTH能够直接从浏览器控制任意低功耗蓝牙设备。这样以前只能通过原生手机应用来处理的场景，现在也可以适用了。该规范由 Web Bluetooth Community Group 发布，并且定义了一个 API，通过蓝牙 4.0 无线标准发现设备并在设备间通信。当前，Chrome 是唯一支持这个规范的主流浏览器。借助Physical Web和 Web Bluetooth，现在有了其他途径来让用户与设备进行交互，且无需让他们在手机上安装另一个应用。这是一个令人兴奋的领域，值得密切关注。

凭着能让Windows应用以容器的方式运行在基于Windows的环境中，**WINDOWS CONTAINERS**已经追赶上了容器世界的步伐。截至目前，微软提供两种 Windows OS 镜像来用作 Docker 容器——Windows Server 2016 Server Core和Windows Server 2016 Nano Server。它们都可以作为Windows 服务器容器运行在 Docker 中。在build agents场景中，我们的团队已经开始使用 Windows 容器。类似场景下的容器都可以很好地工作。微软已经意识到有些方面还存在优化空间，比如减少大型镜像文件的大小、加强对生态系统的支持和对文档进行丰富。

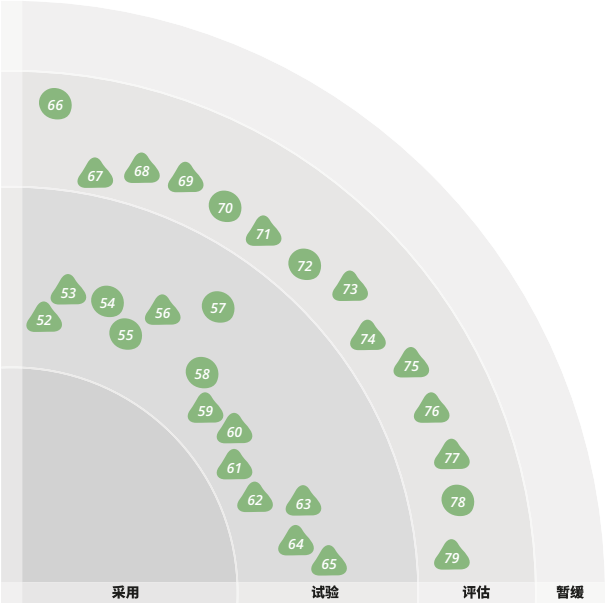
工具

采用

试验
52. Appium Test Distribution *NEW*
53. BackstopJS *NEW*
54. Buildkite
55. CircleCI
56. CVXPY *NEW*
57. gopass
58. Headless Chrome for front-end test
59. Helm *NEW*
60. Jupyter
61. Kong API Gateway
62. kops
63. Patroni *NEW*
64. WireMock *NEW*
65. Yarn

评估
66. Apex
67. ArchUnit *NEW*
68. cfn_nag *NEW*
69. Conduit *NEW*
70. Cypress
71. Dependabot *NEW*
72. Flow
73. Headless Firefox *NEW*
74. nsp *NEW*
75. Parcel *NEW*
76. Scout2 *NEW*
77. Sentry *NEW*
78. Sonobuoy
79. Swashbuckle for .NET Core *NEW*

暂缓



在往期的技术雷达中，我们推荐过Appium，它是最受欢迎的移动端自动化测试框架之一。随着测试套件的扩大，在多个设备上并行运行测试的能力成了缩短反馈循环的关键。**APPIUM TEST DISTRIBUTION**非常有效地解决了这个问题，它能够并行运行测试，且能在多个设备上运行同一套测试。它的出众之处在于在添加或移除测试设备时无需手动配置，并且支持在远程设备上运行测试。过去几年里我们在 ThoughtWorks 的一些项目中使用了这个工具，效果很不错。

我们很开心使用 **BACKSTOPJS** 来做 web 应用的可视化回归测试。作为可视化比较工具，它的可配置视窗和可调节容错能力可以很容易定位到细微差别。它有很优秀的脚本功能，并且可以选择在无界面Chrome、PhantomJS 和 SlimerJS 中运行。我们还发现，它在实时组件样式规范的基础上运行时尤其有帮助。

世界上有数不清的问题都可以用数学优化问题来表达，而其中可以用凸问题来描述的那部分常常能够得到有效解决。
(CVXPY)

世界上有数不清的问题都可以用数学优化问题来表达，而其中可以用凸问题来描述的那部分常常能够得到有效解决。**CVXPY**便是一种针对凸优化问题所开发的开源Python嵌入式建模语言。它由斯坦福大学的学者维护，已经为数个开源和商业解决方案提供了功能齐备的安装套件。它的文档中也包含了许多能够引起开发者使用兴趣的例子。尽管有些时候，我们仍然需要类似Gurobi和IBM CPLEX这类商业解决方案，但CVXPY在原型设计阶段可以说所向披靡。在多数情况下，有CVXPY就足够了。同时，基于最近的优化进展，其开发者一直在为我们提供更多的扩展包（例如DCCP）和相关软件（例如CVXOPT）。

HELM是Kubernetes的包管理器。共同定义某个应用的Kubernetes资源集合被打包成图表。这些图表可以描述单个资源，例如Redis pod，或者全栈的Web应用程序：HTTP服务器、数据库和缓存。Helm 默认带有一些精选的 Kubernetes 应用，维护在官方的图表仓库里。想要为内部用途搭建私有的图表仓库也很容易。Helm有两个组件：一个是称为Helm的命令行工具，另一个是称为Tiller的集群组件。保护Kubernetes集群是一个宽泛而微妙的话题，但我们强烈建议在基于角色的访问控制（RBAC）环境中搭建Tiller。我们在很多客户的项目中使用了Helm，它的依赖管理、模板和钩子机制极大地简化了Kubernetes中应用程序的生命周期管理。

> 在过去的几年里，我们注意到分析类notebooks应用越来越流行。这些深受数学启发的应用将文本、可视化以及代码都合并到了一个鲜活并且可以计算的文档中。
>
> (Jupyter)

在过去的几年里，我们注意到分析类notebooks应用越来越流行。这些深受数学启发的应用将文本、可视化以及代码都合并到了一个鲜活并且可以计算的文档中。如今机器学习越来越受关注，而且这个领域内的从业者越来越多选择Python作为编程语言，这都让基于Python的Notebook受到特别的关注，其中JUPYTER深受ThoughtWorks团队的青睐。除了将**JUPYTER**作为一个分析工具，开发者们还在尝试一些创新的用法，比如将Jupyter用于自动化测试。

Kong是一款开源的 API 网关，它提供一个企业版本集成了专有的API 分析和开发者门户。Kong可以以不同配置方式进行部署，比如作为边缘API网关、内部API代理、甚至是作为服务啮合配置中的边车模式（sidecar）。OpenResty通过 Nginx 模块，配以Lua作为扩展插件，Kong具备了强大和高性能的基础。Kong既可以用PostgreSQL作为单区部署，也可以用Cassandra作为多区域配置。我们的开发人员已经享受到了Kong的种种好处，包括它的高性能，它的API优先的方式(能使其配置自动化)，以及像容器般方便部署等等。它不像其他那些过度庞大的API网关产品，**KONG API GATEWAY**的功能更少，但实现了关键的API网关功能，比如流量控制、安全性、日志监控和权限认证。

KOPS是一款用于创建和管理高可用生产环境Kubernetes集群的命令行工具。它已经成为我们在AWS上管理Kubernetes 集群的首选工具，这不仅仅只是因为它快速增长的开源社区。它也可以 在Google Cloud上安装、升级和管理 Kubernetes 集群。不过我们在 Google 上使用 kops 的经验非常有限，因为我们更喜欢GKE这样的托管式 Kubernetes 服务。我们推荐在可复用的脚本中使用 kops 创建基础设施即代码。对于未来 kops 会如何持续演进，支持托管式 Kubernetes 集群，例如亚马逊自己的 Kubernetes 托管服务EKS，让我们拭目以待。

PATRONI是一个用于PostgreSQL high availability高可用性的模板。出于为PostgreSQL 提供自动故障恢复的需要，Patroni是一个基于 Python 的 PostgreSQL 控制器，它利用分布式配置存储（例如etcd，ZooKeeper，Consul）管理PostgreSQL 集群的状态。同时它还支持流复制和同步复制模型，并提供了一组丰富的REST API，用于PostgreSQL集群的动态配置。如果你想在分布式PostgreSQL设置中实现高可用性，并且不得不考虑许多边缘情况，令人欣慰的是，Patroni 提供了模板来实现常见的用例。

基于微服务的架构关键因素之一在于，服务是可以独立演进的。例如，当两个服务互相依赖时，对其中一个服务的测试通常会stub或mock另一个服务。我们可以手写这些stub或mock，但就像单元测试的mock一样，也可以借助框架，让开发者将注意力放到真正的测试场景上。我们知道**WIREMOCK**已经很久，但一直更倾向于使用mountebank。在过去的一年里，WireMock已迎头赶上，我们建议将其作为候选方案。

YARN是一款快速可靠且安全的JavaScript包管理器。通过锁定文件和确定性算法，Yarn能够确保在一个系统上可以正常运行的设置，在另外一个系统上也能以完全相同的方式工作。通过高效的请求队列，Yarn最大限度地提高了网络利用率，因此我们看到了更快的软件包下载速度。尽管npm（版本5）有最新的改进，但 Yarn 仍然是JavaScript包管理的首选工具。

ARCHUNIT是用来检查架构特征的Java测试库，比如包与类的依赖关系、注解验证、甚至层级一致性。它可以在你现有的测试方案中，以单元测试的方式运行，但目前只能用于Java架构。ArchUnit测试套件可以合并到CI（持续集成）环境或部署流水线，使我们很容易地以演进式架构的方式实现适应度函数。

云计算和持续交付对基础设施的安全有着巨大的影响。当遵循基础设施即代码时，整个基础设施——包括网络、防火墙和账户——都被定义在脚本和配置文件中，而通过凤凰服务器和凤凰环境，每次部署时基础设施都会重新创建一次，并且基础设施在一天之内多次重新创建也是经常发生的事情。在这样的场景下，对创建后的基础设施进行测试既不充分也不可行。一个叫**CFN-NAG**的工具可以帮助解决这个问题。它可以对用于AWS的CloudFormation模板进行扫描，通过模式匹配的方式寻找可能不安全的基础设施，而这一切都是在基础设施创建之前完成的。在构建流水线中运行像cfn-nag这样的工具的速度是很快的，而且它还可以在安全问题流入云环境之前就检测出来。

CONDUIT是为Kubernetes打造的轻量级Service Mesh开源产品。Conduit 实践了 out-of-process 架构，数据层代理使用 Rust 语言实现，控制层则使用Go语言实现。数据层代理以sidecar形式运行，以管理所有Kubernetes集群中的 TCP 流量。而控制层在Kubernetes独立的命名空间中运行，暴露REST APIs 来控制数据层代理的行为。通过代理所有的请求，Conduit 提供了大量有价值的度量数据，用于监控与观测 service mesh 中的交互，包括以下协议：HTTP、HTTP2/和gRPG。虽然 Conduit 在这个领域相对较新，我们仍然会因为它便于安装和使用而推荐它。

尽管让应用程序的依赖保持在最新状态是一件苦差事，但经常对升级进行增量式管理仍然很重要。我们希望这个过程尽可能轻松和自动化。以前，我们经常使用手工编写的脚本将这个流程中的某一部分自动化，现在，我们可以使用商业产品来完成这项工作了。**DEPENDABOT**提供了这种服务，它可以和你的Github代码仓库集成，并自动帮你检查项目依赖是否有最新的可用版本。如果有需要，Dependabot还可以创建Pull Request帮助你方便地对依赖进行升级。通过使用CI服务器，你可以自动对升级后的依赖做兼容性测试，并将兼容的依赖升级自动合并到Master分支。

除了Dependabot，还有其他一些工具可供选择，比如针对JavaScript项目的Renovate，以及针对JavaScript和Ruby项目的Depfu。不过我们更推荐Dependabot，因为它支持多种语言并且简单易用。

在开发前端应用程序时，我们在之前的技术雷达中提到了Headless Chrome作为前段测试中PhantomJS的更好替代方案。现在我们建议评估**HEADLESS FIREFOX**作为这一领域的可行选项。与Headless Chrome相同，Firefox以Headless模式运行浏览器时没有可见的UI组件，可以更快地执行UI测试套件。

NSP是一款命令行工具，用于识别Node.js应用程序中的依赖是否存在已知安全漏洞。在Node.js项目的根目录下运行check命令，nsp会通过检查发布公告来生成安全漏洞报告。nsp提供了一种自定义check命令的方法，可以隐藏所有低于给定CVSS分数的安全漏洞，或者当检测到的任意一个安全漏洞的CVSS分数高于给定值，就会退出并显示错误代码。一旦通过gather命令保存了advisories（公告）之后，nsp也可以在离线模式下使用。

PARCEL是一款类似于Webpack或Browserify的应用打包器。我们在往期雷达中推荐过 Webpack，它现在仍然是款很好的工具。Parcel的特色在于良好的开发体验和打包速度。它不仅具有所有标准的打包特性，并且提供了真正的“零配置”体验，这些特点让它很容易上手和使用。它的打包速度很快，并且在多项性能测试中击败了对手。Parcel 现在已经在社区获得广泛关注，值得留意。

SCOUT2是一款用于AWS环境的安全审计工具。有了Scout2，不必人工浏览所有网页，就可以获取到AWS环境的配置数据；它甚至还能生成一份攻击面报告。Scout2带有预配置的规则，并且很容易扩展以支持更多的服务和测试用例。因为Scout2仅通过AWS的API去获取配置信息和发现安全隐患，所以无需提交AWS安全漏洞和渗透测试申请表。

SENTRY是一款错误追踪工具，可以帮助实时监控并修复错误。像Sentry这样的错误追踪和管理工具，与类似ELK Stack这种传统的日志解决方案有所不同，前者更关注发现、调查和修复错误。Sentry出现已经有一段时间了，并且非常流行——对于目前备受瞩目的“平均故障恢复时间”，错误追踪工具变得越来越有用武之地。因为能够与

Github、Hipchat、Heroku、Slack等平台集成，Sentry可以让我们更聚焦于自己的应用。它能在产品发布之后提供错误通知，让我们跟踪新的提交是否真正解决了问题，并且能在问题再次出现的时候进行通知。

在当今的技术服务中，越多越多的人采取暴露RESTFul API的做法，而且API文档对消费者来说也至关重要。

(Swashbuckle for .NET Core)

在当今的技术服务中，越多越多的人采取暴露RESTFul API的做法，而且API文档对消费者来说也至关重要。在这个领域中，很多团队开始广泛使用Swagger，而我们想重点推荐**SWASHBUCKLE FOR .NET CORE**。它是一款可以为.NET Core项目代码在Swagger中生成鲜活的API文档的工具。你还可以通过它的UI来浏览和测试API操作。

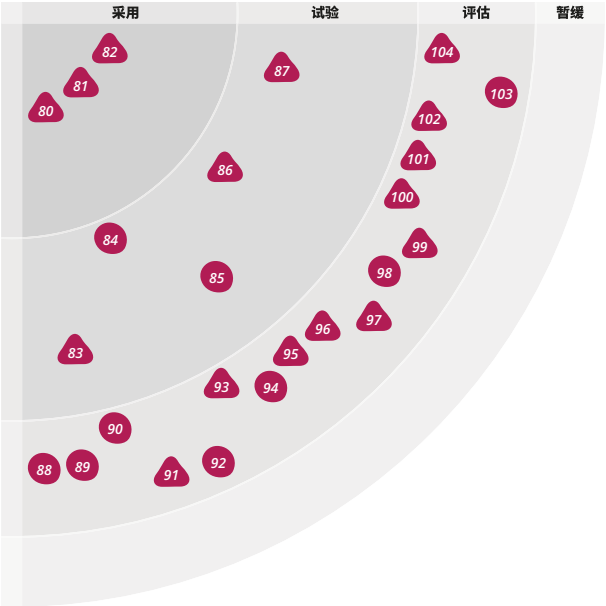
语言&框架

采用
80. AssertJ
81. Enzyme
82. Kotlin

试验
83. Apollo *NEW*
84. CSS Grid Layout
85. CSS Modules
86. Hyperledger Composer *NEW*
87. OpenZeppelin *NEW*

评估
88. Android Architecture Components
89. Atlas and BeeHive
90. Clara rules
91. Flutter *NEW*
92. Gobot
93. Hyperapp *NEW*
94. PyTorch
95. Rasa *NEW*
96. Reactor *NEW*
97. RIBs *NEW*
98. Solidity
99. SwiftNIO *NEW*
100. Tensorflow Eager Execution *NEW*
101. TensorFlow Lite *NEW*
102. troposphere *NEW*
103. Truffle
104. WebAssembly *NEW*

暂缓



ASSERTJ是一个提供流式断言接口的Java库，可以很容易在测试代码中表达测试的意图。AssertJ 提供了可读的错误消息、软断言以及增强的集合和异常支持。我们很多团队选择 AssertJ 作为默认的断言库，而不再是用JUnit和Java Hamcrest的组合。

ENZYME已经成为了测试React UI组件的事实标准。与其他基于快照的测试工具不同，Enzyme 可以进行无设备渲染的测试，速度更快，粒度更细。这很大程度上减少了在 React 应用里所需要的功能测试代码。在大部分项目中，我们会结合单元测试框架（如Jest）一起使用。

KOTLIN的使用率得到了飞速增长，工具支持也突飞猛进。其流行的背后原因包括语法简洁、空指针安全、易于从Java迁移以及与其他JVM语言的互操作性。并且，它还是非常不错的函数式编程入门语言。随着JetBrains 添加了新功能，允许在多平台将 Kotlin 编译为原生二进制文件，以及可以转译为JavaScript，我们相信，对于广大移动和原生应用开发者来说，它具备进一步广泛使用的潜力。尽管现在静态分析和代码覆盖率分析这样的工具还不成熟，但基于我们在许多产品应用上使用Kotlin的经验，我们相信Kotlin已经可以广泛采用。

自从第一次在雷达中提到GraphQL，我们就看到它在项目中被稳定采用，特别是将它作为BFF的远程接口。随着使用经验的提升，我们发现了一个GraphQL客户端 Apollo，将其作为React应用程序访问GraphQL数据的首选。虽然**APOLLO**项目也提供了服务端框架与 GraphQL 网关，但其客户端简化了 UI 组件与 GraphQL 后端数据绑定的问题。值得注意的是，Amazon AWS在最新启动的 AWS AppSync 服务中使用了Apollo。

Hyperledger项目现在已经发展成包含一系列子项目的大工程。针对不同业务需求，可以支持不同的区块链实现方式。例如，Burrow专门用来实现带权限控制的Ethereum，而Indy更专注于数字身份。在这些子项目中，Fabric是最成熟的一个。当开发者们谈到使用 Hyperledger 技术时，实际上大多数时候是在考虑 Hyperledger Fabric。然而，chaincode的编程抽象相对底层，因为它直接处理账本的状态数据。此外，在编写第一行区块链代码之前，搭建基础设施也经常耗去很多时间。**HYPERLEDGER COMPOSER** 构建于Fabric基础之上，加速了将想法实现为软件的过程。Composer 提供 DSLs 来建立业务资源模型、定义访问控制和构建业务网络。使用 Composer，可以在不搭建任何基础设施的情况下，仅通过浏览器来验证我们的想法。需要明确的是，Composer 本身并不是区块链，仍然需要把它部署在 Fabric 上。

安全是区块链经济的基石。

(OpenZeppelin)

安全是区块链经济的基石。在上一期技术雷达中，我们强调了测试和审计智能合约依赖的重要性。**OPENZEPPELIN**是一个能够帮助开发者用Solidity 语言构建安全智能合约的框架。OpenZeppelin的团队围绕智能合约的安全性总结了一系列陷阱和最佳实践，并将这些经验之谈嵌入到源码中。这个框架得到了开源社区的充分审核与检验。我们推荐使用OpenZeppelin，而不是自己编写ERC20/ER721 的通证。同时，OpenZeppelin也与Truffle做了集成。

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。

我们一直在问自己，多年来雷达推荐了无数JavaScript应用程序框架，还有必要再列一个新的吗？**HYPERAPP**因其极简的风格脱颖而出。它占用的空间非常小（小于 1KB），但却涵盖了编写 Web 应用程序的所有基本功能。只有优雅的设计才能将所有东西都减到极致，使其更易于理解和使用。尽管它相对较新，但它吸引了一个规模庞大的社区，我们建议在为新应用程序选择框架时，至少可以考虑它。

RASA是聊天机器人领域的新成员。它并非使用简单的决策树，而是通过神经网络将用户意图和内部状态映射到回应上。Rasa 集成了自然语言处理解决方案（spaCy）。与技术雷达中的其他同类工具不同，Rasa是开源软件，可以自行托管，对于担心数据所有权的使用者来说 Rasa 是一个可行的方案。我们在内部应用中使用了Rasa Stack，效果良好。

REACTOR是一个基于Reactive Streams规范的、用于开发非阻塞式应用程序的 JVM 库，支持 JVM 8 及以上版本。响应式编程强调将命令式逻辑转换成异步、非阻塞和函数式风格的代码，特别是在处理外部资源时。Reactor 实现了 Reactive Streams 规范，并且提供了两个不同的发布者 API：Flux（0 到 N 个元素）和 Mono（0 或 1 个元素），可以高效地对基于推送的流处理进行建模。Reactor 项目非常适合微服务架构，并且为HTTP、WebSockets、TCP和UDP等提供了支持背压（backpressure）的网络引擎。

RIBS即路由器（Router）、交互器（Interactor）和构建器（Builder）的缩写，是来自 Uber 的跨平台移动架构框架。RIBs的核心思想是将业务逻辑从视图树中分离出来，从而确保应用程序由业务逻辑驱动。可以将其看作是Clean Architecture模式在移动应用程序开发领域的一次应用。通过在原生 Android 和 iOS 应用上使用一致的架构模式，RIBs为应用提供了清晰的状态管理模式和良好的可测试性。尽管我们一直建议尽量将业务逻辑放在后端服务，不要将其泄漏到前端视图中，但移动应用程序非常复杂，RIBs可以帮助管理这种复杂性。

我们青睐异步和响应式编程，特别是对于网络I/O密集型的分布式系统。响应式类库通常位于较低级别的非阻塞通信框架（比如Netty）之上。最近，源自Apple的开源非阻塞网络框架SWIFTNIO吸引了我们的注意。**SWIFTNIO**与Netty类似，但却是用 Swift 编写。目前，它已经被 MacOS 和 Ubuntu 所支持，并且实现了HTTP作为更高级别的协议。我们很高兴看到这个框架的使用，并将其集成到更高级别的应用框架和其他协议中。

在上一期技术雷达中，我们推荐了PyTorch，它是一种支持指令式编程风格的深度学习框架。现在，通过在会话上下文之外执行建模语句，**TENSORFLOW EAGER EXECUTION**为TensorFlow实现了同样的指令式风格。随着 TensorFlow 模型的普及和性能的提高，这种改进可以提供更方便的调试环境，带来比 PyTorch 更精细的模型优化。这个功能相当新颖，我们对其表现拭目以待，也期待看到 TensorFlow 社区对它的接受程度。

TENSORFLOW LITE是我们上一期技术雷达中提到的TensorFlow Mobile的指定接班者。和TensorFlow Mobile一样，TensorFlow Lite是为移动设备(安卓和iOS)做调整和优化的轻量级解决方案。我们预期的应用场景是将预训练模型部署到移动端的应用程序中，令人惊喜的是，TensorFlow Lite甚至还支持在设备上进行学习，这样其应用领域会更加广泛。

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。

(WebAssembly)

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。

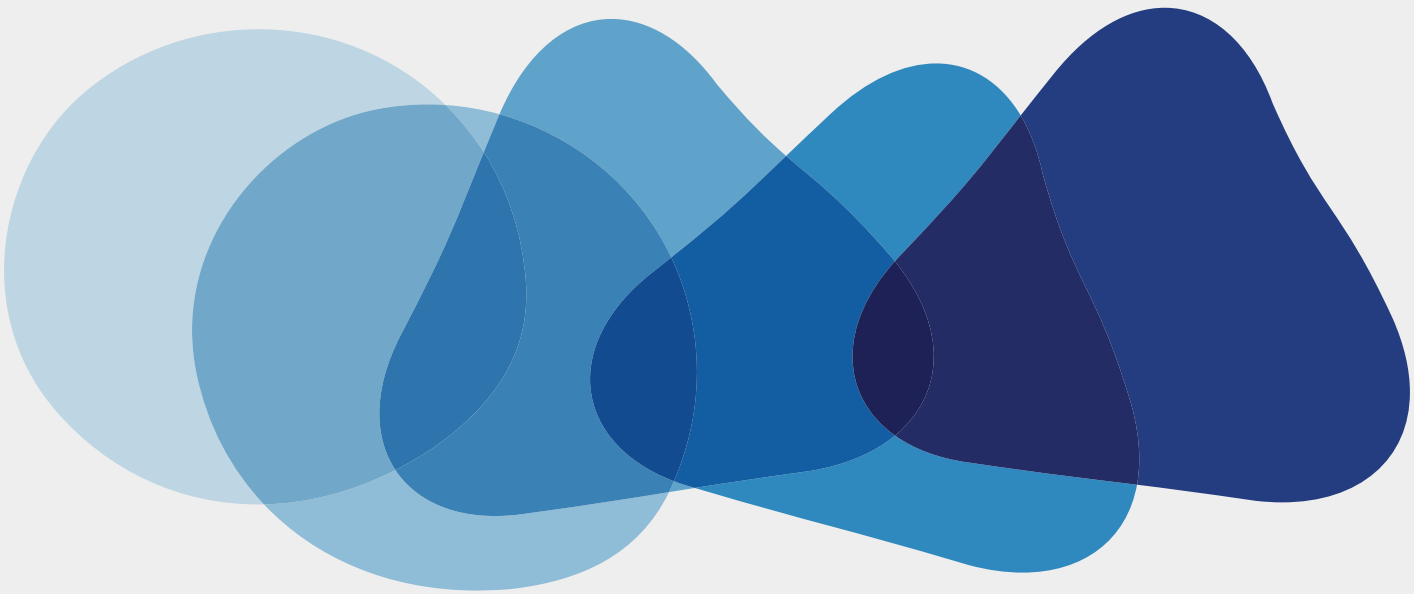
第一时间获知最新版技术雷达发布消息，以及独一无二的技术研讨会与深度内容。

点击订阅

thght.works/Sub-CN

ThoughtWorks®

ThoughtWorks已经从20多年前的一个小团队，成长为现在拥有超过4500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