第11章

治理和标准化：寻找平衡

微服务的卖点之一是能够选择“适合工作的工具”。但是，编程语言或数据存储层的灵活性会增加您的资产的复杂性，并可能增加成本和风险。

您需要找到合适的平衡点。

在本章中，我将深入探讨如何管理对所使用的技术进行选择的过程。

我想首先谈谈为什么您需要了解您的财产：正在使用哪些技术以及哪些版本。这很重要，因为它可以帮助您掌控安全漏洞和成本方面的风险。

然后我将转向护栏，它引导人们做正确的事情并保证他们的安全。当您拥有一支自主授权的团队时，护栏比规则和限制更有效。它们有助于确保每个人都了解他们应该做什么以及为什么这样做。

最后，向人们展示应将精力集中在哪里是一件好事：让他们深入了解系统的当前状态，并为下一步做什么以让事情变得更好提供指导。当您必须响应问题（例如主题访问请求或安全漏洞）时，这种洞察力也会有所帮助。

但让我首先定义治理的含义并解释它的重要性。

为什么治理很重要

治理是为了降低技术风险,并且不可避免地涉及一定程度的标准化。这可能与您使用的技术无关;而是与您使用的技术有关。它也可能与对任何技术的期望有关。如果您不使用铺好的道路,您需要在系统中构建什么?

“治理”听起来像是有人告诉你该做什么,这让它看起来很没有吸引力!但治理就是定义组织的良好面貌,确保人们了解这一点,并在人们没有按预期方式做事时提供自我纠正的方法。根据我的经验,工程师故意不满足期望的情况很少见。他们更有可能没有意识到这些问题。良好的治理包括有效的沟通和报告,使团队能够找出需要采取行动的地方。

了解您的遗产

当您运营微服务架构时,良好治理的最后一个方面(向人们展示他们需要在哪里采取行动)可能是一个挑战,因为要做到这一点,您需要知道您的财产中有什么。然而,通过微服务,您可以拥有数百甚至数千个服务,并且可以有许多不同的数据存储、语言和托管解决方案。

如果您不了解自己的资产,就无法知道有人在没有适当监控或日志记录的情况下将服务部署到生产环境中。您不知道有人在哪里购买了没有支持合同、没有单点登录集成、没有定义数据保留策略的 SaaS 解决方案,所有这些都可能会意外违反处理 PII 数据的策略。[1]了解您的资产还涉及了解您使用的库及其版本,以便您可以确定库中的安全问题是否已在所有地方得到修复。

了解您的财产从所有权开始,如第 9章所述,但当您超越这一点时,它可以成为控制成本和风险的强大工具。

正如我在前面的章节中提到的,在英国《金融时报》,我们建立了自己的系统来跟踪我们的财产信息,但我现在不会这样做。

您可以从技术含量相当低的解决方案中获得很多价值,例如在电子表格中列出所有服务以及有关它们的关键信息。电子表格允许您限制特定列的选择,并且您可以过滤或搜索。

[1]如果一个单独的部门决定雇用自己的合同开发人员或自己采购系统,这种情况通常可能完全在技术之外发生。

源代码管理是跟踪所有权的好方法。例如,在 GitHub 中,您可以在组织内设置团队,以便清楚谁拥有特定存储库。您还可以使用CODEOWNERS 文件来定义谁负责存储库 - 这可以支持更细粒度的所有权意识。

您可能会发现这些解决方案变得难以使用,因为您有很多服务需要跟踪。幸运的是,正如第 161 页的“服务目录”中所讨论的,如果您准备在这方面花费一些时间或金钱,那么可以使用一些解决方案(例如 Backstage、OpsLevel 或 Cortex);你不需要构建一些东西。

无论您使用什么,都应该在其中存储尽可能多的信息,因为能够跟踪数据之间的链接非常强大。

什么样的信息是相关的?

在 FT,我们存储的核心信息是关于系统(代表单个微服务)、产品(例如 ft.com 网站,由许多微服务系统组成)、团队和人员。有关此图的哪些部分的提示,请参见图9-1。

但随着时间的推移,我们添加了很多其他信息。这包括:

- GitHub 存储库。我们导入了所有这些信息,并在可能的情况下,通过匹配存储库名称和系统将它们链接到系统
  姓名。
- GitHub 团队。这些并不总是映射到组织内的团队,因此捕捉这一点很有价值。 · Heroku 团队和应用程序。
- AWS 账户、区域、资源。
- DNS 区域。
- 事故。我们将生产事件与所涉及的系统联系起来。这意味着当新事件发生时,我们可以轻松地查看历史,看看是否能给我们任何线索。事件将在第 13 章中讨论。 · API 网关密钥。该系统使用这些 API 来调用其他 API。

  这对于识别调用系统很有用。

为您的组织提供价值的内容会有所不同,但我的经验是,添加的每个新事物都可以让您提出更有趣的问题,并允许您自动化治理流程的某些部分 - 例如,机器人检查值资源上的标签数量与服务目录中接受的系统列表相匹配。

护栏和政策

您的技术组织希望每个团队做哪些事情？
新工程师如何知道它们是什么？

我怀疑对于很多组织来说，这是隐性知识。当你与认识的人交谈时，或者更糟糕的是，当你弄错并且出现问题时，你就会发现这一点。

正如我在第六章中讨论的，自治并不意味着混战。团队也有责任。这些职责包括构建安全、经济高效、可支持的系统。但您不希望每个团队对“安全”的含义都有自己的定义。

当我第一次在英国《金融时报》工作时，我们有很多政策和标准。这些通常由架构、网络安全或基础设施等中央团队定义。
此类政策和标准是治理的核心，因为它们规定了组织内部的期望。我可以自信地说，大多数开发人员没有阅读这些标准，甚至可能不知道它们的存在。

然而，由于我们在一小部分技术的限制下工作，并且通常将部署和操作系统的责任移交给另一个团队，因此作为 API 团队的高级工程师，我是否理解我们的系统并不重要。打补丁策略，举一个例子。

随着团队选择自己的技术、将系统部署到生产环境并为其提供支持，每个团队都需要了解对他们的期望。

我认为最好将这些信息视为一组护栏，而不是一组政策或标准。他们的存在是为了支持人们并阻止他们伤害自己。

在英国《金融时报》，我们最终决定将其作为清单来处理，涵盖生产系统的生命周期。

这些护栏对所需内容进行了简要说明，并链接到详细的政策文件。

然而，很少有人会在入职过程中寻找并阅读装满文档的驱动器，即使对于那些这样做的人来说，也没有简单的方法让他们知道标准已经改变。

我们尝试尽可能地将护栏纳入我们构建的工具中，这样即使人们从未阅读过护栏，也会被引导做正确的事情。这尤其适用于作为铺好的道路一部分的工具，因为它的部分价值是让团队不必考虑这么多不同的需求。

有些护栏比其他护栏更重要。如果您在受监管的行业工作,或者护栏与明确的立法相关,那么您想要的不仅仅是指导人们做正确的事情,还要确保您正在验证他们是否已经做到了。

这就是为什么护栏变得格外重要,当团队为某事构建自己的解决方案时,团队确实需要阅读政策等。作为离开铺砌道路的一部分,他们必须确保他们的路线符合护栏。

这就是车队在出发前应该仔细考虑的原因之一:他们需要做更多的工作。

自动化护栏虽然护栏为任何构建新

服务的人提供了有用的清单,但真正的优势来自自动化,它们被纳入工具和服务中,以便人们可以自动指导做正确的事情。

这意味着您不需要依赖人们端到端地阅读护栏,也不需要担心人们跟上期望的变化,因为您可以改变自动化和他们会这样找到答案的。

它还解决了庞大的软件产业中的护栏问题之一:您如何知道每个团队是否实际上都遵守了护栏?您不想通过未修补的系统的安全漏洞,或者由于团队正在“尝试”一种新工具而实际上没有人可用而无法向服务部署修复程序来发现这一点帮助有访问权限。

当您即将进行合规审核时,您尤其不想发现这一点!

如果您有用于启动新服务的模板或 API,那么它们应该符合护栏。例如,该服务应该设置一个执行安全扫描的构建管道,应该验证日志的格式,然后自动发送到日志聚合工具等。本质上,将护栏嵌入到您构建的每个工具中。

护栏自动化有助于收集数据,然后您可以使用这些数据来了解整个软件资产,以了解整个组织在特定措施上的表现,并推动团队在他们没有采取的行动中采取行动正确的事情。

我并不是说一切都应该自动化。有些治理太困难或太耗时,无法实现自动化。然而,问问自己自动化是否是一种选择是一个不错的方法。

包括什么

护栏应包括哪些类型的东西?

我们将护栏定义为“我们希望团队考虑的事情,以便我们构建正确的事物并构建正确的事物。”这意味着重点是帮助解决工程师不经常做的事情。

Guardrails 应该成为与专家对话的替代品,避免与团队外部人员协调的需要。遵循护栏应该产生安全、可靠且可操作的系统。

在考虑自己的护栏时,请考虑生产路径、您作为组织对质量、安全性和运营问题的期望,并留意开发人员经常询问的问题。

英国《金融时报》的护栏

我离开时FT的护栏如图11-1所示。

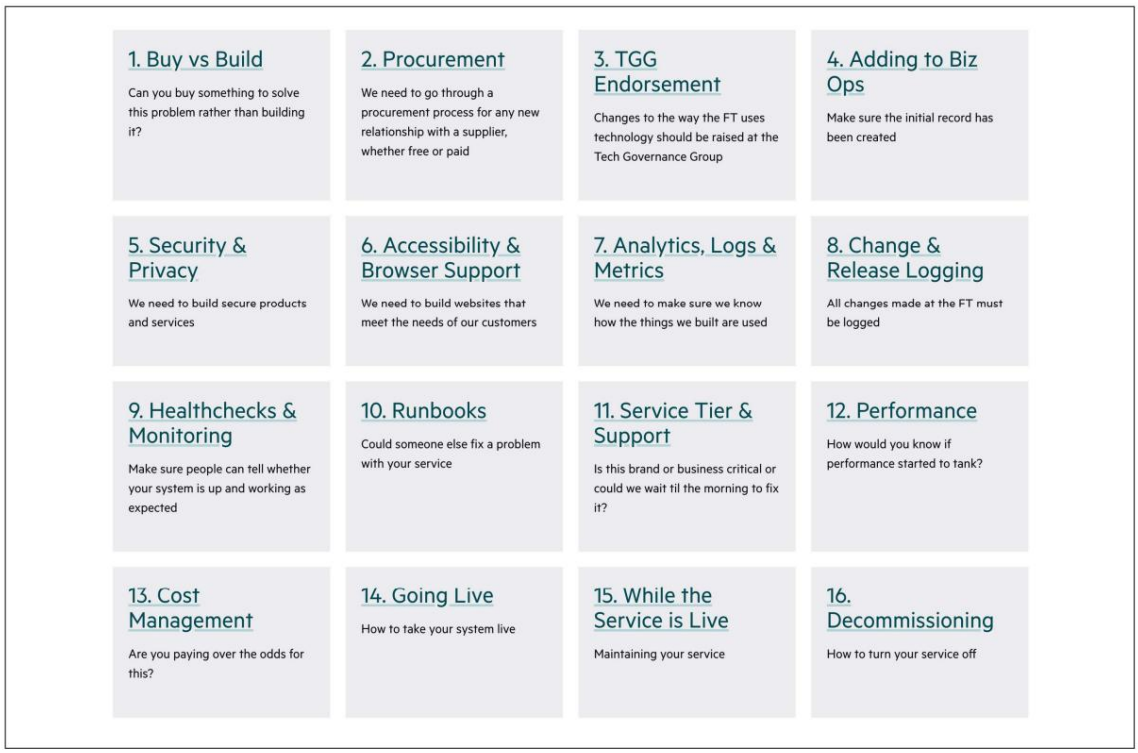

图 11-1。 2022 年 e FT 的护栏。

我们认为工程团队熟悉开发生命周期:代码、测试、审查。我们认为我们不需要为这些活动提供护栏;相反,我们信任团队能够处理这个问题。当代码开始进入生产阶段时,我们就介入了。

对于英国《金融时报》技术部门来说,首要目标是让人们更容易理解什么是好的。

我们按照您在构建新服务时需要解决的顺序松散地列出了我们的护栏。尽管它们看起来很多,但实际上,它们中的大多数都构建在我们铺好的道路和团队自己的工具中:大多数服务将使用应用程序的模板创建,该模板可以整理日志记录、运行状况检查等内容,更改日志记录和安全扫描。

好吧,让我们深入研究一下这些。

当然,它们可能并不都对您的组织有意义。在本章后面,我将介绍其他一些不同规模和不同监管级别的组织处理治理的方式,以增加该主题的广度。

1. 购买与构建

您能购买一些东西来解决这个问题而不是构建它吗?

团队通常会直接跳过这一步,但在构建解决问题的东西之前,您应该看看是否可以购买现成的东西。

您在组织内添加的每一项新服务都会带来相关的成本和风险。
人们关注构建某些东西的成本,但持续支持和维护的成本可能会超过该成本,因此需要将其纳入比较中。

对于非核心业务的事物,您应该尝试让其他人来构建和运行它,从而最大限度地降低运营成本。现在许多服务都以 SaaS 形式提供,这使您的团队能够腾出时间来处理具有业务优势的事情。

还有第三种方法:你能通过现有软件的组合来组合出你需要的功能吗?通常,这是构建一个包装器来调用一个或多个 API 并管理围绕它的流程的问题。

2、采购

对于与供应商的任何新关系,无论是免费还是付费,您都必须经历采购流程。

人们倾向于直接构建而不是购买的原因之一是工程师往往认为采购是一件令人头疼的事情。很多表格需要填写!

然而,采购部门可以成为一个极好的合作伙伴。就我个人而言,我不擅长谈判价格、评估合同风险或评估一家公司是否有可能在三年内生存。我想把它交给一个能做得更好的人。

在英国《金融时报》,我们拥有一支优秀的采购团队,他们了解工程团队想要的工作方式。概念验证 (POC) 工作有一个轻量级流程,旨在将预先填写的表格保持在最低限度。

我们仍然需要提醒工程师,不,他们不应该在不让采购团队知道的情况下使用公司信用卡进行注册,甚至不应该开始免费试用。

这使得采购部门能够及早发现问题,例如,团队似乎正在购买与《金融时报》已经存在的东西直接等同的东西,但没有明确的理由说明原因。

3. 重大技术变革需要讨论技术使用方

式的重大变革。

这个护栏是为了防止不必要地引入新技术。
每项新技术都涉及相关成本,有时,人们引入一项新技术仅仅是因为他们不知道已经有适合他们用例的技术。您还想知道是否有人已经尝试使用该技术并遇到问题。

在英国《金融时报》,我们使用了一个名为“技术治理小组”(TGG) 的论坛。这是一个轻量级的流程,旨在确保对《金融时报》如何使用技术的重大变化进行公开讨论。我将在本章后面详细讨论它。

4. 创建服务记录确保已创建服务的初始

记录。

当您拥有大量服务时,跟踪它们很困难。只要您有服务的基本记录,您就可以自动跟进缺失的信息。

我们确实希望每项服务都有 Biz Ops 记录,因为这就是赋予信息图力量的原因。

最初的记录很短,只需要几个字段。系统是在“预生产”状态下创建的。将它们转移到“生产”状态(上线过程的一部分)会促使团队填写更多数据,但初始记录足以获得很多价值。

我们对这一护栏的合规性很高,因为我们在很多地方使用了独特的系统代码来提供服务。
AWS 资源已被标记;它被输出到日志中;它用于创建警报和监控,并将其链接到适当的团队仪表板。

设置初始 Biz Ops 记录是获取该系统代码的方法。2

5. 安全和隐私您需

要构建安全的产品和服务。

安全和隐私是需要支持团队专业知识的两个领域。
这些团队可以定义其他工程团队需要做什么,并使他们能够轻松地做到这一点。

在英国《金融时报》,我们的安全工程团队定义了安全工具,包括各种类型的安全扫描,包括部署管道中的安全扫描。团队需要确保他们的系统集成了这些工具(我们可以直观地看到这种集成缺失的地方)。

在《金融时报》,我们还尝试在此过程的早期捕获具有特别敏感数据的系统的信息:Biz Ops 中有一些字段可以定义系统是否处理个人身份信息 (PII),因为这具有 GDPR 含义,即,如果弄错了,可能会损失很多钱。

然后,我们可以采取一些措施,例如防止将任何标记有包含 PII 的系统代码的 S3 存储桶设置为公共存储桶。

6. 辅助功能和浏览器支持您需要

构建满足客户需求的网站。

与安全和隐私一样,可访问性并不是每个团队都具备的技能。该护栏提供了有关如何构建可访问网站的信息。

我们还希望有一个地方来定义应支持哪些浏览器,以便工程师了解他们需要处理什么,以便测试人员可以确保适当的覆盖范围。

7. 分析、日志和指标您需要

确保知道您构建的东西是如何使用的。

分析、日志和指标都可以通过将它们连接在一起而获得很多价值。您不想在安全事件期间发现

2由于 Biz Ops 拥有出色的 API,因此我们可以验证用于标记资源的系统代码实际上代表了 Biz Ops 中的记录。

受影响的服务被发送到团队设置的完全不同的日志记录目的地,并且您无法访问!

该护栏还提供了有关如何决定记录内容以及捕获哪些指标的指南。

刚开始做开发的时候,写的日志一般都是我在本地工作时用的东西。我有很多可以打开的调试日志(如果我没有使用调试器),但其他级别的日志记录很少。如果生产中出现问题,我可能会尝试在本地重现它。

在微服务世界中,我发现这要困难得多,并且我真正开始看到在始终输出的日志级别做出的日志记录决策的价值。这是我必须从痛苦的经历中学到的东西;最好通过记录对您应该记录的内容的期望来指导人们。

除此之外,这个护栏是我们指定日志格式和我们想要确保包含的信息的地方。例如,由键值对组成的结构化日志允许在我们的日志聚合工具中进行更强大的查询,并且系统代码和唯一标识请求的相关ID等字段在尝试挖掘时也非常有用。可能发生的事情(我们没有专门的以可观察性为中心的软件,所以我的很多调试都是通过日志查询进行的!)。

同样,对于指标,这是定义每个系统应生成的指标的地方,以提供资产健康状况的全局视图。

8. 变更和发布日志记录必须

记录对生产的所有变更。

您确实希望能够判断系统是否在代码发布后立即停止工作。
在《金融时报》,我们有一个 Change API,可以很容易地连接到标准部署管道中 在其他集成中,编写它的团队创建了一个 CircleCI Orb,这使得许多团队可以轻松地包含它。

我们希望团队在代码投入生产时调用 Change API。以这种方式通知的每个更改都会写入许多不同的位置,包括 Slack 通道和我们的日志聚合工具。这使得任何人都可以轻松查看生产中出现问题时发生了什么变化。

我们还发现,随着代码更改到同一系统中,编写基础设施很有价值,例如,更改 DNS 设置。这为我们提供了一个地方来查看生产中发生的变化。

我们编写 Change API 的目的是为了轻松地向其写入内容并轻松使用它。 Nikita Lohia 的博客文章“The Advent of Change···API”中描述了该架构。架构选择意味着来自

Web 团队将功能标志更改写入其中并将这些更改发送到数据平台是完全可能的,而 Nikita 的团队无需执行任何操作。

9. 健康检查和监控确保人们

可以判断您的系统是否正常运行并按预期工作。

当标准化时,监控会获得很大的价值,即,您可以在每个服务的同一位置找到监控,并且检查看起来相似。

这个护栏定义了健康检查应该是什么样子。第 359 页的“健康检查”对此进行了更详细的讨论。它还定义了如何在我们的团队仪表板和主 FT 操作仪表板中进行监控。

10. 操作手册

其他人可以对您的服务提出问题吗?

我们希望《金融时报》的每一项服务都有一个运行手册,提供信息来帮助任何试图在生产中支持该服务的人。

重点是有助于排除故障的信息。例如:代码在哪里?服务部署在哪里?可以进行故障转移吗?

我们与一线运营团队合作,为操作手册定义了一个标准模板,重点关注从未查看过此服务的人可能需要的信息,这些信息通常也是查看此服务的人需要的信息。几年来第一次想看到副总统。

其中包括有关该服务的服务级别的信息:如果出现故障,是否需要在非工作时间进行呼叫?我们记录了该服务是否运行主动-主动或主动-被动(其中有两个区域,它们是都提供流量还是其中一个处于备用状态,仅在发生故障时使用?)以及任何故障转移机制。我们有关于如何恢复任何数据的备份的信息。其中包含有关在哪里可以找到源代码、日志、指标和监控的信息。

我们还提供了任何有用的故障排除提示。

我们的操作手册实际上是 Biz Ops 中保存的服务信息的子集。
您编辑了 Biz Ops 记录,并且有一个单独的 Runbook 视图,它比 Biz Ops 本身具有更高的弹性(以避免必须跨多个区域运行图形数据库:定期提取 Runbook 数据并将其写入 S3)。3

3在我们的单点登录解决方案发生事故后,我们最终将其压缩并放入 Google 云端硬盘中,导致我们无法访问 Runbook,因为它支持单点登录。叹。

正如第 7章中所讨论的,在采用微服务的早期,我们的操作手册通常状态不佳。有很多信息缺失。

我们对每个运行手册的质量进行自动评分（承认我们不一定第一次就得到正确的权重）。这帮助我们达到了信息普遍存在的地步。然后挑战就会发生变化,因为自动检查信息的准确性及其是否是最新的要困难得多。然而,我们确实在评分中添加了一项检查,这样“TODO 写这个”就不算数了!

在大多数运行手册存在时,我们对最关键服务的运行手册信息进行了更加手动的年度审查,以验证其准确性。

11. 服务等级和支持这个品

牌或业务是否重要,或者您可以等到早上再解决吗?

决定服务需要什么级别的弹性和冗余以及什么级别的支持,会对构建和运行服务的成本产生影响。通常,讨论会留到构建周期的后期,只有在距离上线数周甚至数天后,您才会发现团队和利益相关者的不同期望。这可能会给团队带来压力,要求他们在非工作时间支持一些没有考虑到弹性的事情。

例如,如果某项服务需要高可用性,您可能需要让它在多个数据中心或云区域中运行。运行跨区域的架构比在单个区域中运行的架构更复杂:多区域数据库集群并不简单。依赖关系还带来了额外的复杂性。如果您的新服务需要高可用性,但依赖于非高可用性的服务,那么您可能会遇到比预期更多的事件。

在英国《金融时报》,我们定义了四个服务级别:白金、金、银和铜。白金级系统具有高度弹性并支持 24/7,而青铜级系统则不然。

该护栏试图为决定服务级别提供启发式方法,并且清楚选择它的后果。如果不满足与白金级服务层相关的增强弹性要求,您就不能要求非工作时间支持 - 因为除非您投资于弹性,否则您不应该期望工程师能够 24/7 为系统提供支持!

在英国《金融时报》,我们之所以能做到这一点,是因为我们得到了工程和产品高级领导层的支持。总是存在着快速启动某些东西并随后解决弹性问题的压力 - 有时,这是可以接受的,但您确实需要接受出现问题的额外风险。然而,我认为这是提供的-

正在为青铜级服务提供额外支持,并期望与团队成员进行一些讨论,以适合他们的方式进行排列。

12. 性能

你怎么知道性能是否开始下降?

这提醒我们,服务的性能很重要,了解服务何时发生变化也很重要。除此之外,它还提醒您在构建服务的过程中尽早与利益相关者进行对话,了解什么样的性能是可以接受的。

13. 成本管理您为

此付出的代价是否超出了赔率?

在云中运行为工程师提供了很大的灵活性,但这也意味着工程师可以启动资源而无需考虑成本。

成本管理从您启动的基础设施的合理默认值开始,这是中央团队可以产生影响的地方 - 例如,通过切换平台以使用可用的新型处理器。4

向团队和领导层展示成本的工具非常有用,因为没有这种可见性的团队可能会产生成本,而他们可以通过减少日志记录量或缩小过度配置的服务器来轻松削减成本。为此,您需要对资源进行准确标记,以便了解所有权,但出于多种原因,这是一个好主意,如第9章所述。

重要的是要了解您在构建和运营系统方面的支出超出了您预期的收入。粗略的总拥有成本 (TCO) 计算在这方面确实很有帮助。它不需要精确,只要在正确的数量级即可。例如,您可以根据时间和团队规模并使用标准日费率来分配构建服务所需的成本。一旦建成,您可以评估维护它的持续成本。您可以查看运行该架构的成本。

在《金融时报》,我们发现 TCO 计算可能会以我们意想不到的方式进行:例如,我们有一个旧系统,是用我们不再使用的编程语言编写的,由一个不再存在的团队编写,这意味着没有积极的所有权,但没有明显的团队来承担。这个过时的系统存在风险,而且没有兴趣重写它。在停用它之前,我们进行了 TCO 计算,结果显示该组件带来的广告量

4例如, Honeycomb通过改用 AWS 基于 ARM 的 Graviton2 处理器,节省了 30% 的费用。

让它变得超级有利可图!这促使我们决定投资重建该系统并为其找到合适的住所。

我可以想象使用成本管理护栏来显示当前的资源成本和可以节省资金的替代方案:例如,切换实例类型或更改架构以避免移动大量数据。所有这些都可以自动化。

14. 上线您

如何上线您的系统?

人们很容易忘记最终推出新功能或产品的过程中的步骤。

该护栏提供了一个迷你清单,其中包括填写其余业务运营服务记录、创建运营运行手册、记录任何已知的技术债务以及移交给一线运营团队等步骤。这些事情很容易被团队在急于跨越界限时忘记,但有可能在以后引起问题。

15. 当服务上线时

维护您的服务。

服务需要持续的关注和关注,例如,修复通过扫描工具发现的安全问题;升级版本 最好在版本生命周期结束之前进行;并测试数据存储的备份和恢复过程。

再次强调,最好提供一份清单。更好的是自动捕获这些信息,以便您可以标记关注的区域。

例如,英国《金融时报》创建了仪表板,汇集了已知安全问题的视图,特别关注超出了部署补丁的目标时间的高级漏洞。

我们还存储了有关编程语言的信息。我们可以查看我们的源代码控制,搜索某种语言的过时版本,然后点击受影响的存储库中的链接访问服务(并不完全顺利,但我们可以做到!)。

16. 停用您如何关闭

您的服务?

很容易错过退役步骤。留下的任何东西都可能会带来安全风险,并且可能会带来持续的成本。

这是(又一个)迷你清单,涵盖了在生产中关闭服务之外的步骤。

是否已从监控中删除?相关的 DNS 条目是否已删除?该服务是否已在 Biz Ops 中设置为“退役”?基础设施资源是否被关闭?

目的是让人们轻松知道他们需要做什么,即使他们以前从未这样做过。

在护栏上对齐

英国《金融时报》最初的护栏和政策是由我们的架构团队定义的,规定了团队需要做什么的期望。它们是全面且经过深思熟虑的,但是作为当时领导开发团队的人,它们并没有融入到我们的实践中。将它们作为清单呈现的转变使我的团队更容易找出需要做什么并找到相关信息。

这一转变伴随着一场大型的沟通活动:海报、演示、电子邮件、Slack 消息。我们知道这些事情是我们的责任,因为我们现在受益于自主权。

随着时间的推移,建筑逐渐成为人们所做的事情,而不是人们扮演的角色。架构发生在每个团队内部,不再有一个单独的团队自上而下地定义护栏。

相反,我们拥有一个论坛来讨论并就具有广泛影响的技术决策达成一致,我们将其称为技术治理小组 (TGG)。5

技术治理小组英国《金融时报》技术

治理小组的成立是为了在有足够的信息来评估重大技术变化时对其进行审查:通常是在进行一些调查、潜在的概念验证工作和一些规划之后,但在进行大量投资之前时间和金钱。

TGG 流程有三个主要目的:

- 为人们提供一个清晰、轻量级的流程来分享想法、接收反馈并获得继续进行的认可 · 让其他人轻松了解即将到来的主题,并在相关的情况下为这些决策提

  供意见

- 允许人们回顾并发现为什么做出特定决定
  制成

5这是现有会议的重新启动,只是保留了相同的名称。我们或许应该有
更改了名称,因为这与其说是关于治理,不如说是关于沟通和共识。

我将稍微解释一下这些内容以解释其价值,但首先我想谈谈会议的形式,以及预计将向 TGG 提交哪些技术决策。

我应该指出,这本质上满足了与架构决策记录 (ADR) 和评论请求相同的需求,并借鉴了其中的想法和格式。6

TGG 格式

TGG 定义了技术提案的结构,可作为模板文档使用。
目的是让它变得轻量级,并让该结构帮助人们以一致的方式填写提案。

提案模板包括:

作者 提案
: 的作者,无论是个人、团队还是任何其他群体。他们是该倡议的发起者并对此负责。

需要
: 需要解决的问题。这应该涵盖当前现状中不可行的地方以及改进的机会。这也是声明提案范围之内或之外的内容的地方。

建议的方法 作者首选的
: 方法。

已知的限制和风险
: 所提议方法的已知限制和/或风险,以及任何可能的缓解措施。

影响 采
: 用所提议的方法将会对什么或谁产生影响,无论是积极的还是消极的。

成本
: 实施和维持该提案的任何预期资本支出和运营支出成本。
这包括供应商合同、许可和任何预期的按需基础设施成本。如果需要人员来实施该提案,则应提供所需人数和持续时间的范围或估计。这可能是工程师最费劲的部分,但这是决策过程中了解成本的关键部分!

6我在《金融时报》的同事 Rob Godfrey 定义了我们使用的流程。干杯罗布!

贝内茨

这些可以是定量的（例如,金钱或时间节省)或更无形的（例如,对士气的积极影响,或风险缓解）。

备择方案

已考虑的替代方案。我们需要“不执行任何操作”选项,因为这有助于我们更好地理解潜在需求。

通常,这将是两页或三页内容。

提案在 TGG 会议之前通过公共 Slack 渠道和 GitHub issues 进行了分发。希望至少提前一周分发,但如果提案紧急或简短,分发时间可能会更短。关键是要给人们足够的时间阅读和评论。

责任我们非常清

楚,提案作者有责任确保提案得到合适的人员审核。一般来说,他们知道谁会关心,并且可以专门联系他们以寻求反馈。如果适用,这可能包括工程外部团体的反馈:例如合规性、财务或安全性。

英国《金融时报》拥有多个相当独立运作的开发团队,每个团队由一名技术总监领导。每个小组都应该派出一名代表参加 TGG 会议,通常是该小组的首席工程师之一。如果某个特定群体没有人参加会议,则认为他们支持该提案:如果他们关心某项提案,他们有责任确保有人出席。

尽管我当时没有读过这篇文章,但这与安德鲁·哈梅尔劳在他的文章“以对话方式扩展建筑实践”中提出的想法非常相似。

Andrew 将架构建议流程定义为:

> 规则:任何人都可以做出架构决策。
> 限定者:在做出决定之前,决策者必须咨询两组人:第一组是所有将受到该决定有意义影响的人。第二个是在正在做出的决策领域具有专业知识的人。

这正是我们在 TGG 所拥有的。提案者必须咨询受影响的人,提案被广泛、公开地传播,让那些有专业知识的人有机会做出贡献,而提案者不需要知道这些人可能是谁。

“具有专业知识的人”可以是主题专家（例如,具有 Kubernetes 知识的人）,但也可以是尝试解决同一问题的人,他们可能会告诉您什么有效、什么无效,并且希望能阻止你构建已经存在的东西,或者组织已经知道行不通的东西。

我记得读过一些组织召开会议来做出决策,而其他组织则召开会议来分享决策:工作发生在会议之前。根据我的经验,您可以在一个组织中召开这两种类型的会议,您只需要了解这个特定会议是什么类型。

TGG 是一次批准几乎已经做出的决定的会议。
这项工作发生在会议之前,因为我们要求任何将积极参加会议的人提前就提案文件提出疑虑或问题。

我在那里的时候,我认为我们有一两个提案没有得到认可。我们有一些必须修改的内容。但超过 90% 的提案都获得了批准。

那么为什么要召开这次会议呢?部分原因是这是一种共同的经历,这得到了认可。
而且,当我们在大流行期间都远程工作时,这一点尤其有效,任何人都可以参加旁听会议,有时我们会有多达 30 或 40 人参与。所有这些人都听说了这些提案,所以这是沟通细节的好方法。他们还了解了如何讨论架构决策以及提出的观点类型。会议得到了非常积极的主持和记录。我觉得他们真正展示了我们的工程文化。

需要什么技术提案让很多人在一

个房间呆一个小时的成本很高,而这种同步协作是您希望最小化的。然而,在具有广泛影响的事情上做出错误决定的代价会更高。

为了找到平衡点,我们不需要 TGG 签字即可开始概念验证工作。通常,人们会写好提案并在获取反馈的过程中开始工作（有时获取更具体的反馈会更容易）。但我们预计 TGG 会针对以下任何更改提出提案:

- 具有广泛的影响。这通常意味着工程中的多个团队必须投入大量时间或金钱,或者根据提案对其使用的流程或技术进行重大更改（例如,在源代码控制提供商之间移动）。通常,这些都是战略变化。

· 花费很多钱。对于经常性成本尤其重要,例如,添加具有年度许可成本的工具。 · 从本地化走向全球化。当一个工程团队使用的东西开

始被其他团队采用时,就是讨论这是否是正确选择的好时机。

这并不是一份详尽的清单;我们告诉工程师,“如果您认为应该向 TGG 提交提案,请将其提交。”

决定引入一项新技术不仅仅涉及该技术。还有其他考虑因素。

如果您需要具有特定技能的人能够充分利用它,您能在市场上找到具有这些技能的人吗?他们的雇佣成本更高吗?您现有的工程师想学习这项技术吗?

虽然我主要谈论的是工程师,但我们特意向 TGG 询问了其他人是否对工程以外的领域有潜在影响。

TGG 的优势

好吧,回到TGG的好处。是什么让这个有效?

清晰且轻量级的流程TGG 流

程是轻量级的:按照前几节提到的格式编写一份两页的提案,将其签入 GitHub 存储库,然后参加下一次会议。

人们仍然会拒绝将东西带到 TGG;然而,我的观点是,如果您不花时间写一份两页的提案,那么您就不应该引入重大的技术变革!

轻松了解正在讨论的主题， TGG 可以传播对

整个技术部门正在发生的事情的理解,这意味着当一个团队发现它正在重复其他团队已经做过的事情时,会减少令人讨厌的意外。

记录人们的想法即使人们已经写下了架构决

策,也很难找到它们。当我开始领导内容平台团队时,我找不到太多关于引入容器的书面讨论,尽管我当时就在团队中!

TGG 的好处是可以在一个地方找到提案,并且采用一种格式。

Olaf Zimmerman 讲述了架构决策的伟大历史,深入探讨了捕获决策如此重要的原因,并讨论了各种不同的格式。7在我看来,关键是能够回答经常出现的问题三年后,与新同事,甚至你自己相处:他们到底在想什么?

支持护栏的发展TGG 有两个主要提

案来源。大多数提案来自工程支持小组,涉及对铺设道路或护栏的更改。

这有助于让整个组织的工程师参与决策,因此他们不会觉得这是对他们做的。

TGG 提案的第二个来源更多地与自治团队及其选择技术的自由有关。只要这些选择可能产生广泛影响,我们就会要求将它们提交给 TGG。在支持团队自治的同时,这些选择确实需要公开做出并进行理智检查。

选择技术

如果您考虑一下您现在使用的技术,几乎可以肯定它不是您三年前使用的技术。

事情在不断变化。新的创新提供了新的功能或现有功能的更好版本。

例如,公共云为我们提供了自托管数据中心的替代方案。
最近,法学硕士的到来开辟了新的选择。

但是您如何决定何时引入新技术呢?有哪些启发法?

技术生命周期我想首先介绍两

种不同的方式来思考技术的生命周期。我发现某项技术的成熟度是考虑是否采用该技术以及评估潜在收益和风险的一个关键因素。

首先,我想谈谈技术采用生命周期,如图11-2所示。
该模型模拟了新技术如何被采用,并且是一条钟形曲线:少数

7请参阅“架构决策 制定过程”。

早期采用者随着人们相信这是一个安全的赌注而不断增长,后来只有少数人采取行动。

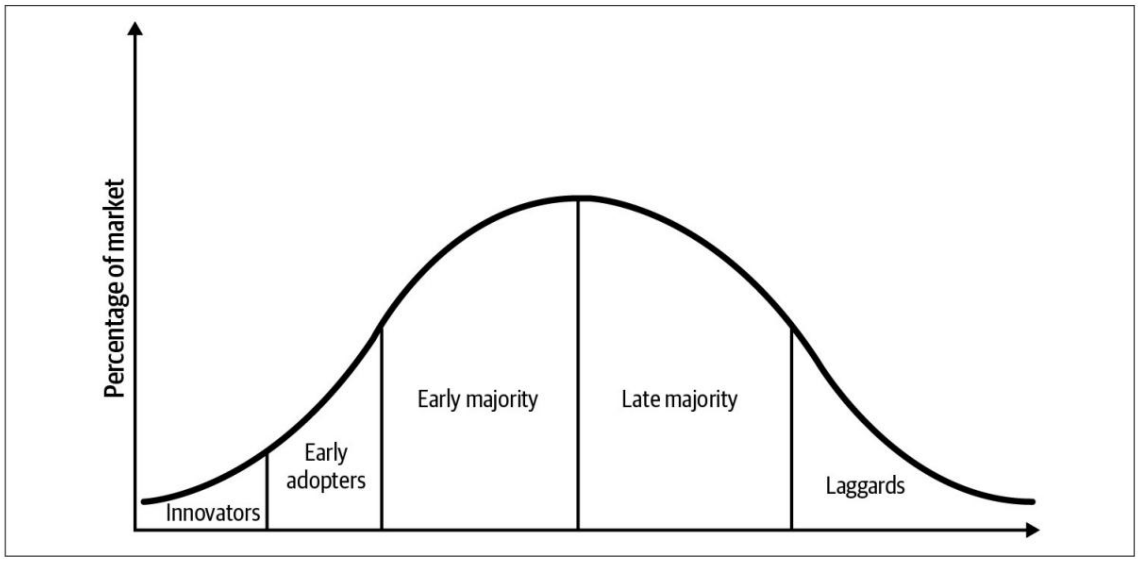

图 11-2。技术采用生命周期,采用或接受新技术的模型。

采用者主要有五类:创新者、早期采用者、早期大众、晚期大众和落后者:

创新者

创新者喜欢尝试新事物,并且乐于冒可能失败的风险。如果您正在考虑在现阶段采用一项技术,您可能无法找到很多资源可以学习:还没有人写过这方面的文章或在会议上发言过。 Stack Overflow 上没有太多答案。您可能会发现相关的工具不存在或不是很成熟。您可能会发现这种特殊的技术从未得到广泛采用:这可能意味着您需要在几年后转向替代技术。

早期采用者 在早期

采用阶段,有足够的信息让人们在尝试新技术时感到相对安全。使用起来可能仍然有点麻烦,但失败的风险较低;有些人已经成功地做到了这一点。

早期大众 当早期大

众研究一项技术时,已经有关于使用该技术的案例研究和现实生活报告。除了新颖和闪亮之外,实际的好处是什么已经变得很清楚了。将提供帮助和潜在的相关工具。

晚期大众 至
此,该技术已经成熟且易于理解。采用它并不是特别危险,而且它可能感觉像是标准做法。

落后者 当
最后一批人采用一项技术时,创新者很可能已经继续前进。它的风险低,潜在回报也低:如果它为您的竞争对手解决了问题,那么您的竞争对手可能已经采用了该技术。

为了用一个例子来说明这一点,让我们考虑一下容器化。当我在《金融时报》的团队于 2016 年开始使用容器时,该技术尚未做好生产准备。我们是集群编排的早期采用者,我们是创新者:我们构建了自己的集群编排。后来,Kubernetes 成熟了,我们看到创新者成功地使用它,我们也成为了它的早期采用者。

我想介绍的第二种思考技术生命周期的方法是 Simon Wardley 开发的,他根据确定性水平和普遍性水平绘制了技术(见图11-3) 。

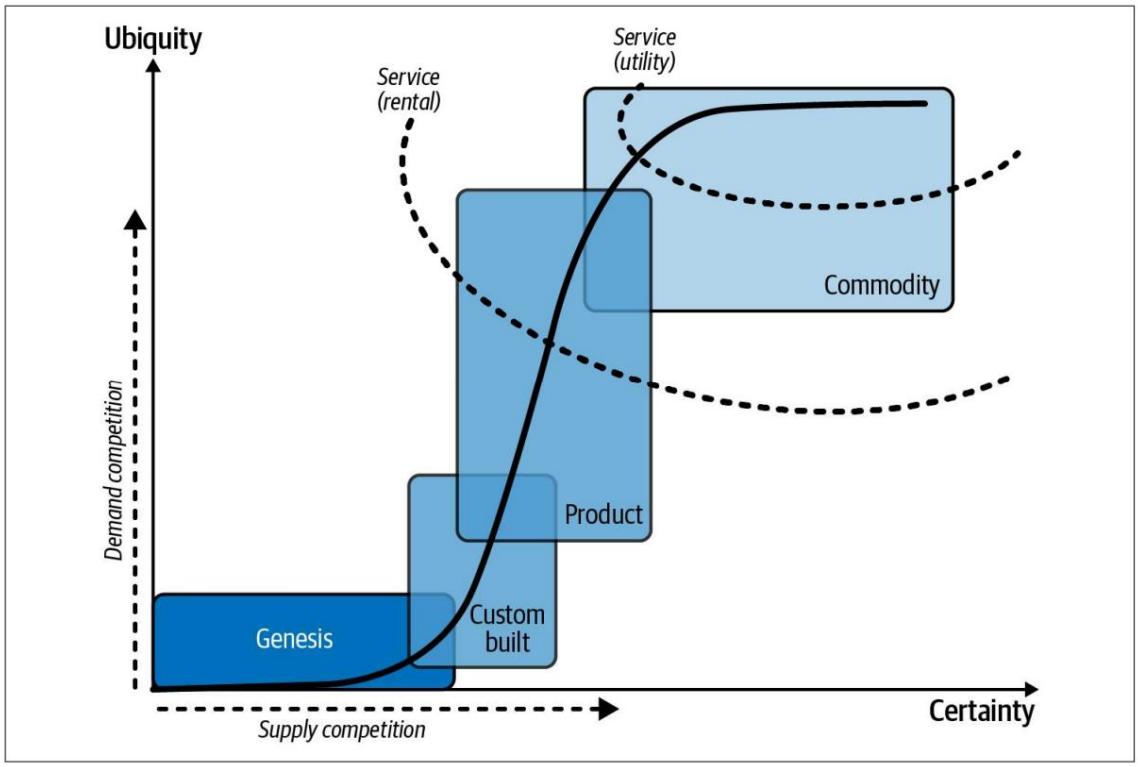

图 11-3。西蒙·沃德利(Simon Wardley)的技术成熟度曲线,一开始是需要定制的东西,如果成功的话,可以成为一种商品。

当一项技术首次出现时,它是罕见的,并且它是否能发挥作用还不确定。
如果成功了,它会逐渐看起来是一个越来越好的赌注 确定性会增加;更多的人会使用它 它的普遍性将会增加。8

这条曲线有四个阶段 起源、定制、产品和商品:

创世纪

这项技术新颖且罕见,但令人兴奋。人们写下了它的潜力。

定制

人们开始使用这项技术并了解它。人们写了如何使用它。

产品技术

应用更加广泛。人们写了如何操作和维护它。

商品 技术无处

不在;用例已被理解。人们写下可以在其之上构建什么。

这清楚地映射到图 11-2 中的采用生命周期:一项新技术开始时是专家(创新者)的东西,你必须自己构建它。然后,随着它的成熟,它将被越来越多的人采用,变得无处不在。有趣的补充是,随着一项技术得到更多人更好的理解和使用,它很可能变成一种可以租用或购买的产品,然后最终成为一种商品。9这似乎很直观 还有一个例子将围绕计算资源。

第一台计算机是定制的,然后您可以购买预制的计算机。现在,您可以租用计算机(云),甚至只租用计算周期(无服务器)。

回到我关于容器化的示例:最初,我们定制了自己的集群编排。随着时间的推移,人们创建了一个我们可以安装和使用的产品 Kubernetes。现在,FT 的容器运行在 EKS 上,EKS 是 AWS 的托管 Kubernetes,正在向商品化迈进。

在这两条生命周期曲线上,围绕特定技术所处的阶段需要考虑一些因素(图11-2和11-3)。

对于不成熟的技术,你必须自己构建一些东西,你会承担很高的风险 如果它不起作用,你可能不得不转向不同的解决方案 但如果成功,因为你的竞争对手可能

8参见 Simon Wardley 的解释, “关于绘图和进化轴”。

9当然如果成功了!

尚未从这项技术中受益。然而,这对你是否有意义取决于你的团队或组织文化:人们是否愿意解决所有问题,并且事情可能会出错?

另一方面,如果某种东西可以作为商品使用,那么采用它就不会有太大的风险。可能也不会带来巨大的回报,因为许多其他人已经在使用它了。这是赌注,通常如果你可以购买某种商品作为商品,你就应该这样做。然而,您仍然可以选择构建一个更好的替代方案,特别是当您意识到替代方案最终会得到不同供应商工具的大杂烩,而这些工具几乎但不能完全满足您的需求时。 10

如果您必须对其进行大量定制才能满足您的需求,您还应该注意不要购买现成的产品。

当然,这还取决于有哪些替代方案,即您可以有哪些替代选择来发展您感兴趣的能力。

然而,我认为这两个模型有助于为接下来的几个部分奠定基础,我想在其中讨论将创新保留在对您的业务最重要的事情上,并且无聊的技术是好的。

为关键业务成果保留创新通常,您应该只在有可用产品时构

建某些产品(如果这是您公司的业务差异化因素),即构建更好的实施将是您业务的核心。

这就是 Simon Wardley 的另一种图表(Wardley 地图)可以派上用场的地方。 Wardley Maps 从连接用户、需求和能力的价值链开始。

首先查看您为客户提供的可见价值。你们公司为他们做了哪些事情?在英国《金融时报》,这将通过印刷报纸和网站提供新闻。这些东西都是建立在其他技术之上的。

例如,该网站运行在一个使用计算的平台上,该平台依赖于电力。请参阅图 11-4了解报纸的简化价值链。

10为 Suhail Patel 的这一点及其精彩的措辞干杯。

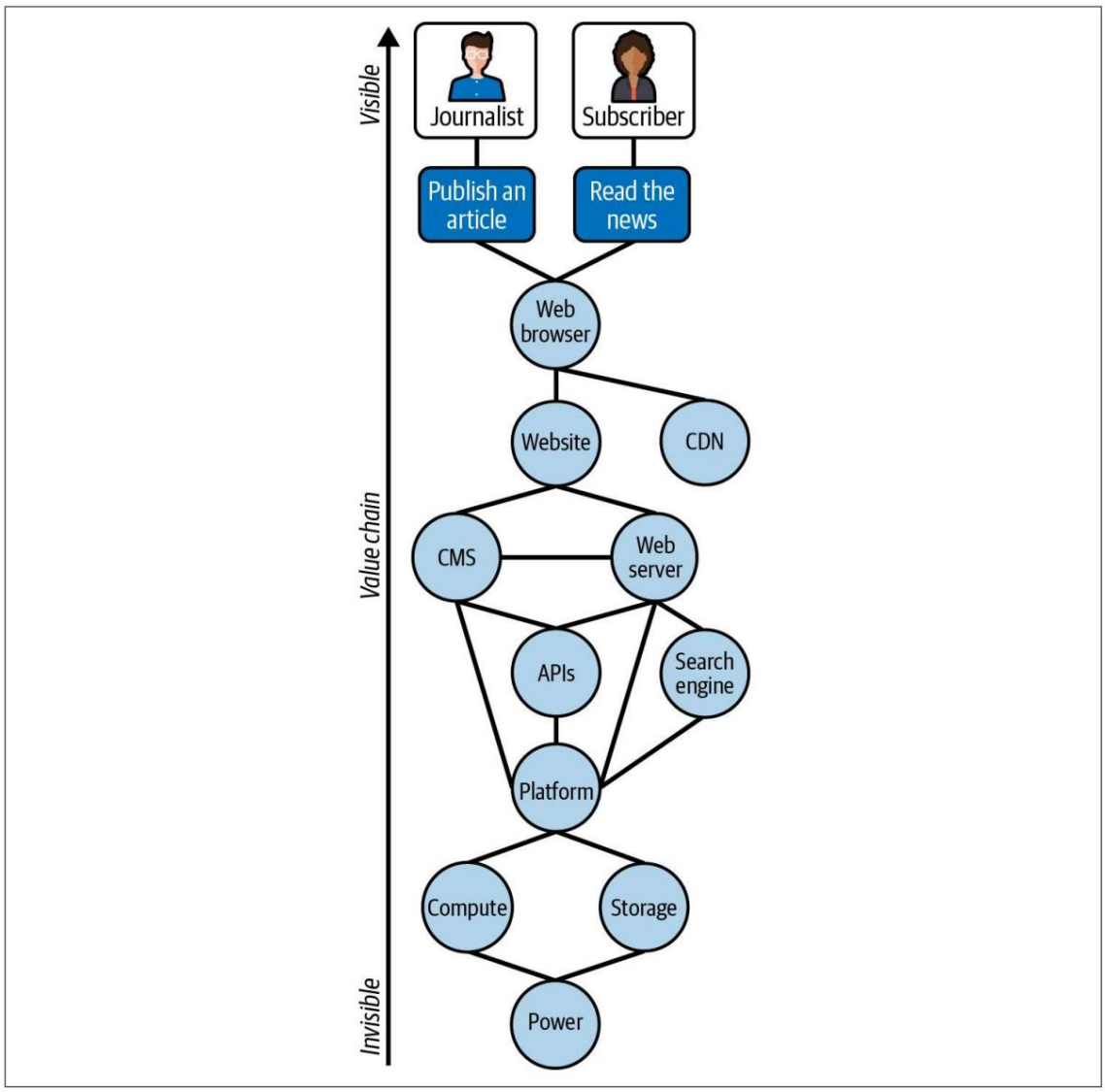

图 11-4。报纸的简化价值链。

您应该将创新重点放在对客户来说最明显、最重要的事情上。

这意味着您可能不应该构建不同的方式来进行持续集成，因为这个问题有许多您可以租用或购买的解决方案。你可能会构建出比你的用例好 5% 的东西，但你还必须维护它。将时间投入到对您的业务具有差异化优势的事情上可能是更好的选择。您可能无法为此购买或租用解决方案！

当你将价值链与上一节讨论的技术演进周期结合起来时，你会得到一张沃德利图，它会变得更加有趣（见图11-5）。

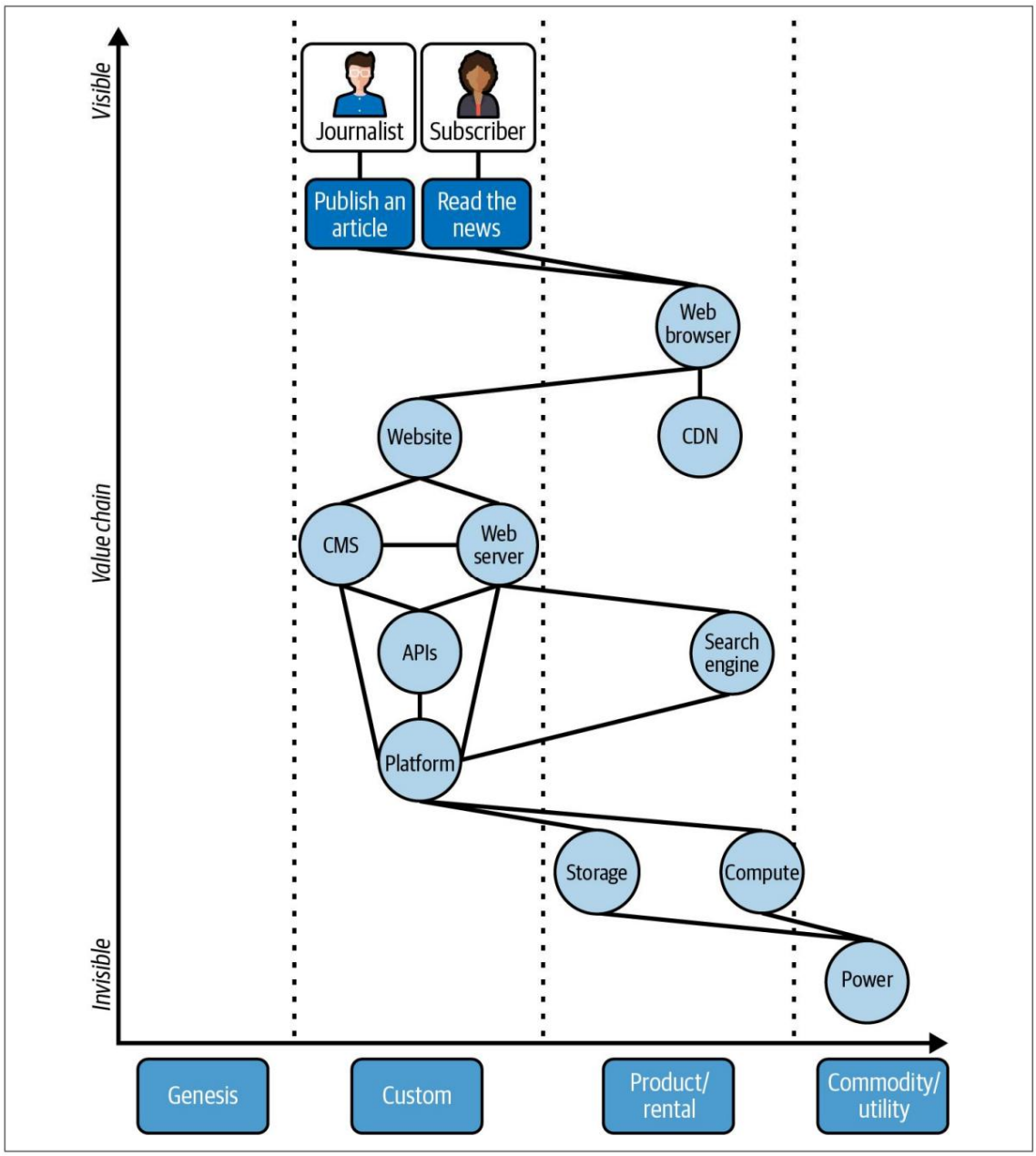

图 11-5。报纸的简化沃德利地图。根据技术演进周期绘制价值链。

价值链地图基础上的东西很可能是一种商品。

中间的东西可能是一种产品　你可以而且应该从货架上购买它。

顶部和左侧的内容是您应该构建的内容;这是你应该集中创新的地方,特别是当它提供竞争优势时。

但值得注意的是,事物也会发生变化。定制的东西一旦达到一定的受欢迎程度就可以作为产品使用。然后,它们就可以成为商品。
回到计算的例子,我们过去常常购买服务器并将其安装在机架上,然后为公共云中的服务器付费。现在,我们使用无服务器选项运行代码:计算是一种商品。

使用钻孔技术

> 无聊的好处是这些东西的功能可以被很好地理解。
> 但更重要的是,它们的故障模式是众所周知的。
>
> 丹·麦金利11

当你自己构建一些东西时,你无法轻易获得其他人的帮助。 Stack Overflow 上不会有任何博客文章或答案告诉您如何解决特定问题。

当没有可用的产品或商品选择时,您自己构建一些高风险、高回报的东西可能是值得的。如果你做得好,你就会领先于你的竞争对手。

既然已经有产品甚至商品解决方案可用,那么构建自己的集群编排(选择我的团队实际所做的事情)的潜在回报还不够高,不足以成为一个好的决定。当集群编排无法作为产品或商品提供时,采用容器可能不是正确的决定:虽然我在《金融时报》的团队在运行微服务方面节省了资金,但除了我们自己的集群编排之外,我们还可以构建什么?

总是需要考虑机会成本。

同样,如果您采用市场上新的产品,您可以获得优势,但这会带来额外的风险。新的创新技术以新的创新方式失败。这些功能不一定被很好地理解,故障模式也不一定被很好地理解。

我认为工程师在研究一项新的、有趣的技术时往往不会充分考虑风险,或者更确切地说,他们不会考虑长期风险。我们在生产中支持系统的时间通常比构建系统的时间要长得多。如果您选择尖端数据库,您对构建该数据库的公司在三年后仍具有偿付能力的信心有多大?公司不会被收购并且数据库不会被关闭吗?如果发生这种情况,您有计划吗?

11请参阅“选择镗孔技术”。还可在Boring Technology Club 以幻灯片 + 演讲者笔记的形式提供,以及我建议两本都读。

在微服务架构中,迁移到新数据库并不是什么问题。远不是一个大爆炸的举动。因此,您承担这个风险可能绝对没问题,但您至少应该权衡一下。

其他需要考虑的因素是成本和安全性。初创公司现在可能会给你折扣,但三年后价格会急剧上涨。他们也可能不具备与老牌公司相同水平的安全成熟度。

正如 Dan McKinley 所说,选择镗削技术有很多话要说。无聊并不意味着不好,它意味着你了解失败模式的事情。你可以谷歌寻找答案。

丹指出,在任何时候你可以做的新的和创新的事情的数量是有限的,并且仍然能够取得成功。你应该像你拥有有限数量的“创新代币”一样思考它。你打算把它们花在什么地方?

一种全新的数据存储:一种创新代币。新的编程语言:创新令牌。

如果您将其花在编写集群编排上,那么您可能不会将其花在提供新业务功能的东西上。这是一个大胆的选择。总的来说,我认为这对我在英国《金融时报》的团队来说是合理的,但这绝对意味着我们在相当长的一段时间内专注于技术而不是产品。

限制替代方案

一般来说,您应该尽可能使用现有的技术,但在大多数情况下,您也应该选择使用公司已有的技术。

这是因为向庄园添加技术是有代价的。你必须理解它,你必须支持它,你必须在生产中操作它。您在招聘时必须保持对您所在地产中所有技术的了解,并培训四处走动的人员。

例如,向堆栈中添加一种更适合特定问题空间的新编程语言的边际效益几乎肯定会被额外的复杂性所抵消。其他编程语言意味着创建库的团队必须添加一种新的语言,而且是一种他们可能不知道的语言。

我确实想指出,这并不意味着永远坚持使用相同的编程语言:你应该改变,但你应该尽量控制在任何时候使用的语言数量。

仅仅因为某些东西正在使用并不意味着它有任何好处。您不希望团队采用一种正在使用但实际上没有人热衷的技术。

另一个例子:让我们考虑一个迁移到不同 CI/CD 管道的团队 - 例如,从 GitHub Actions 迁移到 AWS CodeDeploy。该团队可能会错过添加到集中支持的管道中的任何新功能,例如 DORA 指标跟踪。 12

权衡是 TGG 流程可以增加价值的地方:您可以计算出已经在使用的内容,并仔细考虑以新方式越野的特定用例的收益和成本。

再次引用 Dan McKinley 的话:“添加技术很容易,使用它很难。”13您必须考虑安装和部署、升级、扩展、备份和恢复、日志记录、指标、工程师培训等。

在公共云中并使用无服务器技术,稍微改变了方程式:如果您不必管理数据库,那么需要理解的新东西就会减少。不过,添加新技术仍然是一项开销!

进一步评论。如果您决定添加新内容,您应该考虑一下可以删除哪些内容。

当一个团队采用一项重复现有技术的新技术时,有三种选择。您可以让该团队无限期地支持新技术,作为本地优化。我可以告诉你,几年后,现在团队中的人可能对此不再那么热情了。

其次,您可以看到它的进展情况,如果成功,请将其移交给平台团队拥有。这可能会增加该团队的认知负担(因为您很少会雇用新团队),但这可能没问题。

第三,您可以确定这是一个更好的选择,将其移至该平台团队所有,并将其他所有人也迁移到该平台。如果更好,那可能还可以,但通常会有长尾;由于有其他任务而尚未腾出时间进行迁移的团队。需要领导层的支持才能确保迁移最终完成。

明确哪些地方可以接受重复总会有一些地方确实不可以进行重复。例

如,您可能认为引入额外的编程语言、云提供商或监控系统成本太高、风险太大或太复杂。您可以移动整个组织,但您不希望一个或多个团队选择替代方案。

12感谢 Joe Wardell 提供的示例!

13请参阅Boring Technology Club 幻灯片中的幻灯片 57 。

在护栏或平台文档中详细说明这些内容是个好主意。

期待事情会发生变化你应该期待事情

会发生变化。选择成熟的技术可以让您处于比选择全新的实验性技术更好的位置,但我们并不都还在编写 COBOL。

这意味着你应该尝试做出以后很容易改变的决定。

微服务可以帮助响应变化,因为在系统的小范围内尝试新技术并随着时间的推移迁移其他服务更容易。

您将无法标准化一次并在多年内保持精确的堆栈。
你的目标应该是让人们了解当前的堆栈是什么,并使迁移到新的东西成为一个简单的过程,让人们得到很多关注和帮助。

要了解当前批准的堆栈是什么,我发现技术雷达(在第 6章中介绍)非常有用。技术雷达列出了技术,并提供是否评估、试用、采用或保留该技术的建议。 Thoughtworks的原始技术雷达每季度更新一次,是了解您应该考虑采用或放弃的技术的好方法。

洞察导致行动

当您知道您的财产中有什么并且您已经实现了护栏的自动化时,您就拥有了一个强大的工具,可以深入了解您的软件的当前状态
财产。

这使您可以指导团队下一步该做什么。我已经提到过 SOS 系统,它会查看操作手册信息并告诉团队他们应该首先通过评分系统填写什么。

在英国《金融时报》,我们还构建了一些工具来收集有关已知安全问题的信息:同样,优先显示要首先修复的问题。

当您需要回应问题时,对您的遗产的深入了解也会有所帮助。例如,主题访问请求意味着您需要找到存储个人数据的所有系统:英国《金融时报》在 Biz Ops 中标记了这一点。

有人需要对 Log4Shell（在第 9章中讨论）等问题做出回应，而整个行业的见解确实可以实现这一点。这是治理的关键部分：团队可能是自治的，但您的法律和合规团队以及您的技术领导层希望知道，当出现需要响应的关键问题时，有人正在检查每个团队。

对可视化的投资使得回答诸如“我们有多少服务依赖于这个易受攻击的库？”之类的问题变得更加容易。或“需要付出多少努力才能摆脱这个价格刚刚翻了三倍的工具？”

其他组织的治理

本章非常关注《金融时报》，我意识到并非每个组织都是一样的。所以我想谈谈其他一些组织以及他们如何处理治理问题。感谢 Suhail Patel 和 Stuart Davidson 花时间阅读本章，并向我提供他们关于治理在其组织中如何运作的见解。

Monzo 的治理

Monzo 是英国的一家挑战者银行，以在严格监管的行业中运行数千个微服务并频繁发布代码而闻名。截至 2023 年，产品和技术团队拥有 300 多名员工。该公司已成立八年，正在扩大规模。

Monzo 的理念是风险和治理是每个人的责任，每年的强制性培训提醒人们这一点。

从形式上看，Monzo 拥有与大多数其他金融机构一致的标准第一/第二/第三道治理线。它将第一道防线嵌入特定领域的团队组中，并且第一道防线中的工程师和人员之间存在直接沟通。这些一线人员实际上是管理风险管理/登记册的集体合作伙伴，代表参加风险委员会的审查和会议。更一般地说，他们为工程师提供日常指导。

嵌入式模型运行良好，这些治理专家在很大程度上是团队的一部分，直接参与团队渠道和通话，并帮助制定复杂的决策。这种可见性使其比纸质表格和/或偶尔的奇怪电话更加人性化。

Monzo 在全公司范围内定义了有关变更管理、第三方供应商、打破玻璃、14事件管理等的政策。所有这些都是用简单的英语写成的，最多两到三页，并且

14有关详细信息，请参阅“我们如何保护 Monzo 银行平台的安全” 。

年度强制性培训的一部分。这让人们对政策记忆犹新,而每年的审查意味着人们知道现有政策是否发生了变化。

Monzo 在工具上投入了大量资金来强制执行其策略,特别是在代码审查、部署和进入生产环境的更改方面。该公司允许工程师和团队有很大的自由来决定变革的重要性以及班组内的自治。

与《金融时报》一样,Monzo 的服务等级反映了服务的重要性,而特定服务的自动化控制水平(例如多方批准和变更授权)取决于该服务是否会影响重要的业务服务。越接近关键能力,限制和自动化控制就越多。

目标是采取基于风险的方法,几乎没有全面的限制。要将新的微服务投入生产或对现有微服务进行更改,通常只需要拥有该服务的团队中的同行批准。

Monzo 拥有一种基于变革提案的文化。工程师定义变更的范围和上下文,并且有一个用于横切变更的架构审查流程,允许其他工程师的输入。这是为了确保在做出难以或不可能逆转的变更之前进行深入审查,而不是召开变更签字会议。

对于重大技术变革,例如迁移到新的数据存储或从自托管 Kubernetes 迁移到 Amazon 的 EKS,有一个正式的重大技术变革流程。组织内的每个小组都会收到通知,并且通常会与监管机构进行对话,让他们了解即将发生的变化。

Monzo 内置自动护栏以避免风险。这些包括:

- 单个员工无法在其他人看到的情况下部署到生产中改变。
- 很难发布未启用授权的API。
- 部署不使用标准技术堆栈、不通过标准部署管道以及 Monzo 中的代码的服务是很困难的单一仓库。

在 Monzo 的工程设计中,首要理念是让护栏感觉无缝且不妨碍日常工作,要么让即时且客观的工具来处理它,要么根据爆炸半径采取基于风险的方法。

这些工具会发出大量决策审计日志,Monzo 可以检查这些决策并为审计员提供证据。总的来说,Monzo 不想让工程师们放慢他们日常工作的速度,除非这真的会拖垮银行。

Skyscanner 的治理Skyscanner

是一家总部位于英国的全球旅游公司。与 Monzo 和 FT 一样,它也采用基于微服务的架构,并且频繁发布代码。 Skyscanner 在全球拥有 1,200 名员工,并且正在扩大规模。

Skyscanner 只允许通过 CloudFormation 进行 AWS 更改,并且拥有一个名为CFRipper的基于规则的系统,该系统可以自动应用一组针对基础设施更改的规则,并且是任何构建管道中的强制性步骤。

还有一个用于注册服务的元数据服务,如果存储库中没有元数据或在元数据服务中找不到该服务,CFRipper 会阻止部署。

Skyscanner 使用 CloudZero 进行成本管理,不仅会带来服务的 AWS 成本,还会通过 NewRelic 和任何 Databricks 工作负载带来相关的可观测性成本。每个团队都通过元数据服务与其拥有的服务相关联,因此 Skyscanner 可以确定每个小队、每个服务和关联域的成本。该公司发现,向团队提供信息通常会推动正确的行为。

长期以来,Skyscanner 认为容器足以让许多不同的技术进入业务,但对于每种不同的语言都有相关的成本支持、不同的技术方法、项目之间的人员调动等。

此后,它开始整合一系列特定技术以及供应商提供的技术,并在更具战略性的基础上与他们合作,看看每个关键供应商还能带来哪些其他好处。问题是,是否存在在一系列事情上都“足够好”的供应商,而不是拥有许多最佳供应商?

Skyscanner 使用了 Change Tokens 的概念(即 Dan McKinley 在他的“Use Boring Technology”博客文章中谈到的创新代币),无论是在关键的新业务能力方面,还是在任何一个地方正在发生多少并行工作流方面。推动技术变革的时间:这是为了限制系统中的流失量。技术投资和创新是关键,但如果每个人都同时这样做,这可能会很困难,而且会面临风险。

Skyscanner 拥有生产标准,最初有点像清单,但现在将其重新制作为真/假声明,以便工程师可以更轻松地评估其代码的适用性。

Skyscanner 和我一样发现,“工程师喜欢挑战”,而且通常会以新技术的形式出现,但如果你改变叙述方式并深入研究现有技术,就可以推动一种最佳实践文化。已经,它可以非常强大。 Skyscanner 很幸运,拥有几位研究其所使用技术的世界专家(例如,Guy Templeton 讨论 Kubernetes、Dan Gomez Blanco 讨论 Kubernetes)

OpenTelemetry 和 TestContainers 的创始人 Richard North），因此，专注于这些技术对业务更有效，并且仍然可以通过掌握某个主题的感觉来激励工程师。

如果你遇到困难该怎么办

这里有两个主要挑战。首先，挑战在于让人们相信治理值得投资，而且治理并不是要阻止人们创新，也不是要坚持一套僵化的标准，让每个人都放慢脚步。

其次，解决软件资产的当前状态，以降低风险并遏制蔓延。

对于第一个挑战，首先要讨论风险。寻找特别危险的事情，看看是否可以引入自动化或最少量的流程来保护人们免受其影响。例如，确保凭证不会因签入源代码管理而泄露。一步一步来，解释好处。这包括工程以外的人员，因为这通常是快速完成工作的压力来源，这意味着团队会走捷径并引入风险。

与支持微服务架构良好发展的许多变化一样，如果您的领导团队了解其中的好处并支持您，您将取得更大的进步。
花时间向该团队推销轻量级治理的好处。

如果您已经拥有一堆不同的技术，您可以做些什么来改进呢？

首先要防止情况变得更糟。建立一个论坛，让您可以讨论技术选择。记录讨论和做出的决定。

将您的护栏放在一个清单中，并要求任何新服务遵守这些护栏才能投入生产。

然后投资于了解外面的情况并更好地了解当前状态的方法。
专注于最关键的服务以及包含最敏感和个人信息的服务。设定定期目标以使事情变得更好。这就是自动化真正可以提供帮助的地方，因为您提供了衡量标准和目标。例如，我们的系统可操作性得分之所以有效，是因为团队可以制定指导方针，将得分提高 25%。

您可以阻止情况变得更糟，但最终，您可能必须做出一些艰难的决定，其中涉及团队必须迁移部分堆栈。旨在给予较长的通知期，并提供工具和人员来帮助完成该过程。

总之

在第 7 章中,我们讨论了铺装道路的好处:可以提供公共服务,但如果绝对必要,还有一条非道路路线。

为了使其发挥作用,您需要做好几件事。首先,您需要一组护栏来定义交付生产就绪服务的含义。这应该包括所有考虑因素,例如采购、架构审查、安全和隐私、可访问性、可观察性、更改日志记录、运行状况检查和操作手册等操作信息,以及详细说明上线或退役时需要发生什么的清单。服务。

如果团队想要为特定服务做出不同的技术选择,他们需要遵守这些护栏。有时,这会排除其他选择 例如,也许没有选择不同日志聚合或监控解决方案的选项,因为当你无法在一个地方看到所有内容时,你会损失太多。

您需要的第二件事是就技术变革达成一致的流程,无论是铺好的道路还是选择新事物的自主团队。基于文档的轻量级审查流程意味着会咨询合适的人,人们不会意外地构建已经存在的东西,并且您有机会在几年后回顾并理解所做出的决定。

最后,您需要就引入新内容的意义达成一致。一般来说,这适用于您无法购买的东西,或者这对您的业务至关重要,以至于您可以从定制解决方案中受益。

第12章

增强韧性

当请求通过网络时,分布式系统意味着额外的延迟和更高的失败机会。如果超时和重试错误,缓慢的服务可能比损坏的服务更糟糕,因为线程会被占用以等待其响应。一旦服务恢复,挑战还没有结束,因为大量的请求可能会让服务崩溃。

我们需要以不同的方式构建基于微服务的系统。这些服务应该被编写来处理它们所依赖的问题,包括关闭它们正在运行的主机。

系统应该能够抵御故障,并具有内置冗余。重试、恢复和补救应该尽可能自动化且优雅。仅当您确保单个服务出现问题时系统的其余部分可以工作时,微服务承诺的故障影响范围较小才适用。

在本章后面,我将讨论如何构建弹性服务,然后是弹性系统。首先,让我们讨论一下弹性的含义,特别是构建弹性分布式系统面临的挑战是什么。

什么是韧性?

简单地说,韧性是承受困难或从困难中快速恢复的能力。

任何生产系统都会出现问题。即使系统的某些部分承受压力或停止工作,有弹性的软件系统也将继续提供可接受的服务水平。它还将快速恢复并且不会丢失任何关键数据。

我们将在本章中讨论的许多有助于增强弹性的事情也会使您的系统变得更难以理解且运营成本更高。您必须决定需要建立多少弹性,这取决于您可接受的服务水平。你能在降级状态下运行一段时间吗?哪些功能是绝对必要的,哪些是必备的?

这些并不是纯粹的技术决策;而是纯粹的技术决策。它们取决于对您的业务以及可能跨越小型自治团队之间界限的端到端流程的深入了解。构建弹性系统的大部分工作都发生在微服务级别,但不是全部:您还需要做出更高级别的决策。哪些能力至关重要?要处理的特定请求的可接受持续时间是多少?

您准备支付多少层冗余费用?

其中一些决策可以由团队中的高级工程师和产品人员在团队级别做出。其他决定需要与高层领导和利益相关者进行讨论。

区域、可用区和弹性

我在设计弹性系统方面的大部分经验都是在 AWS 上运行的。您将在本章中找到以下术语,因此我想在这里为那些没有相同经验的人介绍一些关键的 AWS 概念。我的目的是提供足够的背景信息,以便您可以将我正在谈论的内容映射到其他云提供商中的类似概念。

区域

AWS 区域是位于世界各地不同物理位置的数据中心集群。区域之间完全独立。

可用区 每个 AWS

区域至少有三个隔离且物理上独立的可用区 (AZ):离散的数据中心,但通过高带宽、低延迟网络互连。

AWS 建议跨可用区对应用程序进行分区。对于最关键的系统,您可能还希望在多个区域运行服务。多区域服务的运行成本更高,而且运行起来也更复杂,因为这些区域实际上是相当独立的。例如,支持区域之间数据复制的多区域数据库相对较少。如果您不使用这些,则必须自己解决这个问题。

分布式系统的弹性如果您习惯在整体系

统中工作,其中代码进行的调用是在进程中的,那么您可能没有意识到迁移到调用通过网络进行的分布式模型的全部含义。

许多年前,L Peter Deutsch、James Gosling 和 Sun Microservices 的其他人记录了分布式计算的八个谬误,这是刚接触分布式计算的程序员通常会做出的错误假设。这些谬论在今天仍然非常重要,并继续困扰着开发人员。他们是:

1、网络可靠。

2. 延迟为零。

3. 带宽是无限的。

4.网络安全。

5. 拓扑不变。

6、管理员一名。

7、运输成本为零。

8.网络是同构的。

这些是思考弹性和一般系统设计的有用框架。[1]我想简要讨论一下它们,它们可能使您陷入困境的方式,以及您可以采取哪些措施来保护自己。在本章后面,我将扩展本次演练中提到的弹性技术。

谬误一:网络可靠您通过网络进行的任

何呼叫都可能会失败。交换机可能会损坏,网络可能配置错误,或者您调用的服务可能由于某种原因没有响应。你需要防御性地编码。

为您拨打的任何电话设置超时,这样您就不会无限期地等待响应。
如果有意义,请重试该调用,但不要重试,除非您的请求是幂等的(缺乏响应并不意味着请求未得到处理)。

如果无法获得响应,请进行处理:优雅地失败,并返回有意义的错误信息。

1有关更详细的讨论,请参阅Arnon Rotem-Gal-Oz 所著的“分布式计算的谬误解释”。

不仅仅是代码中的调用会受到网络问题的影响。如果您依赖于在某处聚合的日志或指标,则不能假设每个日志或指标都会成功发送,或者它们会很快发送。如果您依赖基于这些日志或指标的警报,这意味着什么?

在系统级别,您应该部署服务,以便即使网络的一部分发生故障,也不会导致所有实例瘫痪。例如,这意味着服务可以在不同的可用区甚至不同的区域运行。对于容器来说,这就是使用 Kubernetes 之类的东西的原因,它允许您指定反关联设置,以便应用程序的副本不会部署在同一台计算机上。

谬误 2:延迟为零延迟

是在网络中的两点之间发送信号所需的时间。光速是这里的限制:没有什么比光速更快了。从欧洲向美国发送消息并返回的时间绝不会少于 30 毫秒。您可以通过确保服务位于同一地理区域,或使用具有多个存在点或 POP 的内容交付网络 (CDN) 来缓存更靠近最终客户端的数据,从而减轻这些地理限制的影响。

但这并没有解决另一个问题,即当您进行多个调用时,网络调用的延迟会增加(这在微服务架构中很常见)。为了缓解这种情况,您应该尽量减少拨打的电话数量,例如一次性请求所需的所有信息,或者至少并行拨打电话。

您还需要测试您的防御代码,以应对您发出的调用返回响应但返回速度缓慢的情况。缓慢可能比下降更糟糕。您最终可能会进行大量处理,但永远不会导致客户看到响应,因为他们已经厌倦了等待并在页面上点击“刷新”。您的重试还可能会增加已经超载的系统的负载。您可以使用超时和断路器来解决这个问题,本章稍后将对此进行介绍。您还可以查看您所做的调用是否必要:如果数据不经常更改,您是否可以缓存一些数据?至少,您可以在出错时使用过时的版本吗?

减少延迟影响的最后一种方法是将系统设计为异步的,在服务之间放置队列,以便服务在将消息放入队列后就完成工作。这对于不需要检索信息的工作流程非常有效,并且消除了处理重试的一些复杂性。一个例子是结账流程,一旦消息进入队列,您就可以返回“订单成功”,并让队列的使用者在下订单或发生任何问题时通知客户。

谬论3:带宽是无限的带宽

是给定路径上数据传输的最大速率。网络上传递的大量数据可能会导致网络拥塞,影响性能并导致数据包丢失（这也可能会花费您大量的数据传输费用,具体取决于您的系统架构）。

设计您的服务以最大程度地减少您传递的数据量。设计界面以便人们能够获得他们需要的信息可以产生很大的影响。

我在英国《金融时报》内容平台团队工作期间的一个例子来自我们对遵守REST最佳实践的渴望。我们有一个用于文章列表的 API,但首次设计仅包含每篇文章的唯一标识符。这意味着任何请求该列表的人都会立即调用 UUID 来请求每篇文章,以获取他们需要在列表上显示的标题、作者和主题。这意味着每个列表页面有 n+1 个请求,其中 n 是列表中的项目数;并检索每篇文章的完整信息,其中大部分立即被丢弃。

将这些关键字段添加到列表响应中可以减少调用次数和传递的数据。然而,它确实带来了如何确保在更新内容时列表中的信息也得到更新的挑战,因为列表和文章更新是分开进入我们的系统的。这是围绕微服务架构中任何类型的缓存的挑战:服务如何知道它们不拥有但引用的数据何时已更新?也许最简单的方法是缓存一段时间,然后返回源以获取最新版本:新版本可用的通知可能会导致不经常更改但实现起来更复杂的内容的流量减少并且更容易出错。

谬误 4:网络是安全的这更

多的是关于安全性而不是弹性,因此考虑到本章的重点,我将保持简短。

您不能假设您的网络是安全的。任何通过网络的呼叫都可能被拦截。这对我们如何构建基于微服务的系统有影响。

通常,人们隐含地信任网络边界内的任何服务。这给你带来的情况实际上与单体应用非常相似,一旦你进入,你就可以访问整个事物。

另一种方法是采用零信任模型,您始终验证调用您的服务的用户是否是他们所说的人,并对传输中和静态的数据进行加密。如果这样做,您将获得更深入的防御,因为获得一项服务访问权限的攻击者无法轻松访问所有其他服务。

谬论5:拓扑不变网络拓扑是网

络中链路和节点的排列。它发生了变化　当然它确实发生了:硬件被替换,新版本意味着微服务实例消失并出现新实例。其中一些变化是永久性的,而另一些则是系统其他部分内置的弹性的结果。

此外,您无法控制其他团队为其拥有的服务器和服务做出的决策。

这就是您需要服务发现的地方:查找服务当前运行位置的方法。虽然使用 DNS 比硬编码 IP 地址更好,但 DNS 记录的生存时间值往往太长,无法在网络拓扑始终变化的环境中工作（例如,使用容器或云提供服务）。 – 西宁。容器编排器（例如 Kubernetes)提供服务发现作为核心服务。

这不仅仅是要找到当前的服务位置,还要应对意外的变化。如果您调用另一个团队拥有的服务并且该团队将这些服务移动到不同的区域,会发生什么情况?如果您对这些服务进行大量调用,您的延迟可能会受到影响。对于您可能调用的第三方服务来说更是如此。

谬论6:只有一名管理员在任何

重要的系统中,都会有不止一个人对网络做出决策。您不知道何时有人要升级您调用的服务,或将其移动到不同的主机或不同的区域。在这些决定已经实施之前,您也可能不会发现这些决定。

为了防止这种情况造成严重破坏,您应该首先确保在组织内定义期望和最佳实践。例如,您的组织可能决定服务必须向服务发现机制注册,并且必须使用相同的系统代码作为标识符将日志、指标和监控发送到中央系统,无论该系统现在是否正在运行完全不同的地方。

作为服务的所有者,您应该了解您所做的更改是否也可能影响其他人。这是 API 的重大变化吗?如果它可能破坏集成,请让人们知道。只要有可能,就可以在不造成任何停机的情况下进行更改,无论是通过逐个实例升级还是通过并行部署新版本并转移人员。或者,设计系统以实现更高程度的解耦　例如,通过使用消息队列。这意味着消费者和生产者根本不需要互相关心。

谬论7:传输成本为零有两

种类型的成本与分布式系统相关:运行分布式系统的成本,以及在两个独立服务之间发送请求的成本。

通过网络需要花钱。设计不当的基于微服务的系统可能需要将大量数据从一个地方发送到另一个地方,而带宽可能会很昂贵。即使是设计良好的产品也可能会产生各种成本,您需要时刻掌控这些成本。

您还需要花费时间来打包然后拆包您发送的每个请求。通过网络的调用比进程内的调用慢得多。如果呼叫的是世界不同地方的服务器,光速会使情况变得更糟。如果延迟对您的用例至关重要,那么微服务可能不适合您。即使不是,您也需要设计流程,以便在不同的云提供商区域之间不会出现同步调用链。

谬误 8:网络是同质的您不能期望

网络上的每台服务器都是相同的。事实上,对于微服务,您应该会看到一些服务是用不同的语言编写的或在不同的硬件上运行的。

这意味着您需要构建可互操作的服务,即它们符合标准并使用相同的数据表示形式,可能通过 HTTP 将其作为 JSON 发送。

微服务的弹性

分布式系统的所有谬误都适用于微服务架构,但拥有大量小型服务意味着两件事。首先是缺点。您绝对会看到瞬态故障始终产生影响　这纯粹是一个数字游戏。这使得构建重试、断路器、舱壁和

更多的。

其次,好处是:精心设计的服务将使系统整体更具弹性。与单体应用不同,您不会同时失去所有功能。您可以优雅地降低部分体验 - 例如,如果您无法访问个性化推荐,则决定显示最常阅读内容的静态列表。

不过,为了做到这一点,您需要了解什么对您的系统和组织有意义。这与技术无关,而是与您的业务有关。

了解您的服务级别要求

构建弹性系统的很大一部分是了解您的系统。您是否有可预测的负载或是否存在意外的峰值?某项特定功能可以停机多长时间而不造成严重影响?优雅的失败会是什么样子?

例如,您是否打开失败或关闭失败?对于像《金融时报》这样的订阅报纸,如果你无法检查某人的订阅状态,打开失败可能是最好的选择:你不会惹恼那些付费订阅的人,也许还惹恼那些能够看到内容的人。免费一段时间就会喜欢它并订阅。对于支付系统,可能失败关闭是一个更安全的选择:您不希望人们获得免费订单![2]

您应该查看系统中的流程,从最重要的开始,并思考什么是重要的。您应该为客户提供什么水平的服务?您需要与业务利益相关者讨论才能真正理解这一点。他们知道如何使用服务以及客户的期望。

服务级别目标正如《站点可靠

性工程》一书的作者指出的那样,“如果不了解哪些行为对于该服务真正重要以及如何衡量和评估这些行为,就不可能正确管理服务,更不用说管理好服务了。”[3]

如果有人告诉您某个特定网页需要“快速”返回结果,这意味着什么?当然,这取决于上下文。但是您应该与您的利益相关者一起找出该特定页面的响应时间开始“太慢”的界限,当这种情况发生时,您可以说您的性能下降了。

首先确定衡量服务水平某些方面的服务水平指标 (SLI)。然后,您与利益相关者就每个 SLI 的目标值或值范围达成一致。这将为您提供服务级别目标 (SLO)。

当您拥有小型自治团队时,设置适当的服务级别目标可能是一项额外的挑战 - 也许没有人了解完整的端到端流程。然而,考虑整个流程很重要:如果不了解流程的其余部分,仅仅因为这是您的团队拥有的部分而尝试为部分流程设置 SLO 并不是特别有意义。

以下是一些有助于设置 SLO 的常用措施。

2此示例来自 Gergely Orosz 的Pragmatic Engineer 时事通讯,2022 年 8 月,他在其中指出,从支付提供商返回的未知错误消息被视为“成功”,导致 2019 年印度的 Uber Eats 订单免费两天!

3 Betsy Beyer 等人,站点可靠性工程(塞巴斯托波尔:O Reilly,2016 年)。

请求持续时间处理

对服务的请求需要多长时间。通常,您希望排除偶尔的异常值,并测量 95% 的请求完成的持续时间（第 95 个百分位,有时缩写为 p95）。例如,您的 SLO 目标可能是 95% 的时间低于 1.5 秒。

错误率

有多少百分比的请求会导致错误?您应该在这里决定“错误”的含义 - 例如,由于用户发送垃圾数据而发生一些不成功的请求。您希望这些被算作错误吗?

例如,对于 HTTP 请求,您可以设置一个目标,即在任何五分钟内来自 5** 状态代码组的响应少于 1%。

系统吞吐量您的服务

每秒处理多少个请求?相关的 SLO 应该不仅仅能够覆盖流量高峰。当我刚加入英国《金融时报》时,我们有支持“雷曼日+ 10%”的想法。当时,《金融时报》的高峰期是雷曼兄弟于 2008 年申请破产的那一天。

我发现有用的一件事是考虑如果在正常流量高峰发生的同时失去两个区域之一会发生什么。这会鼓励我将 SLO 设置为峰值流量的两倍:如果所有流量都进行故障转移,两个区域都需要能够自行处理峰值流量。

可用性您的

服务可用的时间是多少?这可能基于在特定时间段内成功的格式正确的请求的比例,并且通常以“9”的数量来讨论:99.99% 是 4 个 9 的可用性,相当于每月 4.38 分钟的停机时间或每年 52.60 分钟。

当然,在可用性方面,客户（或您的业务利益相关者)可以接受的内容还有很多细微差别（并且业务人员不会以“9”的方式思考）。每年一次停机近一个小时可能比分散的小规模停机更糟糕。或相反亦然!值得了解的是,在某些特定时间,中断是否会产生巨大影响,例如月底的工资系统;或在报纸需要付印前 20 分钟的报纸生产系统。

捕获有关关键时间的信息并使工程师了解这些信息,即使您的 SLO 实际上是根据可用性设置的。它为他们提供了更多背景信息,并允许他们避免在会产生不良影响的情况下进行更改。

英国《金融时报》的另一个例子是确保提醒工程师今天是预算日,或者美国大选即将举行。4

错误预算对于

Google 在网站可靠性方面的工作,我特别喜欢的一件事是错误预算的想法。这个概念是,几乎任何系统都应该有错误的余地,因为 100% 可靠性在大多数情况下是错误的目标。获得最后 0.005% 的成本非常昂贵,而且您的客户无法分辨出差异,因为他们用来访问您网站的内容并非 100% 可用。

如果您能够养成这样的心态:只要达到可用性目标,即使不可用也没关系,这样您就可以就所需的可靠性级别做出决策。例如,如果您正在构建一个内部系统,一次不可用半小时就可以了,您可以在这段时间内移动区域 - 您不需要运行多区域,这将为您节省金钱和成本。打造更简单的架构。

建立有弹性的服务

让我们想象一个非常简单的场景,其中客户调用您的微服务 A,并且作为处理该请求的一部分,A 对服务 B 进行同步 HTTP 调用。

幸福的路径是 B 快速响应,通过“200 OK”响应和一些有效负载,A 提取所需的信息,并向客户发送响应。

但不愉快的道路又如何呢?如果 B 响应“404 未找到”,您可能希望向客户返回适当的错误消息,但如果 B 响应“500 内部服务器错误”怎么办?或者如果 B 根本不回应怎么办?

当您在基于微服务的架构中构建服务时,您正在构建一些不能期望其他服务可用的东西。

在《金融时报》,当我们构建第一个微服务时,我们并没有真正考虑到这一点,因此我们的系统期望能够在启动时连接到数据库或其他服务。这不是一个好主意,因为您开始有一个复杂的图表,其中需要首先启动哪个服务,这意味着您在不需要时将服务耦合在一起。

随着时间的推移,我们学会了如何构建在启动时不希望能够连接到其他服务的服务,并且可以在任何时候处理这些服务的丢失。

4你不应该假设报社的每个工作人员都会关注新闻!

但要做到这一点,您需要了解无法连接到您所依赖的系统的影响。在什么情况下应该重试呼叫?你应该等多久?回答这些问题是设计微服务的重要部分。

接下来,我想谈谈有助于使服务具有弹性的几种模式。您应该确保您构建的服务使用这些模式,因为弹性始于每个服务。您还需要使整个系统具有弹性,我将在接下来的部分中解决这个问题。

冗余构建弹性服务

的第一部分是确保您拥有多个微服务实例,并且这些实例运行在不太可能同时不可用的主机上。

在云中,您希望实例位于不同的可用区中,因为区域的设计目的是最大限度地减少电源、冷却和网络等物理基础设施的共享,并且在一定程度上保证不同可用区中的服务器不会共享数据。同时进行升级。这也适用于容器:如果您在云中运行 Kubernetes 集群,您希望服务器位于不同的可用区,并且希望确保同一容器映像的副本在不同的可用区中运行(您可以这样做通过指定反关联规则)。

为了获得更高的弹性保证,您应该在不同的区域运行。我们的承诺(也许没有那么强烈!)是,失去多个地区的情况应该很少见。

即使您不进行不可变部署,拥有多个微服务实例也支持零停机部署:您一次将新代码部署到一个实例,同时由其他实例处理请求。

一旦您拥有多个服务实例,您就需要一种方法来平衡实例之间的请求。最简单的方法是设置一个负载均衡器,并让实例在启动时向该负载均衡器注册。

快速启动和正常关闭我从 Heroku 的12 因素应用程序概

念中学到了很多东西,这是一种构建 Web 应用程序以在云中有效运行并作为分布式架构的一部分的方法。

这 12 个因素之一是可处置性,它特别适用于构建弹性服务。

您应该将微服务的各个实例视为一次性的。换句话说,它们可以立即启动或停止。

这意味着您应该最大限度地减少启动时间。如果负载增加,这允许快速扩展,并且更容易移动实例,例如移动到不同的主机上。

当您的服务收到 SIGTERM 信号（操作系统发出的终止程序的请求)时,它们也应该正常关闭。对于接收 HTTP 请求的服务,这意味着拒绝任何新请求、完成正在进行的请求并退出。对于从队列消费的服务,您可以将作业返回到队列。

但正常关闭还不够。硬件故障可能意味着您没有时间响应 SIGTERM。这意味着您的服务需要能够从突然关闭中恢复。大多数情况下,这意味着确保操作是幂等的:它们可以多次应用,其结果与应用一次相同。

设置适当的超时我们通常更擅长

处理完全关闭的服务,而不是降级的服务。关于服务是否足够健康以服务流量的问题不再那么模糊。

有助于解决性能下降问题的第一件事是确保为调用设置超时。我发现由于请求未超时而导致事件或使事件变得更糟的情况很常见。通常,库中的默认值相当长,约为 10 秒。你应该显着减少这个。最好是尽快失败。

我惊讶地发现一些 HTTP 客户端库根本没有指定默认超时,这意味着对无响应服务器的调用将永远占用一个线程。

您应该将超时设置为多少?您希望将它们设置为仅在出现问题时才触发超时。在这里,您需要测量正常延迟作为基准。一个好的启发方法是将值设置为 p99 延迟值的几倍,并且通常不高于几秒。

返回 O 和重试一旦设置

了超时,当调用超时或失败时,您会怎么做?

您希望您的服务能够处理连接到其他服务时出现的间歇性问题。例如,在服务部署期间,可能会发生这种情况,其中请求已路由到不再侦听的实例。

最简单的做法是重试请求。然而,这并不总是合适的。您不想重试因请求格式错误或资源不存在而第二次失败的请求。

如果您正确使用 HTTP 状态代码,您通常会重试 5xx 错误,因为这些错误表明服务器无法满足明显有效的请求。

但是,您不想太早重试:您希望下一次调用有可能成功(例如,因为负载均衡器现在已从池中取出已关闭的实例)。

当有大量请求需要重试时,一种幼稚的重试方法(只需等待一段时间然后重试,也许几次)会导致问题。您最终可能会出现负载峰值,这可能会压垮仍在运行的实例。当刚刚再次启动的服务因为收到新请求并重试旧请求而受到大量流量影响时,这也可能很糟糕。

那你该怎么办?下一个改进是应用某种指数退避。因此,例如,您的第一次重试发生在 1 秒后,下一次重试发生在 2 秒后,然后是 4 秒……。

这有助于缓解潜在的负载峰值,但也引发了一个问题。你还要继续努力多久?

当发出原始请求的人厌倦并在页面上点击刷新后,您不想继续重试。你正在做的工作只会被扔掉。

如果您在此过程中将重试内置到多个服务中,则这尤其成问题。

我不建议在同步调用堆栈中进行多次重试。短暂等待后重试应该会被路由到不同的服务。

最后一点评论:将“抖动”添加到您的重试行为中。这意味着在重试之前等待的时间要增加一点随机性,例如,使其在 1 到 2 秒之间。

这样做的原因是重试否则会聚集在一起,从而产生非常尖的负载配置文件,这可能会给您带来问题。[5]

使您的请求幂等在分布式系统中,您不确定请

求是否已在堆栈中的某个位置成功处理。发出的任何请求都可以得到处理,而无需将其返回到调用服务的响应。

5 AWS 博客上 Mark Brooker 写了一篇很好的文章。

这意味着您需要确保您的请求是幂等的,即使这些请求会在某处更改状态。

实现此目的的一种方法是在请求中包含唯一的幂等键。这意味着具有相同幂等性密钥的后续请求将不会被第二次处理。6

保护你自己

您不需要对呼叫您的服务了解太多。您需要做的就是采取措施保护自己。

因此,举例来说,您应该防止有人通过发送过多流量来破坏您的服务。

如果您位于 API 网关后面,则可以通过限制请求数量来做到这一点。根据您使用的 API 网关,它可以在不同的时间范围内应用,因此您可能会限制每小时的请求,但也要考虑突发保护,因此一小时内的请求不会在 1 秒内到达。

另一种方法是减载,当您的服务开始处于负载状态时,您会故意以最少的处理快速失败。例如,您可以统计当前正在处理的请求数量,并在超出限制时返回 HTTP 503(服务不可用)。

测试服务弹性当您在服务中构

建弹性时,您还应该对其进行测试。

人们常常为顺利的路径和失败的情况编写测试,而不是为服务速度缓慢的情况编写测试。在进行负载测试或混沌工程之前,这些测试将捕获处理速度减慢过程中的错误,这两者都将在本章后面讨论。

轻松构建弹性服务作为一个组织,您应该让人

们轻松构建弹性服务。你需要提供脚手架来做到这一点,这应该是铺好的道路的一部分。许多可以提供帮助的工具应该由平台团队拥有。

最基本的选择是为您的编程语言创建库,以确保为每个请求设置超时并支持退避和重试,并使用合理的方法

6 Bart de Waters发表了一篇精彩的博客文章,介绍了 Shopify 如何构建弹性支付系统,其中包括对幂等键的讨论。

默认配置设置。轻松覆盖这些设置并提供文档来帮助人们确定应该使用哪些设置。

除此之外,提高弹性是公司选择 Kubernetes、服务网格和 API 网关等技术的原因之一。编写服务的团队可以专注于业务需求,将应用程序部署到 Kubernetes 集群,服务网格处理集群内的服务路由、安全性、可靠性和流量管理,以及保护对外部访问的 API 网关。面向 API.7

构建弹性系统

微服务架构中的弹性不仅仅涉及单个服务。您还需要在整个系统中建立弹性。

在这里,我将讨论由产品工程团队构建的一个或多个微服务组成的系统,包括构成这些系统一部分的任何第三方系统。在下一节中,我将讨论构建弹性平台。

缓存在设

计弹性系统时考虑缓存非常重要。缓存存储经常读取离消费者更近的数据,从而保护您的站点免受暂时性问题的影响并降低成本。如果大部分请求是从缓存中获得服务并且从不靠近后端,则您不需要那么多后端服务器。

您可能认为对于新闻网站来说缓存是一个坏主意,因为新闻是特定时间的。然而,放置在新闻网站主页上的热门新闻文章将占任何一次发生的阅读量的很大一部分,因此短期(最多几分钟)的缓存会带来巨大的好处。如果您还发送突发新闻警报,这一点尤其重要!

然而,缓存增加了复杂性。如果您将一篇文章缓存在 Content API 中,并且还使用 CDN 缓存更接近消费者的网站页面,那么很难说何时可以认为文章已成功更新。当不同地方的缓存被清除时,人们会看到该文章的不同版本。

如果您严重依赖缓存,那么还存在一个风险,如果您需要完全清除缓存,您的后端系统将会超载。与许多事情一样,您可能应该测试一下!

7有关服务网格和 API 网关的更多信息,请参阅James Gough 等人的Mastering API Architecture 。

处理级联故障我见过许多事件,归

根结底都是“我们的恢复能力让事情变得更糟”。一般来说,这是由于级联故障造成的。例如,一个实例过载,因此负载均衡器停止向该实例发送流量,但现在所有其他实例都获得额外负载,从而增加了它们失败的可能性。

或者,因暂时性错误而重试会导致过载并导致服务完全瘫痪!

您需要对系统进行设置,以便一个实例或一个可用区发生故障时不会导致其他所有系统过载。如果您设计一个具有两个区域的系统,并且具有仅在一个区域运行的故障转移机制(就像《金融时报》所做的那样),您需要能够处理该一个区域中的所有流量。

您应该定期对此进行测试,例如通过混沌工程测试期间的负载测试。您很容易忽视架构的变化或负载的增加,这些变化或负载的增加已经超出了您的处理能力,并且只有进行测试才能真正确定。

您还应该轻松了解负载何时开始导致问题,以便您可以通过扩大规模来做出响应,并确保扩大规模很容易实现或自动发生。

您应该做的另一件事是安装断路器。如果电路过载,电路中的断路器将会失效,从而保护整个系统。

在软件中也是同样的想法,尽管不涉及物理组件:软件“断路器”跟踪调用另一个系统的故障,如果故障数量超过阈值,断路器就会“跳闸”并打开电路,停止进一步的呼叫。

然后,在适当的时间后,断路器将允许呼叫通过以测试下游系统是否恢复运行。如果有效,电路将关闭,呼叫将正常发送。

当然,你必须弄清楚断路器打开时你要做什么。
这是更大讨论的一部分:后备策略(如果有的话)怎样才有意义?

回退行为

设计您的系统很容易,就好像连接其他服务时只会出现间歇性问题,但这不太可能是现实。

如果您的服务无法访问它所依赖的其他服务,会发生什么情况?这是设计系统架构时的关键,并且可能是您的利益相关者和工程团队面临的问题。这通常是一个商业决策。

您的客户期望什么?当事情不太顺利时,你应该向他们返回什么信息?

有后备选项吗?例如,如果您正在调用以根据用户的个人资料检索供用户阅读的建议文章列表,那么在失败时,您可以向他们显示当前最受欢迎的文章,或者不返回任何内容,只需从其中删除该部分呈现的网页。

您还可以做出架构选择,以最大限度地减少服务不可用的影响,例如,将信息缓存在服务中一段时间并使用它。这会减少正常操作期间的负载,并且当缓存项过时时,您可以返回到源,要么接受此请求将花费更长的时间,要么最后一次返回缓存中的内容,同时返回到源以获取最新的内容。日期值。如果下游系统不可用,这还提供了使用过时数据的选项。

当您调用外部系统时（例如与 SaaS 产品集成时）,制定可接受的后备行为也很重要。

如果您的支付网关不可用,您会怎么做?

避免不必要的工作虽然您应该小心重试以避免级联失

败,但它们也可能导致不必要的工作,产生永远不会返回给客户的响应。

当您同步处理请求时,即客户调用一个服务,该服务调用另一个服务等,您需要考虑整个请求流程。如果您在每个服务中实现退避和重试,并且一个调用要经过多个服务,那么您最终可能会做大量工作并花费大量时间。

让我们考虑如图12-1所示的一个小型服务链。客户调用服务 A,服务 A 调用服务 B,服务 B 请求来自数据库 C 的数据（这是一条逻辑链,不显示单个实例或任何负载均衡器或网关）。

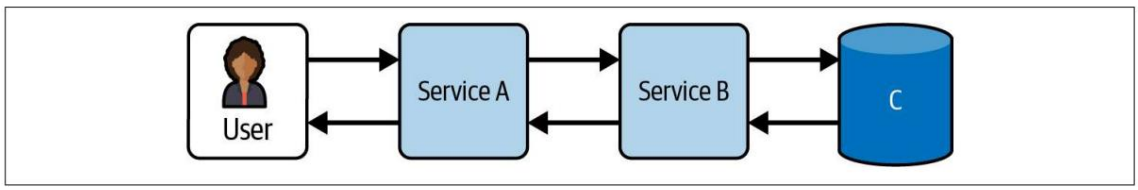

图 12-1。服务调用链示例。

如果 C 出现问题,B 会重试呼叫。但如果 A 也设置为重试呼叫怎么办？ B 在 1 秒后重试,然后,因为它设置为指数退避,所以在 2 秒后重试。然后A等待1秒再次尝试B,整个过程重复进行。

这会比正常响应时间增加 7 秒。客户很可能在此完成之前就点击了刷新。但您不想删除服务 A 中的重试,因为如果存在调用服务 B 的 blip,这可以提供恢复能力。

所以,你可以做什么?

如果调用需要同步,请设置时间预算。发起请求后,在进入堆栈时,设置带有最佳使用日期（时间戳)的请求标头。
每个服务都应该检查,如果最佳食用日期已经过去,则停止处理并快速返回。至少您不会做很多额外的工作,然后启动服务就有责任决定做什么,或者向客户提供什么选项,例如,一条错误消息说“目前不可用” ， 请稍后再试。”

但这种情况表明,同步调用很可能会导致除了最短暂的问题之外的任何错误。如果可以的话,最好将系统设计为异步的。

Go 异步链式同步调用

不是一个好主意。在两个服务之间放置队列可以将它们解耦。生产者或消费者中的任何一个都可以消失并重新出现,而不会影响对方。这意味着消息将保留在队列中并最终被消耗。使用队列重新运行失败的请求也变得更加容易:例如,您可以将它们放入另一个队列中,当潜在问题得到解决时可以处理该队列。生产者不必了解消费者的当前状态。事实上,生产者甚至不一定知道哪些服务消费这些消息,并且可以很容易地添加另一种消费者类型。

例如,当我们在 FT 重建变更 API 时,变更消息经过验证以确保它们采用正确的格式并包含必填字段,然后写入队列。这意味着对 Change API 的调用通常会成功且快速地返回。最初的实现让消费者将更改写入 Slack 通道,并写入处理更改跟踪的第三方系统。当我们停用该系统时,我们可以轻松地开始将更改写入数据存储,而无需更改调用代码或更改 API 的入口点。队列使编写和运行服务变得更加容易。

故障转移

如果您在多个区域中运行,您可能希望找到一种方法将所有流量仅发送到这些区域的某些子集,可以通过某些监控自动发送,也可以手动发送。

您确实需要考虑整个系统拓扑,包括从其他域调用服务。例如,如果您的服务在欧洲运行并且通常调用地理负载平衡的 API 平台,该怎么办?如果该 API 平台因故障转移而仅在美国境外运行,则平均延迟可能会大幅增加。

这会导致不可接受的延迟或超时吗?

此外,如果您想要依赖故障转移方法,则需要每个区域都能够处理您的所有流量。

最后,如果您不实践这种方法,您可能会发现您很难让所有事情都失败并正常工作。在英国《金融时报》,我们定期练习对 API 平台和网站进行故障转移,并让不同的工程师来做这件事,因此当我们真正需要这样做时,我们知道它会起作用,而且每个人都对这个过程感到非常满意。

备份和恢复您绝对应该

确保您能够应对数据问题。这可能是数据库丢失、数据损坏(这可能包括由于某人进行广泛的数据更改而导致的数据损坏)或不一致。

您需要备份数据,并且需要定期测试是否可以从这些备份中进行恢复。否则,您无法确定恢复过程在您需要时是否有效。

在微服务架构中,会存在一定程度的数据重复。了解哪个数据源是数据的规范版本非常重要。这绝对必须得到支持。

可以从此数据源重新填充其他数据。这是否有意义取决于您组织的环境。如果有大量数据需要重新填充,您可能会决定无法在可接受的时间范围内完成此操作,并采取措施备份相关数据存储。

重复的另一个方面是数据的不一致,要么是暂时的(例如,因为一段内容的新版本尚未到达每个数据存储),要么是永久的(例如,该新版本的发布在一个区域失败,但没有失败另一个)。

这意味着要考虑如何了解不一致的情况,并制定修复这些不一致的机制。对于诸如内容发布之类的内容发布(重新发布可以解决问题)来说,编写检查多个数据存储中的数据并进行比较的工具就足够了。事实上,在英国《金融时报》,我们走得更远,因为内容发布是幂等的(可以多次进行,没有副作用),我们编写了重复发布最近内容的工具,只是为了减少内容发布不一致的可能性。结束数据存储。

灾难恢复您应该在
不同的可用区甚至不同的区域中运行单独的服务实例。

不过,即使您位于多个区域,您也应该考虑如果其中一个区域出现故障很长一段时间该怎么办。您能否在新地区复制您的生产系统?你需要多长时间?

您可以使用新帐户进行操作吗?这意味着将数据备份存储在主帐户之外,但它可以在发生勒索软件攻击时挽救您的生命。

构建弹性平台

产品工程团队构建的系统存在于更广泛的环境中:完整的软件资产,包括平台功能,例如 DNS 或 CDN 提供商、源代码控制、构建和部署工具等。

其中一些功能将在内部构建,其他功能将由第三方提供。无论如何,这里也需要弹性,并了解您的后备选项。

对外部问题的抵御能力

现代系统倾向于使用SaaS解决方案,效果良好。为什么要花时间构建对您的业务不重要且其他人已经构建的东西?

除此之外,还有一些类型的系统您绝对不应该自行构建,包括支付平台、身份和访问管理、CDN 和 DNS。
最好让专家构建和运行这些系统。

但这意味着您的系统中有一个关键部分并未运行。当涉及到这些系统时,您应该如何考虑弹性?

您可能认为这些是第三方软件,不属于您的责任,但是当您的客户无法查看您的网站时,他们不知道您的网站是由于您控制范围内的原因还是由于您依赖的供应商而关闭在。他们只是知道您的网站已关闭。

如果您有第三方提供关键功能,您需要了解他们期望提供的服务级别,以及如果他们无法满足该级别会发生什么。

您应该将一些服务级别协议 (SLA) 作为与这些提供商签订的合同的一部分。根据经验,如果系统停机时间意外长,这可能会让您获得一些退款,但您的 SLA 也很可能会受到影响。

仅涵盖供应商确认您有问题之前的时间,而不是他们解决问题之前的时间。重要的是要了解您可以期待什么。例如,提供商是否定期备份您的数据,以便您可以在组织中的某人意外删除您的所有票证或所有源代码存储库时进行恢复?

您也需要做出决定。一个决定是,您是否想通过拥有第二个提供商来建立冗余?这是一个权衡。虽然它可以提供更大的弹性,但拥有第二个 CDN 提供商将使您花费两倍的费用。日常中潜在的更大问题是你必须将所有事情实施两次,这会在每次做出改变时减慢你的速度。您还将很难使用任一供应商的差异化功能,因为如果一个供应商出现故障而您必须切换,则该功能将不可用。您还需要定期测试冗余,否则您可以打赌它在您需要时将无法工作。

这是我在《金融时报》上进行过几次讨论,最近一次是在我在第 8章中提到的 2021 年 Fastly 中断之后。几个月前进行的代码更改允许特定类型的配置更改,以及当客户进行该更改时,它让 Fastly 垮了。这不仅影响了《金融时报》,还影响了互联网上的大量其他网站,包括 Netflix、Reddit 和 Amazon。 8

最终,我们当时的决定是,在权衡影响与第二家 CDN 供应商的额外成本的基础上,我们接受偶尔的中断。事实上,在英国《金融时报》主要网站无法访问的时间里,英国《金融时报》的记者们就转而使用 Twitter 来分享突发新闻。

这并不意味着您不应该为关键软件安排二级提供商。这只是意味着您需要考虑是否准备好支付费用。

内部工具工程支持团

队的部分职责是为产品工程团队构建可靠的工具。至关重要的是,您需要这些工具的可用性能够满足该工具停机时将受到影响的服务的可用性要求。

让我们考虑一下这对于不同类型的工具意味着什么。

部署工具哪些东西

可以让您部署或发布代码?该列表可能包括您的源代码控制软件以及您的构建和部署管道。

8请参阅“大规模互联网中断影响包括亚马逊、gov.uk 和卫报在内的网站”， e Guardian。

这些东西可能至少应该与通过它们部署或发布的服务一样有弹性。

情况并非总是如此 并不总是可能的,特别是当您使用 SaaS 选项时 但至少值得考虑是否可以解决它们宕机的问题。

让我们看几个场景。如果您使用云选项进行源代码控制,如果该选项发生故障会发生什么?随着 GitOps 的兴起,GitHub 或 GitLab 的不可用也可能会阻止您部署代码,如果您将基础设施存储为代码,您可能也无法访问它。您有备用计划吗?

您的流程越依赖这些工具,解决这些问题就越困难。你脱离了练习。

我并不是说不要使用 GitOps,而是制定一个备份计划,并定期测试它。
或者接受源代码控制提供商的中断意味着您无法部署代码,并希望它不会在您自己的事件中发生。

操作工具如果您的操

作工具不可用,您就无法知道您的服务是否按预期运行。

这意味着可观察性、监控和日志记录工具需要具有高度的弹性和可用性。

同样,您在事件期间用于沟通的工具也需要可用。如果您已将事件管理流程大量集成到 Slack 中,就像我们在《金融时报》所做的那样,那么当 Slack 宕机时您会做什么?

您的操作手册是否具有高可用性?当您尝试解决重大事件时,您最不想看到的就是发现无法访问操作手册。

您的备份不必非常复杂。我们为 Slack 进行事件管理的备份是一个 WhatsApp 群组和一个用于跟踪信息的 Google Doc。

对于操作手册,我们知道我们的 Biz Ops 图形数据库的可用性不高;它只在一个地区运行。我们没有努力创建这个多区域,而是定期提取运行手册信息并将其存储在高度可用且持久的 S3 存储桶中。

然而,正如我们发现的,当我们的单点登录系统出现故障时,我们的 S3 文件受到单点登录的保护。我们无法访问操作手册。之后,我们引入了第三级备份,将 Runbook 文件压缩并存储在 Google Drive 上。

您用于事件管理的其他工具怎么样?您有视频通话的备份吗?聊天?如果电子邮件不起作用,您可以与人们联系吗?

最后,这是我们通过惨痛的教训才了解到的,您是否为可观察性工具和您的服务使用相同的顶级 DNS 域?失去顶级域名,你就完全盲目了。

也要考虑数据

如果有人意外或故意删除了源代码控制系统中的所有存储库,会发生什么情况。你有备份吗?

这同样适用于任何类型的基础设施配置。希望您已经转向基础设施即代码,而这一切都在源代码控制中。如果没有的话,如果有人不小心删除了配置,你能恢复吗?

跟踪计划工作的系统中的工单也是如此。如果有人意外地进行了更改,从您的 Jira 实例中全局删除了某个字段,您可以通过恢复数据备份来恢复该字段吗?这些类型的事情往往只有在事情发生之后才会浮现在脑海中,但是提前花时间考虑可能会出现什么问题以及如何恢复可能非常有价值。

验证您的弹性选择

您可能认为您已经在系统中建立了弹性,但您无法知道这一点,除非您看到出现问题时会发生什么,例如系统的某些部分发生故障,或者负载激增比你计划处理的要大。

在生活对你进行测试之前,你应该先测试一下你的弹性选择。
有几个流程可以提供帮助。

混沌工程混沌工程的

理念是,当出现问题时,您应该测试您的系统是否按照您期望的方式工作。微服务架构很复杂,你认为会发生的事情可能实际上不会发生。混沌工程可以让您了解系统如何实际响应特定场景。9首先要澄清的是,这是一个误导性的名称,人们现在通常使用更让人放心的“弹性工程”。这个术语不太可能吓到业务利益相关者和领导层!10

9 Casey Rosenthal 和 Nora Jones 在 Netflix 首创了这一技术,并撰写了《混沌工程》(Sebastopol:O Reilly,2020)。

10虽然混沌工程显然听起来更酷。

混沌工程经常发生在生产中。正如我之前提到的,没有其他环境拥有完整的数据或完全相同的系统集。它不应该导致问题。毕竟,您希望检查您的恢复能力是否有效。

有一些工具可以帮助进行混沌工程,使场景自动化并允许您频繁运行它们,以便您发现影响系统弹性的更改。但是,如果您是混沌工程新手,您可以从小处开始,进行手动更改。我几乎可以保证会有一些令人头疼的时刻,但你会学到一些东西。

拉斯·迈尔斯 (Russ Miles) 在他的《学习混沌》一书中描述了如何为“比赛日”做准备工程:

- 决定参与者是否知道将要发生的事情。对于你的第一次实验,我建议提前通知人们,因为这样压力更小,你仍然会学到很多东西。之后,你可以采用 Russ 所描述的“龙与地下城”风格,人们不知道会发生什么:这会告诉你是否真的能够发现问题所在。

- 决定谁将参加。您可能想故意排除那些总是带头处理事件的人。毕竟,当事情真正发生时,它们可能无法使用。

- 获得批准。确保这不会让人们感到惊讶。例如,这包括检查这是否与您网站的重大营销活动不相符。你不会期望事情会出错,但最好还是小心一点。

一旦你计划好比赛日(当然,不必是一整天),接下来做什么?

正常的是什么样子的?

第一步是了解您的系统的正常情况。如果您不这样做,当出现问题时您如何知道如何解释图表和日志?

确定一个小改变 考虑一个小

改变。也许会关闭服务器,导致流量激增,减慢某些响应。

您应该使其具有较小的爆炸半径,并且您希望系统能够适应变化。不要故意破坏生产中的东西;它会让你非常不受欢迎。

预测将会发生什么 想想你期望

看到什么。警报会起火吗?您预计会对响应时间或错误率产生影响吗?

进行实验看看您的
假设是否正确。系统是否仍按预期运行?你看到你预测的变化了吗?

我发现混沌工程场景是将系统移交给新团队的一种非常有效的方法。当您尝试解决问题时,您会发现很多有关系统的信息,并且您也会更有信心,如果问题真的发生,您可以解决问题。

我们还使用混沌型实验来帮助我们的一线运营支持人员在首次构建新网站和内容发布平台时了解它们。
我们至少进行了一次实验,团队中的一名工程师发现了问题,并在其他人做出回应之前修复了它,这也非常值得一看。

测试备份和恢复如果您有备份并

且从未测试过恢复它,那么您拥有的只是一个文件。也许连这个都不是!

然而,我听说过许多事件,结果表明备份过程已中断数天、数月甚至数年。

您确实需要测试这个过程,并且定期进行测试。

对于充当关键数据真实来源的数据库,您应该跟踪上次测试恢复的时间。我们在服务存储库中执行了此操作,并且我们可以查找未在(比方说)去年测试过恢复的关键系统。

熟能生巧

你做某事的次数越多,你就越有可能在压力下把它做好,并且你越不可能在变化阻止它发挥作用的地方失败。

我们过去经常练习将 ft.com 和内容发布平台的故障转移到单个区域。这意味着如果我们需要作为事件的一部分,任何数量的人都可以这样做,对此我们感到非常放心。

负载测试负载

测试向您展示您的弹性在实践中的表现。它对于了解服务器开始过载时发生的情况特别有用。你最终会遭遇连锁故障吗?

这意味着您需要加载测试组件直到它们损坏,这对于容量规划也非常方便。

您的目标应该是运行逐渐增加负载的测试以及负载尖峰的测试。
您可能会看到完全不同的结果。

真实流量可以提供比合成流量更真实的结果,如果您有能力自动扩展生产环境,您可以从在生产中进行负载测试中获得很多好处,也许可以通过记录真实流量并以更高的速度回放吞吐量。

如果您不愿意在生产中这样做,特别是如果您打算运行负载测试失败,您还可以设置一个临时环境来匹配生产并在那里运行测试。

从事件中吸取教训

每次事件发生后,您应该问自己的一件事是“我们的韧性对我们有帮助吗?”在最好的情况下,由于系统内置的弹性,您最终可能会对未遂事件进行事件审查,但不会对用户产生实际影响。

在其他情况下,您可能会发现事件是由您没有想到的场景引起的。

一次只做一件事关于复

原力的推理很复杂,当您尝试推理多个出错的事情时,情况会变得更加复杂。

一次专注于如何处理一个问题会很有帮助。因此,举例来说,在《金融时报》,我们没有尝试计划同时失去 AWS 和 DNS(显然,如果 DNS 提供商也在 AWS 上,那就不同了)。

这都是风险评估的问题。两个高弹性且独立的系统同时出现问题的可能性有多大?困扰你的事情通常是当你发现他们并不像你想象的那么独立时。然而,在您为可能发生的所有可能的单一事件制定计划之前,不要担心组合问题。

如果你遇到困难该怎么办

一般来说,如果您将所有时间都花在扑救生产事故上,您就会知道自己正在与恢复能力作斗争,所以从这里开始。

确保您利用这些事件来了解您的系统。查看本章中围绕构建弹性服务和系统所涵盖的想法:设置超时、后退和重试、缓存等。确定可以增强您对未来此类事件的弹性的操作并采取这些操作。

最初,您可能会发现许多潜在的改进,但我发现这可能有点让人不知所措。事件发生后,您将它们全部添加到积压工作中,但几个月后,许多都没有取得进展。有两件事可以提供帮助:首先,如果您发生了第二起事件,而该已确定的操作本可以阻止它发生,请同意您将立即采取此操作。其次,承诺对任何事件采取一项商定的行动。

当你不再被这些自然混沌工程实验压垮时,就开始运行你自己的实验吧。保持简单,并测试如果丢失可用区或区域会发生什么情况。

花一些时间考虑您的数据,并确保您定期备份并且知道如何恢复它们。

运行一些负载测试,看看系统在哪里开始出现问题。再次,安排时间采取一些行动。

您可能需要首先讨论如何为此类工作分配团队时间。通常,人们认为产品工作是首要任务,但您的产品需要可用!

如果你需要与你的产品团队进行这样的对话,请引用 eBay 前产品和设计高级副总裁、 《Inspired》作者马蒂·卡根 (Marty Cagan) 的话。
卡根很清楚,产品经理需要腾出时间进行工程工作,否则他们将面临系统无法获得支持的风险。 Cagan 建议,产品管理部门应将所有工程能力的 20% 交给工程部门,让其按照他们认为合适的方式进行支出。 “他们可能会用它来重写、重新架构或重构代码库的有问题的部分,或者更换数据库系统、提高系统性能 无论他们认为有什么必要,以避免必须来到团队中说“我们需要停下来重写。”11

对我有用的一件事是为所有以技术为重点的工作分配一个看板通道,以便产品所有者可以看到该工作与产品功能的平衡。该通道中的任务由工程团队确定优先级,但我们确保解释了执行这些任务的价值。

建立一个有弹性的系统需要时间,但是小的改进会不断积累。

总之

在本章中,我们研究了分布式系统的谬误。网络并不可靠,它一直在变化,而且变化是由不同的人做出的。延迟和带宽限制可能会产生很大的影响。

11参见“工程想要重写”。

我们介绍了服务级别要求以及在介绍一些合理的服务级别指标之前了解您的业务需求的重要性。

接下来,我们介绍了如何通过冗余、快速启动和正常关闭、错误时的退避和重试来构建弹性服务,但需要注意的是请求必须是幂等的。

然后,我们研究了系统级别的可靠性。在负载下的系统中,过多的退避和重试可能会导致一系列故障,因此请设置超时并了解可以在何处回退到不同的行为。

仅仅构建有弹性的产品是不够的。工程师使用的平台和工具还需要具有高可用性和弹性。确保您还考虑允许您发布代码更改并查看生产中发生的情况的工具。

您可能认为您拥有一个有弹性的系统,但这是您应该测试的东西。运行混沌工程实验并练习故障转移和从备份恢复数据。