通过在上面使用混沌工程来提高弹性并改善客户体验 AWS

AWS 规范性指导

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 规范性指导: 通过在上面使用混沌工程来提高弹性并改善客户体验 AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

通过在上面使用混沌工程来提高弹性并改善客户体验 AWS

Laurent Domb，Amazon Web Services 联邦金融首席技术专家

2025 年 4 月（文档历史记录）

混沌工程是一门在应用程序上进行实验的学科，目的是建立对组织和应用程序承受生产中动荡条件的能力的信心。这是一种主动的弹性方法，其目标是通过在人员、流程和技术中引入受控故障，验证您的应用程序和组织是否能够吸收、适应并最终从服务损坏中恢复。其目的还是在漏洞可能导致生产中断或其他中断之前，识别并消除这些漏洞。

在Amazon，我们知道分布式系统中故障是不可避免的，以至于即使存在故障也能正常运行是正常的操作模式。由于服务之间的交互必然会失败，因此您需要了解服务在各种故障模式下的反应，并构建能够抵御依赖性故障、重试风暴、可用区受损和主机资源耗尽等关键漏洞的服务。

让我们以重试风暴为例。客户端中的局部故障可能会严重影响多项服务。这通常被称为蝴蝶效应。重试风暴是蝴蝶效应的表现，在这种效应中，失败的依赖关系会触发客户端以及这些客户端的客户端重试失败的操作，从而导致流量呈指数级增长。服务之所以过载，是因为在处理性能下降的同时，除了重试流量外，它们还必须响应常规流量。

混沌工程已成为对分布式系统日益复杂的回应。它是一种多学科的方法，它结合了混沌理论、系统思维和工程原理，设计和管理能够抵御意外事件和行为的复杂系统。混沌工程的核心是理解和管理复杂系统在不确定性和不可预测性条件下的行为。它认识到，依赖于预测和控制结果的传统工程方法通常不足以应对分布式系统的复杂和动态性质。随着这些系统的发展，它们往往超出了任何一个人的理解范围。

混沌工程提供了概念、技术和工具，可以故意将故障注入系统，以便在弱点出现在生产中之前将其发现。这种积极主动的方法使组织能够建立信心，相信他们的系统将在压力很大的条件下运行。尽管混沌工程仍然是一种不断演变的实践，但它代表了向设计、管理和操作现代计算系统的根本转变，以便在复杂性和互连性不断增加的情况下保持弹性。

本指南的以下各节讨论了混沌工程的好处，解释了如何进行混沌工程实验，并描述了在组织中大规模实施混沌工程时可以采用的方法。还包括实验计划样本和实验结果文档，可用作混沌工程实验的模板。

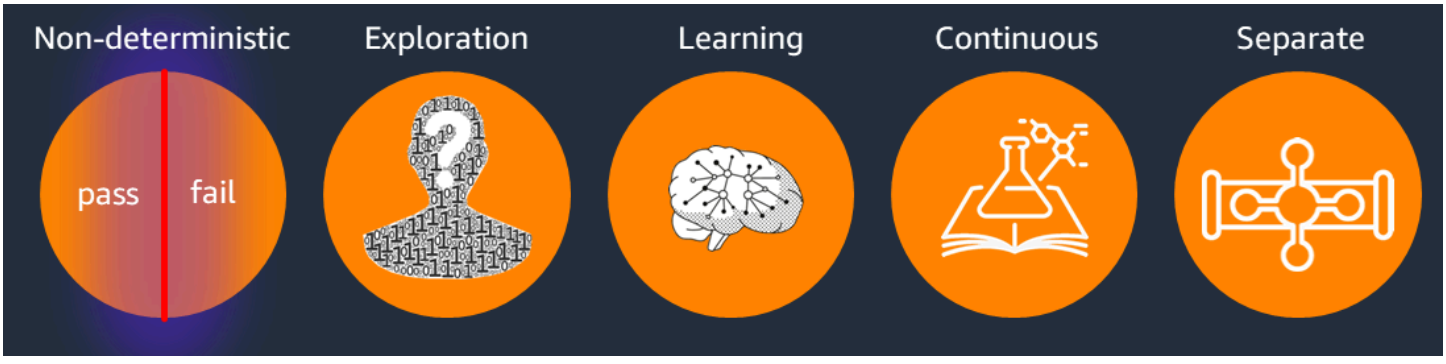
下一节将探讨混沌工程的特征与传统的弹性测试（例如单元测试、烟雾测试或集成测试）有何不同。

概述

将弹性测试与混沌工程进行比较

弹性测试是确定性的。也就是说，它可以验证应用程序中已实现的弹性机制的已知特征，例如断路器、重试、故障转移或回退。它证实了这些应用程序组件是如何吸收受控中断的，而对用户的影响微乎其微，甚至没有影响。因此，弹性测试侧重于验证已知的故障模式，这些模式注入到应用程序组件中，目的是产生 pass/fail 结果。作为流程中的一个步骤，你应该持续运行弹性测试，以确保你的弹性态势不会出现倒退。在弹性测试中，你通常不对真实组件运行测试，而是模拟模拟特定组件的 API。这种方法允许在受控环境中对故障场景进行一致、可重复的测试，使其适用于自动化管道集成和回归测试。

相比之下，混沌工程是非确定性的。也就是说，它基于假设，可以验证你的思维模型，即应用程序及其依赖关系（人员、流程和技术）如何吸收、适应并最终从意想不到的故障模式中恢复。因此，混沌工程侧重于对未知故障模式进行端到端验证，目标是尽早发现缺陷，并在缺陷演变为大规模事件之前对其进行修复。混沌工程可以促进持续学习，应通过单独的混沌管道或临时实验来练习，这样你就可以在任何时间点运行多个实验，而不会阻碍开发人员部署代码的工作效率。



混沌工程过程通常从混乱游戏日开始，这是一个专门的活动，团队故意在应用程序中注入受控的错误或故障。游戏日是渐进的：它从较低级别的环境（例如开发或测试）开始，随着信心的增强，逐渐发展到更高级别的环境（例如舞台和预制环境）。通过系统地在这些环境中移动，团队可以在注入的故障进入生产之前验证其系统是否能正确容忍这些故障。这种有条不紊的进展确保了在生产环境中进行混沌实验

时，团队已经对其系统的弹性能力建立了极大的信心。游戏日流程是一种主动的方法，用于识别应用程序架构和操作实践中的弱点和漏洞，同时消除意外生产中断期间的学习压力。

混沌工程的价值

复杂的系统在当今世界无处不在。从金融市场到医疗保健，它们在我们生活的许多方面都起着至关重要的作用。我们期望这些系统始终处于运行状态。但是，复杂的系统通常容易受到意外事件和行为的影响，这些事件和行为可能会产生重大后果。Organizations 需要为颠覆做好计划，而不必怀疑它是否会发生。他们可以通过在关键或任务关键型业务服务中应用场景测试来做到这一点。这就是混沌工程发挥作用的地方。

混沌工程提供了一种管理复杂系统的方法，可以帮助降低风险和提高弹性。为混沌实验做准备的过程要求团队对系统的行为提出假设，这可以加深他们对系统的构建方式和运行方式的理解。这种准备工作通常会揭示心理差距、架构见解和操作知识，否则这些知识可能仍未被发现。通过进一步了解复杂系统如何应对故障，混沌工程可以提高系统设计和管理的透明度和问责制。您的组织实施混乱工程的频率越高，他们在运营方面的准备就越充分。混沌工程可以帮助你建立设计弹性应用程序的最佳实践，这些应用程序可以在组件故障中幸存下来，对用户影响最小甚至根本没有。这可以确保关键应用程序在预期的服务级别和抗冲击能力范围内运行，同时不断增强团队对自己系统的了解。

为不利条件做好准备

在此基础上再接再厉 AWS，您将使用不同类型的服务，包括亚马逊弹性计算云 (Amazon EC2) 等区域服务、区域服务（例如亚马逊简单存储服务 (Amazon S3) Simple S3）、全球服务（例如 AWS Identity and Access Management （IAM）、第三方软件即服务 (SaaS) 服务和本地服务。每种类型的服务都会暴露出不同的故障域，您需要考虑这些故障域。您如何为自己造成的事件或您的组织无法控制的第三方造成的事件做好准备？

为了帮助了解您的应用程序可能如何应对不利条件，可以使用 AWS Fault Injection Service (AWS FIS)。 AWS FIS 是一项完全托管的服务，用于以受控方式运行故障注入实验。您可以使用此服务来注入 AWS提供的场景，例如可用区电源中断和跨区域连接问题，或者通过将该服务提供的各种故障操作链接在一起来构建自己的实验。 AWS FIS 使您的团队能够持续练习和学习他们的应用程序将如何应对常见故障，并在发现缺陷时对其进行修复。

练习受控混沌工程

控制混沌实验的关键原理是：

- 从与您的生产环境相似的环境开始。

- 为实验建立假设并停止条件。
- 从小处着手。
- 控制你的混沌实验。
- 设置影响范围。
- 了解您的服务基准。
- 安排实验。
- 首先进行补救，然后进行实验。
- 密切监视实验。
- 从结果中学习。
- 确定发现的优先顺序，进行补救并进行验证。
- 在整个组织中传播所学知识。

要成功扩展混沌工程，必须以受控的方式实施混沌实验。使用时 AWS FIS，您可以使用 Amazon CloudWatch 警报创建停止条件。您可以将这些条件合并到实验模板中，以确保实验在越界时停止并回滚到其上次已知状态。 AWS FIS 还提供安全杠杆。当你使用这些杠杆时， AWS FIS 会停止和回滚账户中所有正在运行的实验 AWS 区域，包括多账户实验，并阻止启动新的实验。这样可以防止在某些时间段内注入错误，例如交易时间、销售活动或产品发布，或者响应应用程序运行状况警报。在手动脱离安全杆之前，安全杆一直处于接合状态。

进行混沌实验时，应定义保护措施以防止环境中出现不良副作用，尤其是在实验有可能影响生产中的应用程序的情况下。在计划实验时，要预测它可能对环境中的其他应用程序产生的任何不利影响。例如，其他应用程序可能会收到来自参与实验的应用程序的错误消息，如果它们共享基础架构，则可能会遇到高请求量或遇到资源争用。在进行实验之前，请记录这些风险并解决任何已知或不可接受的问题。

混沌工程入门

在进行实验之前，我们建议您准备一些基本要素，以充分利用混沌工程实践。这些基本要素包括：

- 可观察性（指标、日志、请求跟踪）
- 你想探索的现实世界事件或故障清单
- 通过领导力支持来赞助组织韧性
- 根据潜在的业务影响，优先考虑关键发现，而不是在进行混沌实验时发现的新功能

混沌实验的可观测性

可观察性包括指标、日志记录和请求跟踪，在混沌工程中起着关键作用。在运行实验时，您需要了解业务指标、服务器端指标、客户体验指标和运营指标。如果没有可观测性，您将无法定义稳态行为或创建有意义的实验来验证您对应用程序的假设是否成立。

指标

下图显示了您可以针对不同类型的应用程序的混沌实验跟踪的指标类型。

- 业务指标-稳定状态表示系统的正常运行，由您的业务指标定义。它可以用每秒交易量 (TPS)、每秒点击流、每秒订单数或类似的衡量标准来表示。当您的应用程序按预期运行时，它会显示出稳定的状态。因此，在运行实验之前，请验证您的应用程序是否运行正常。稳定状态并不一定意味着故障发生时不会对应用程序产生任何影响，因为故障的百分比可能在可接受的范围内。稳定状态是您的基准。

例如，支付系统的稳定状态可以定义为处理 300 TPS，成功率为 99%，往返时间为 500 毫秒。从视觉上看，可以将稳定状态想象成心电图 (EKG)。如果系统的稳定状态突然波动，则表明您的服务存在问题。

- Server-side 指标 — 要了解您的资源在实验期间的表现，您需要深入了解其在实验之前、期间和之后的性能。要衡量您的资源对的影响 AWS，您可以使用 Amazon CloudWatch。 CloudWatch 是一项监控应用程序、响应性能变化、优化资源使用并提供对运行状况的见解的服务。在实验期间，您需要捕获服务器端指标，例如饱和度、请求量、错误率和延迟。
- 客户体验指标 — 开启后 AWS，你可以使用 CloudWatchRUM 通过 Locust、Grafana k6、Selenium 或 Puppeteer 等工具模拟用户请求，从而捕获真实的用户指标。真实的用户指标对于进行混沌工程实验的组织至关重要。通过监控实验期间真实用户受到的影响，团队可以准确地了解故障和中断将如何影响生产中的客户。关键的客户体验指标包括第一字节时间 (TTFB)、最大内容绘制 (LCP)、下一次绘制的互动 (INP) 和总屏蔽时间 (TBT)。
- 操作指标 — 干预措施衡量您以自动方式缓解故障的成功程度，例如，在使用复制控制器或自动扩展等机制重启 pod、任务或 EC2 实例期间，成功的客户端请求延迟。能够在故障期间进行自动干预与良好的用户体验直接相关。了解这些缓解机制随着时间的推移是否存在任何偏差至关重要。通过定义成功和失败的自动缓解措施的指标，您可以创建指导方针，帮助识别整个系统的潜在回归情况。

日志记录

集中式日志记录是了解混沌实验之前、期间和之后的应用程序组件状态的关键。我们建议您从所有应用程序组件中收集日志，以构建一个统一的视图，了解每个组件在注入实验时正在执行的操作。这为端到端实验流程提供了清晰的画面。

请求跟踪

请求跟踪使您能够观察应用程序中任何单个请求在各个组件中的流动，从而全面了解注入的故障对系统及其依赖关系的影响。通过跟踪请求，您可以看到故障是如何通过不同的服务和组件传播的，因此您可以更好地评估中断的范围。要跟踪您的请求 AWS，可以使用AWS X-Ray。

混沌实验中注入的失败场景

将常见故障注入应用程序的目的是了解应用程序对这些意外事件的反应，以便您可以创建缓解机制并使系统能够抵御此类故障。此外，您应该使用混沌工程来重播历史故障场景，以验证缓解机制是否仍按预期运行，并且不会随着时间的推移而发生偏差。

在计划混沌工程实验时，请考虑以下事件。

故障模式	说明
服务器损坏	重启 EC2 实例，删除 Kubernetes pod 或亚马逊弹性容器服务 (Amazon ECS) 任务，以了解您的应用程序对此类崩溃的反应。
API 错误	向你自己的服务 API 中注入 AWS 错误，以了解应用程序的行为。
网络问题	引入延迟或拥塞，或者阻塞连接以模仿现实世界中的网络问题。
AWS 可用区受损	重播整个可用区的损失，以验证跨区域的恢复。
AWS 区域 连接受损	重播网络损害 AWS 区域 ，以验证次要区域中的资源对此类事件的反应。
数据库故障	对数据库副本或损坏的数据进行故障切换，或者使数据库实例无法访问，以了解对应用程序和恢复策略的影响。
暂停数据库和 Amazon S3 复制	暂停跨可用区域的数据库或 Amazon Simple Storage Service (Amazon S3) 复制， AWS 区域 或者了解下游应用程序的影响。
存储降级	暂停 I/O、移除 Amazon Elastic Block Store (Amazon EBS) 卷或删除文件以验证数据持久性和恢复能力。
依赖性障碍	降低或降低您所依赖的下游或上游服务（包括第三方服务）的性能，以了解端到端的流程和对客户的影响。
流量激增	生成用户流量峰值以测试自动扩展功能，并查看冷启动时间会如何影响您的整体应用程序状态。
资源枯竭	最大限度地利用 CPU、内存和磁盘空间，以验证应用程序是否正常降级。

故障模式	说明
级联故障	启动主故障，这些故障会级联到下游应用程序和组件。
部署不好	推出有问题的更改或配置，以验证回滚机制。

组织复原力赞助

混沌工程在整个组织中应用时可以提供最大的价值。我们建议您与执行发起人合作，他可以帮助您在整个组织中设定韧性目标，消除对该领域的恐惧、不确定性和疑虑，并开始转型过程，让韧性成为每个人的责任。

为了支持建立混沌工程实践的商业案例，请将混沌工程工作投入到关键业务项目中。让韧性成为加速的资产和驱动力，将有助于你随着时间的推移跟踪成功情况。首先计算每月或每个季度的严重事件、平均恢复时间以及这些事件对您的客户和组织造成的影响。与您的团队一起设定一个目标，即在 6 到 12 个月内减少事件数量，因为针对混乱工程实验对应用程序堆栈进行了改进。

衡量事件是否高度重复。例如，假设过期的 TLS 证书会导致停机，因为客户端无法建立可信连接。如果由于多个 TLS 证书过期而在一年内发生多起事件，则可以对 TLS 证书过期进行实验，并验证您的团队是否收到警报或能够自动缓解此类问题。这将有助于确保您能够抵御此类故障。

要跟踪混沌工程在一段时间内的进展，请捕获以下指标，以帮助突出混沌工程在应用程序生命周期中的价值：

- 降低事故率 — 跟踪一段时间内的生产事故数量，并将该数字与混沌工程的采用相关联。预计严重事件的发生率将下降。
- 改善平均解决时间 (MTTR)-计算解决事件所需的平均时间，并跟踪这些数据，以查看混沌工程是否会随着时间的推移而有所改善。
- 提高应用程序可用性-使用服务级别指标来显示随着应用程序弹性通过混乱实验提高可用性的提高。
- 缩短上市时间 — Chaos 工程可以让您有信心更快地推出创新产品，因为您知道自己的应用程序具有弹性。追踪产品发布速度的提高。
- 降低运营成本 — 量化诸如警报噪音、操作负荷和管理应用程序的人工工作量之类的指标是否随着混乱实践的到位而减少。
- 增强信心 — 调查开发人员、站点可靠性工程师 (SRE) 和其他技术人员，以评估混乱工程是否增强了他们对应用程序弹性的信心。感知很重要。

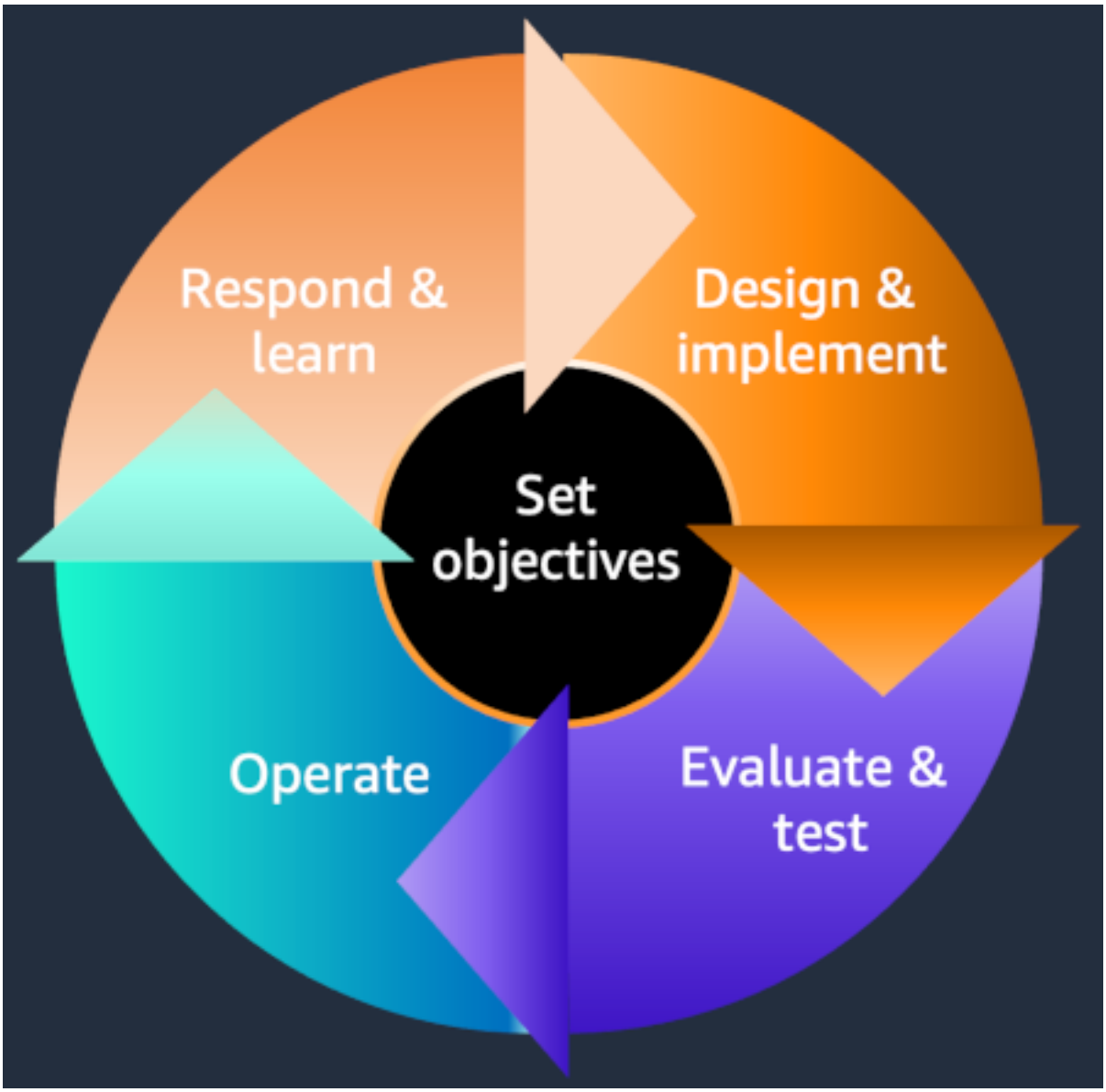
- 改善客户体验 – 将混乱工程与客户的积极成果联系起来，例如减少服务中断、回滚和中断。

确定补救的优先顺序

在执行混沌实验时，您可能会发现应用程序无法按预期运行的需要改进的领域。修复此类项目将成为待办事项中的工作，必须与其他工作（例如功能开发）一起确定优先顺序。我们建议您为这些增强留出时间，以避免 future 出现故障。考虑根据这些学习和补救任务可能造成的影响程度对其进行优先排序。直接影响应用程序弹性或安全性的发现应优先于新功能，以避免对客户造成影响。如果团队难以将补救工作置于功能开发之上，请考虑联系您的执行发起人，确保根据业务风险承受能力设定优先级。

在上实施混沌工程 AWS

混沌工程是AWS 弹性生命周期评估和测试阶段的一部分，如下图所示。分布式应用程序不能与其他应用程序或客户端隔离运行，因此我们建议您查看整个弹性生命周期。随着网络的发展、上游和下游应用程序的变化以及客户端的使用随着时间的推移而变化，分布式应用程序的变化是恒定的。



要了解应用程序的这些更改会如何影响其弹性，请将混沌工程作为日常运营的一部分。你可以用不同的方式实现混沌实验：

- Ad hoc — 您可以将混沌实验作为一次性实验来解决特定的问题或问题。

- 混乱游戏日 — 这些是结构化和重复发生的事件，旨在验证应用程序的可靠性和弹性。混乱游戏日的目的是识别人员、流程和技术中潜在的弹性问题或缺陷，并练习识别、缓解和响应事件的流程和程序。
- Chaos pipeline — 持续集成和持续交付 (CI/CD) 旨在构建新功能并将其安全地部署到整个环境中。要实现混沌工程实验，请创建一个与您的管道分开的混沌 CI/CD 管道。为了理解原因，让我们假设您要在 CI/CD管道中添加一个混沌工程实验，该实验会给下游组件注入越来越多的丢包。该实验运行 3 次，每次需要 5 分钟才能完成。每次运行时，丢包率从 10% 增加到 20% 再到 30%，实验总共需要 15 分钟才能完成。如果您有 100 个 parallel 部署，则必须等待 1500 分钟才能完成单个实验。如果你要进行 10 个实验，那么对你的开发者的影响将难以忍受。在规模上，混沌工程需要自己的管道，允许您与软件开发生命周期 (SDLC) 流程并行运行实验。
- Can@@ ary 部署 — 加那利群岛为混沌实验提供了测试环境。通过将一小部分流量引导到金丝雀服务或使用流量镜像或重播等方法，您可以验证新的基础架构或代码更改，而不会对稳定的生产系统产生任何影响。您可以针对金丝雀进行实验，并在必要时注入错误，因为您可以将影响范围限制在最终用户身上。
- 预定实验-您可以安排实验，以验证应用程序的可预测恢复机制。使用预定实验重播常见事件，以捕捉您的系统如何从自动扩展组后面的 EC2 实例、移除 Kubernetes 容器或删除 Amazon ECS 任务等事件中恢复。

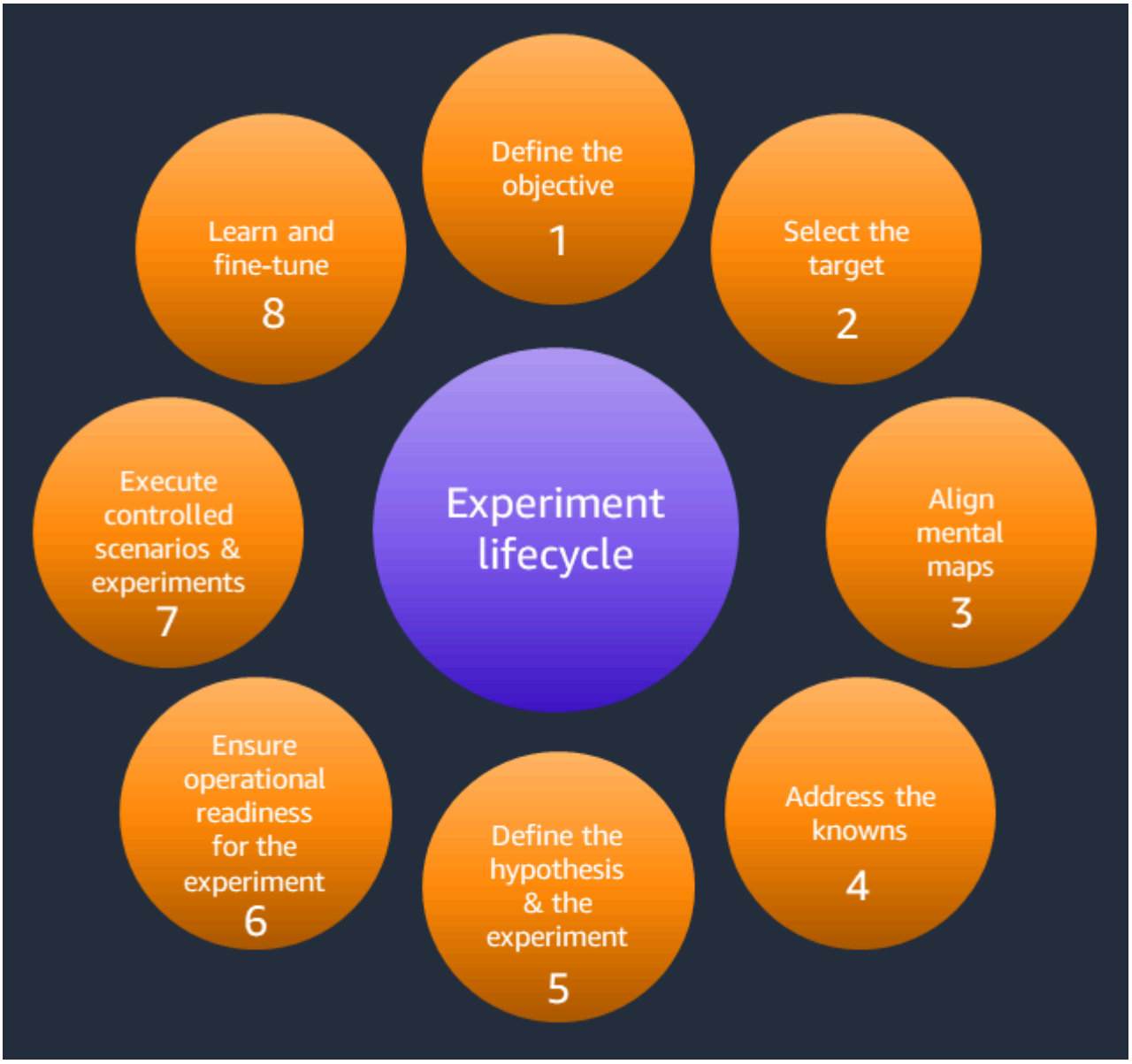
连续混沌工程实验生命周期

如上一节所述，你可以用不同的方式实现混沌工程实验。在所有情况下，进行有意义的混乱实验的关键是了解应用程序、历史事件和已实施的补救措施，并清楚地了解需要调查的领域，例如弹性或安全性。你对应用程序的了解可以帮助你对应用程序的潜在弱点提出假设，并了解在注入故障时它将如何检测、修复和恢复。

混沌实验生命周期包括以下步骤：

1. 定义实验的目标。
2. 选择目标应用程序。
3. 对齐思维导图。
4. 解决应用程序的已知问题。
5. 定义假设和实验。
6. 确保实验准备就绪。
7. 运行受控场景和实验。
8. 从实验中吸取教训并对其进行微调。

图中说明了这些步骤，并在以下各节中进行了讨论。



定义目标并设定期望

在每次实验之前，请确保您的目标和期望是具体的、可衡量的、可实现的、相关的和有时限的。明确定义以下内容：

- 识别系统和服务中的潜在故障或弱点，以了解它们可能如何影响用户。这包括识别可能的故障模式，例如服务器崩溃、网络故障或软件错误，并评估它们对系统整体性能和可靠性的潜在影响。
- 通过定义系统和服务的关键风险指标 (KRI) 来@@ 量化故障的影响。这包括在延迟、吞吐量和错误率等指标偏离稳定状态时测量故障的影响。通过了解此类偏差的影响，您可以根据业务风险确定缓解故障的优先顺序。

- 制定并验证缓解或预防故障的策略。这包括确定潜在的解决方案，例如冗余、错误更正或备用策略，并在受控环境中测试其有效性。通过验证这些策略，您可以确保有效预防或缓解故障，并且可以放心地将其部署到生产系统中。
- 改善事件响应和灾难恢复流程。通过在受控环境中重播故障，您可以测试事件响应流程，识别潜在的瓶颈或差距，并完善灾难恢复程序。这有助于确保您做好准备，以便在发生意外故障时快速有效地做出响应。

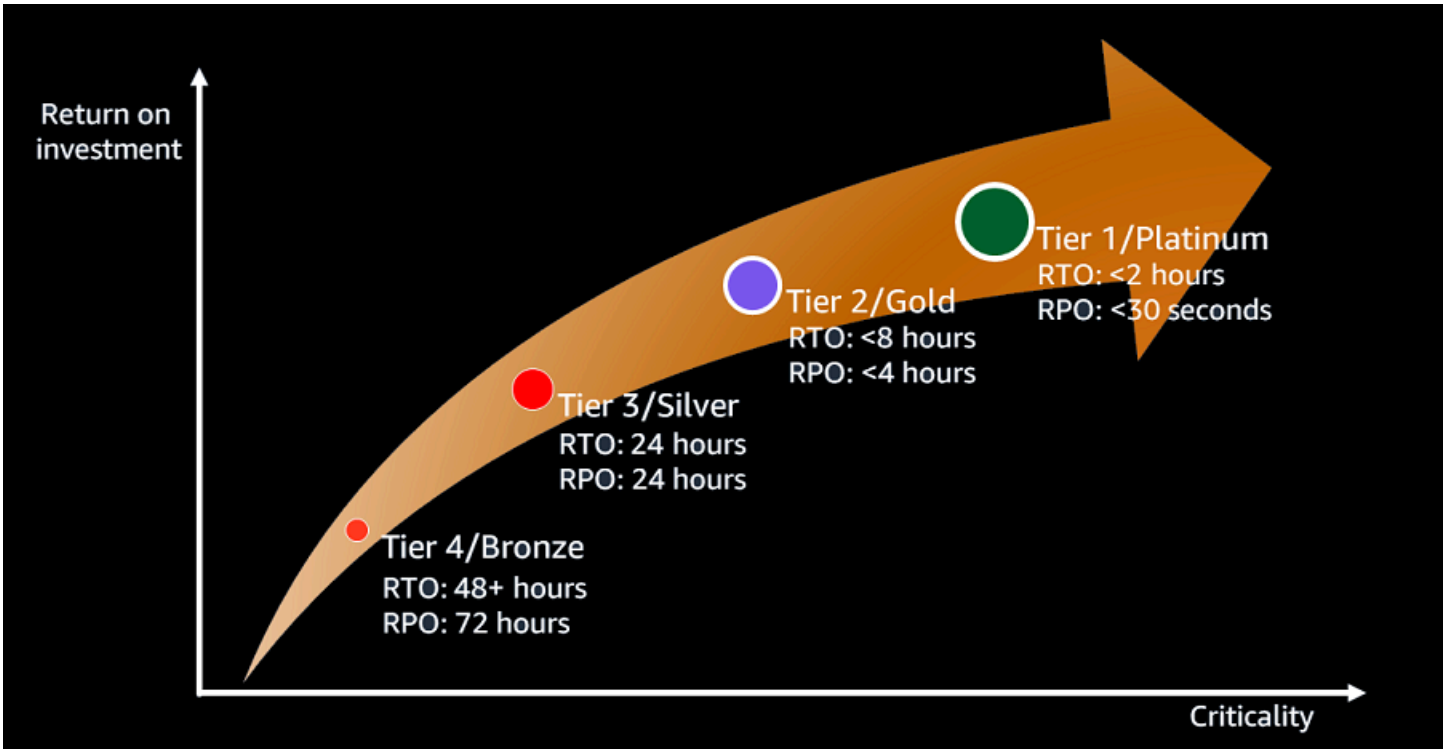
选择目标应用程序

混沌工程是一项强大的技术，但需要深思熟虑的优先级排序才能最大限度地提高价值。在决定将混沌工程工作重点放在哪里时，首先要考虑企业的关键服务。要求您的团队在软件开发生命周期阶段进行迭代，然后首先开始在测试环境中注入错误。 Business-critical 应用程序与收入、客户体验和核心运营直接相关。对这些服务的混乱实验可以发现漏洞，如果不加以解决，这些漏洞可能会严重影响组织乃至整个市场。例如，首先关注面向客户的服务，例如交易系统或订单系统。优先考虑这些中央服务可以为每投入的时间提供最大的保护。

在关键服务之后，查看基础组件，例如数据库、消息队列、网络和共享服务 API。它们可能被用作整个组织的共享组件或服务，因此它们的故障将导致广泛的问题。确认基础设施服务的弹性可以让人们确信它们不会削弱其上方的依赖应用程序。例如，关闭 Kafka 集群的混沌工程实验揭示了很多关于下游应用程序容错能力的信息。尽管系统基础设施不是直接面向客户的，但它是一个主要的混乱工程目标。

别忘了绘制人员、流程、设施信息和第三方依赖关系的心理差距，因为如果这些差距与组织的影响容忍度目标不一致，可能会造成重大干扰。有关衡量混沌工程投资回报率的更多信息，请阅读战略文档中的“量化混沌工程的投资回报率”。将混沌工程作为战略必要性进行投资。

下图显示了在不同服务层级上运行混沌实验的投资回报。



对齐思维导图（应用程序发现）

当你进行临时实验或游戏日时，你将通过举行白板会议来开始应用程序发现过程，重点是绘制应用程序的细节。（如果你在混沌管道中运行实验，那么通过定义目标应用程序，你已经对齐了思维导图。）理解心理差距的一个好方法是让最年轻的团队成员先画出应用程序的示意图，然后让更高级的工作人员逐步添加到图表中。这将揭示不同经验等级之间在理解方面存在的任何差距。

该图应描述应用程序的直接上游和下游依赖关系，以及任何关键的第三方集成。确保通过应用程序的请求的预期流程保持一致。绘制关键工作流程和用户旅程，以清楚地了解客户如何使用该应用程序。考虑使用序列图来捕获这些信息。

在这次协作会议之后，团队应该对应用程序、其关键依赖关系和监控能力有一个共同的思维模型，并了解风险，以便做出明智的决定，继续或取消潜在的混乱实验。

解决应用程序的已知问题

混沌工程实验旨在主动发现应用中的缺陷。通过注入延迟增加、服务器重启或可用区电源损坏等故障，您可以验证您的应用程序是否能够容忍实际中断。但是，此过程假设目标应用程序具有基本的稳定性和运行状况。在已经存在问题的应用程序上进行混沌实验有可能掩盖更深层次的问题。

在进行混沌工程之前，团队应解决其应用程序中的任何已知缺陷、错误和性能问题。

定义假设和实验

过去导致您的应用程序或组织内其他应用程序中断的事件可以成为混乱实验想法的绝佳来源。例如，之前的中断是由配置错误还是缺少弹性模式引发的？ 回顾事件历史并通过混沌实验重现现实世界失败的根本原因，是培养未来应对类似问题的适应能力的有效方法。

实验概念的另一个宝贵来源可以直接来自最熟悉应用程序的工程师、架构师和操作员。允许团队成员提交他们认为可能会严重干扰应用程序的假设失败场景，这样您就可以根据内部知识收集想法。然后，应用团队可以评估这些拟议方案中哪些可能产生最大的潜在影响或暴露最大的未知风险。针对此类高风险、鲜为人知的场景进行混沌实验可以产生重要的学习并防止将来出现问题。

第三个想法来自于进行弹性建模，以预测可能导致已确定的业务损失的情况。一些弹性建模练习采用基于组件的方法来构建弹性模型，而另一些则采用基于系统的方法。基于组件的方法会问这样一个问题："当组件 x 处于极端负载下或出现故障时会发生什么？" 然后，开发弹性模型的团队推测了这种情景对更广泛应用的影响，并确定了目前为检测和减轻该情景的影响而采取的监测和预防性控制措施。或者，基于系统的方法遵循自上而下的流程来突出显示应用程序的不良状态，例如 "网络店面显示过时的库存水平"，并邀请应用程序团队预测哪些或哪些条件会导致应用程序以这种方式运行。

确保实验准备就绪

您需要可量化的指标来衡量不利条件对应用程序及其行为的影响，如前面关于可观察性的部分所述。能够测量应用程序的行为使您能够确定不利条件是否对应用程序产生了影响以及影响程度有多大。

要了解您的应用程序是否受到影响，最好的方法是测量其稳定状态。稳定状态衡量正常运行的样子，通常与给定应用程序的业务和客户体验指标保持一致。在进入下一步之前，请确保具备可观察性以了解影响，并准备好回滚机制，以防实验结果不如预期。

运行受控实验和场景

在 AWS，我们不建议对正在生产的应用程序进行初始混沌实验。混沌实验的目的是学习一些关于应用程序在压力条件下的行为的新知识。在实验过程中，应用程序的行为可能是不可预测的，因此首次在生产环境中进行实验可能会对客户产生影响。因此，您应始终在影响真实用户可能性最小的较低级别环境中运行初始混乱实验，然后在验证并确信您的应用程序可以吸收、适应注入的操作并从中恢复过来之后，在您的环境中进行迭代。

使用记录关键细节的文档（类似于附录中提供的实验计划文件）对每个实验进行全面规划。要包括的一些关键领域是稳态定义、假设和失效注入方法。混沌实验的规划、执行和分析可以在单个工件中完成。

完成实验的书面计划后，准备好所有必要的代码，以注入文档中概述的计划中断。

为了捕捉实验期间的潜在影响，请确保可观测性机制到位。如果您还没有自动捕获实验结果（例如 AWS FIS 实验报告）的方法，请确定将在执行过程中记笔记、捕获仪表板屏幕截图并带领团队完成实验的团队成员。

学习和微调

每次实验结束后，作为一个团队聚在一起回顾和反思混沌实验。有意识地努力保持无可指责的心态。你的目标应该是进行公开、建设性的对话，完全侧重于获得最大限度的学习，而不是指责别人。

首先回顾实验的稳态定义和假设。应用程序的行为是否符合预期？ 有没有让假设失效的意外？ 讨论对应用程序在实验期间反应的观察结果，包括好与坏。收集的数据——指标、日志、屏幕截图等——应该准确地讲述发生了什么。

以好奇心而不是判断力来进行此次数据审查，并根据所学知识确定可以改进应用程序设计、文档、监控或其他功能的领域。这些行动项目被列为后续项目，以提高应用程序的弹性。

通过这种无可指责的方法，你可以坦率地谈论出了什么问题以及如何解决问题。假设所有参与其中的人都有积极的意图，并相信他们正在努力取得良好的结果。您的共同目标是通过持续的学习和适应来实现组织发展和进步。以建设性、无可指责的方式进行的混沌实验审查为您的团队提供了一个安全的空间，让他们获得宝贵的见解，从而使您的应用程序和组织从长远来看更加可靠和有弹性。重点仍然放在学习上，而不是人们身上。要在团队中传播所学知识，请在中心位置发布实验结果报告并公布结果，以便其他人可以从中吸取教训。

在整个组织中扩展混乱工程

当您的组织采用混沌工程时，对其进行标准化和实施将带来挑战。在成熟的早期阶段，不同的团队可能会使用不同的工具和前几节中描述的混沌工程过程的变体。同时，尽管混沌工程具有潜在的好处，但有些团队可能根本不会优先考虑或采用混沌工程。以下各节为如何克服这些挑战提供了指导。

总体而言，你的混沌工程方法应设计为在集中领导和分散参与之间取得平衡。这种平衡有助于确保将混沌工程整合到开发流程中，并在整个组织中共享所学知识。

建立混沌工程实践

标准化混沌工程实践可以加快混沌工程的采用。跨团队共享从实验中学到的经验可以放大混沌工程投资的回报。

作为混沌工程实践的一部分，建立一个集中的卓越中心，或召集一组主题专家。作为一个小型的集中式职能，该团队可以在软件开发、基础架构、安全和业务团队中运作，并维护这些团队使用的标准。为简单起见，卓越中心被称为集中式实践团队，在本指南的其余部分中，应用混沌工程的小组被称为实践团队。

集中式实践团队的作用

集中式实践团队负责在整个组织中开发和实施混沌工程实践。他们与实践团队密切合作，指导他们设计和进行实验，并确保实验对业务有价值。集中式实践团队还为开发、基础设施和安全团队提供指导和支持，帮助他们将混沌工程整合到开发流程中。

集中式混沌工程实践团队的主要职责包括以下内容：

- En@@ able ment — 集中的混沌工程职能部门充当主持人，通过游戏日和研讨会介绍混沌工程的实践。他们指导团队进行混沌工程，包括选择故障情景、定义假设以及生成报告以与更广泛的组织共享。集中式实践团队应拥有培训材料，并努力提高练习队使用混沌工程的技能。
- 咨询 — 集中式实践团队还可以发挥咨询作用，监督实践团队进行的实验。他们的经验和知识可以确保实验为业务带来价值，并且以安全的方式进行。同样，该团队可以监督实验的执行和汇报，为刚接触混沌工程的人提供指导。
- 营销和价值跟踪 — 传达混沌工程的商业价值是此类计划成功的关键。每个参与混沌工程实验的团队都应从整个业务的实验中收集数据，并证明组织对混沌工程的投资的价值。这包括量化和庆祝每次实验期间避免的事故数量、实验失败后可能产生的停机时间，以及如果生产中出现故障情景对业务的总体影响。通过收集和集中来自各个团队的此类数据，并在整个组织中提供数据，集中式实践团队可以跟踪和影响在整个组织中采用混沌工程所产生的价值。

- 标准 — 集中式实践团队应拥有并维护进行混沌实验的流程、规划和报告实验的模板以及用于进行实验的工具。

  中心团队应拥有并管理实验计划模板、实验报告模板、流程文档和支持材料。最佳实践文档和支持材料为实践团队提供了有关主题的指导，例如他们可以用来限制实验影响的护栏、何时在生产中进行实验，以及如何随着时间的推移发展混沌工程的使用。有关模板和输出的示例，请参阅附录。

  集中式实践团队还应拥有进行实验的流程，包括沟通和上报，以及在实验之前或实验期间何时以及如何与组织中的其他团队沟通。该过程还应概述何时需要护栏。

  集中式实践团队还应选择并拥有用于进行混沌实验的核心工具（例如，诸如的工具 AWS FIS）。诸如负载生成工具之类的补充工具的选择和实施应由实践团队来决定。实践团队应该能够调整整体流程和工具，以最好地满足他们的需求。

练习队的作用

集中式团队负责推动整体混沌工程策略，而实践团队则参与该过程并拥有实验的开发和执行。这有助于确保实验与每种特定的产品或服务相关，并且所学知识是可操作的，可以应用于提高产品的可靠性和弹性。集中式实践团队充当组织混沌工程标准和流程的导师和所有者。但是，为了防止集中式团队成为瓶颈，各个练习队需要向中心实践学习，自己进行混沌实验。

建立实践社区

除了创建一个集中的团队外，我们还建议您建立一个由对混沌工程感兴趣的从业者组成的非正式社区。该社区提供了一个平台，用于在实践团队和更广泛的组织之间共享知识、最佳实践和经验。

实践社区可以由集中的混沌工程实践团队运营，但组织中的任何人都可以成为社区的一员。集中式团队可以利用实践社区来广播最新动态和来源学习成果，并从使用集中式团队管理的标准和流程的实践团队那里收集反馈。社区将充当反馈循环，向集中式团队通报混沌工程实践在整个实践团队中的有效性。然后，集中式实践团队可以调整其文档和支持工件，以最好地支持产品团队。

将混沌工程融入您的运营弹性中

混乱实验是您的企业为防止生产中发生事故而进行的投资。有必要确定企业在哪些方面可以从这项投资中获得最大的回报。该组织可以与集中的混沌工程实践团队合作，更新其标准，并确定哪些产品足够重要，需要进行混沌实验。

系统开发流程

混沌工程和混沌实验应作为应用程序生命周期的一部分反复执行。与团队定期进行灾难恢复测试的方式类似，他们应该在一年中持续定期地进行混乱实验和比赛日。这种方法可以改善组织对事件的预期、观察和响应方式。

结论

在过去的十年中，混沌工程学科已经取得了长足的进步。它已被各行各业所采用，并帮助组织创建弹性服务并提高客户满意度。（有关组织如何实施这些实践的示例，请参阅混沌工程故事。）混沌工程使组织能够通过在其应用程序堆栈的各个级别（包括云提供商服务）注入受控故障，从而降低其任务关键型应用程序的风险。能够以可控的方式影响整个应用程序堆栈，可以实现持续的弹性、卓越的运营、可观察性和以恢复为导向的架构，而不会受到生产中断的压力。这种方法还可以改善整个组织的弹性测试实践。要开始拥抱混沌工程，请举办混沌游戏日或研讨会，展示混沌工程可以为您的组织提供的价值。

资源

AWS FIS 失败模型实现：

- AWS Fault Injection Service 用于演示多区域和多可用区应用程序弹性（AWS 博客文章）
- AWS 故障注入模拟器支持 Amazon EKS Pod 上的混沌工程实验（AWS 博客文章）
- 宣布推出适用于 Amazon ECS 工作负载的 AWS 故障注入模拟器新功能（AWS 博客文章）
- 使用 Amazon EBS 容量指标和 AWS 故障注入模拟器提高应用程序弹性（AWS 博客文章）
- 在@@ 多账户 AWS 环境中利用 AWS 故障注入模拟器的混沌工程（AWS 博客文章）
- 使用 AWS 故障注入模拟器在 Amazon RDS 上进行混沌实验（AWS 博客文章）
- 使用混沌工程构建弹性无服务器应用程序（AWS 博客文章）
- 使用 FIS 中断现货实例（AWS 工作坊）
- 在@@ 交付管道中实现混沌工程自动化（AWS 社区帖子）

其他资源：

- 投资混沌工程作为战略必要条件
- 混沌工程故事
- 亚马逊 CloudWatch文档
- 亚马逊 CloudWatch RUM 文档
- AWS X-Ray 文档
- AWS FIS 文档

附录：示例文档

本节提供的示例文档使用虚构的宠物收养网站 (PetSite) 作为混乱实验的目标应用程序。

- 实验计划文件
- 实验结果文档

实验计划文件

稳定状态

进程名称	宠物收养网站
物理架构	（链接到架构图。）
逻辑架构	（链接到逻辑图。）
定义稳定状态	使用最大内容绘图（LCP）衡量，宠物收养网站的平均页面加载时间为2.5秒或更短，99个百分位延迟（P99）为4.0秒或更短，基准为5000个并发用户。
稳定状态指标	跨用户群捕获的 LCP 指标和黄金指标（延迟、吞吐量、错误率、饱和度）。
稳态可观测性	LCP 将被用户的浏览器捕获，发送到 Amazon CloudWatch，然后使用 CloudWatch RUM 进行检查。在 60 秒的时间段内，将汇总该时间段内所有请求的平均时间和 P99 LCP 时间。 Top-level 黄金指标是通过使用来捕获的 CloudWatch。
实现稳态的过程	Grafana K6 将用于创建负载，该负载可模拟大约 5K 并发用户的正常生产流量水平。

可观测性要求

团队应该能够查看以下内容：

- 稳定状态：要验证应用程序是否处于正常状态，需要观察什么？
- 故障状况：故障状态将如何显示在仪表板上？ 例如：
  - 应触发的警报
  - 应该生成的日志
- 故障影响：要查看预计会受到影响的组件（影响范围），应注意什么？
- 恢复：如何看待和衡量恢复以捕获 MTTR？
- 调试：有关实验失败的疑难解答详细信息。

下表提供了可观测性要求图表的建议和示例。你应该根据你的具体实验来定义应该观察的内容。

需要注意什么	链接到可观测性工具	正在观察到什么
输入来源	Grafana K6 仪表板	• 正在运行的容器数量 • 每秒请求数
应用程序的整体运行状况	• 宠物收养 CloudWatch 仪表板 • 宠物收养用户体验仪表板 (RUM)	• Amazon EKS 健康节点数 • 亚马逊 EKS 节点 CPU 利用率
工作流程运行状况	宠物收养 CloudWatch 仪表板	LCP 时间，黄金指标
跟踪	宠物收养 X-Ray 仪表板	• 请求延迟 • 请求计数 • 失败次数
日志	宠物收养 CloudWatch 日志	Pod 遇到的任何错误都将发送到 CloudWatch Logs。

实验定义

实验名称	亚马逊 EKS p PetSite od CPU stress
实验源代码	（链接到实验源存储库。）
实验描述	本实验探讨了 PetSite 应用程序 pod 的 CPU 使用率的增加将如何影响整体客户体验。 通过向每个正在运行的 PetSite pod 注入 CPU 压力，我们将能够了解是否会对客户产生影响以及这种影响的程度。
实验要求或参数	应用程序负载：平均产量 Pod 标签选择器：`app=petsite`
实验持续时间	10 分钟
环境	Alpha 测试环境
实验目标资源	PetSite 应用程序窗格
通过负载生成工具引入的实验基线	• 54% 的请求的 LCP 小于 2.5 秒。 • 46% 的请求的 LCP 小于 4 秒。 • 未发现任何错误。
退避条件	无

假设

如果	影响	恢复
如果在正常生产级流量下，PetSite应用程序 pod 在 10 分钟内经历或导致 CPU 使用率	P99 为 4.0 秒或更短的 P50 用户的 LCP 时间将保持在 2.5 秒以下。消费者应该能够加载 PetSite 登录页面。	检测： • CPU 压力将通过中配置的警报进行检测 CloudWatch。

如果	影响	恢复
超过 60%，那么稳定状态会怎样？		• LCP 指标还将针对用户体验下降生成警报。 Self-healing: • 微服务架构的分布式特性意味着许多 Pod 实例在多个可用区中运行。 • EKS 集群控制平面将从受影响的 pod 转移流量，并在工作节点上启动新的 pod。 恢复： 当 CPU 利用率恢复正常时，LCP 应自动恢复。

实验过程

根据您的特定实验量身定制以下示例分步流程：

1. 验证对所有 Amazon CloudWatch、 CloudWatch RUM 和 AWS X-Ray 控制面板的访问权限和功能。
2. 验证应用程序环境的运行状况：
   a. 使用 CloudWatch控制面板确认 EKS 集群运行正常。
   b. 通过示例 URL 访问测试宠物收养网站应用程序部署。
3. 启动负载以达到稳定状态：
   a. 确认负载生成器正在运行，并且每秒发送 5000 个请求。
   b. 等待 5 分钟，让应用程序达到稳定状态。
   c. 使用 CloudWatch RUM 控制面板确认应用程序的稳定状态。
4. 启动故障（实验）：

a. 打开控制 AWS FIS 台。

b. 运行宠物收养舱压力实验。

c. 确认实验正在控制台中运行。

5. 观察故障对应用程序的影响：

a. 从 CloudWatch RUM 和 CloudWatch 仪表板中捕获屏幕截图，并记下任何异常数据点。

b. 实验完成后 AWS FIS，捕获更多屏幕截图以记录应用程序在没有 stress 的情况下是否恢复到稳定状态，并记下数据点中的任何异常。

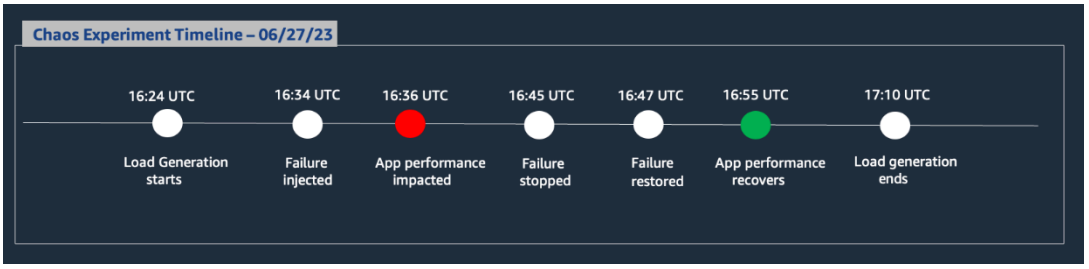
c. 如果稳定状态未恢复，请采取措施恢复应用程序并记录所采取的步骤。

6. 验证环境是否已恢复正常：

- 查看所有业务、用户体验、应用程序和基础架构指标，以验证系统是否已恢复到已知状态。如果有帮助，请捕获仪表板屏幕截图

实验时间表

确保捕捉端到端实验的时间表，从负载生成、故障注入、影响观察和应用程序恢复，到停止负载生成时结束。以下示例对此进行了说明。



实验结果

实验运行 ID	实验结果
PET-ADOPT-EXP-23	（链接到实验结果。）

已识别的缺陷

- Kubernetes 集群没有检测到 PetSite pod 的 CPU 损失，因此它没有安排额外的部署。
- 该实验没有导致4XX或5XX的错误率增加。
- 我们需要调整 Pod 的运行状况检查，以考虑资源限制时对 LCP 的影响。

实验结果文档

配置

记录实验的具体配置。例如：

- 负载生成设置为模拟 5K 用户每秒总共发出 85 个请求。

先决条件

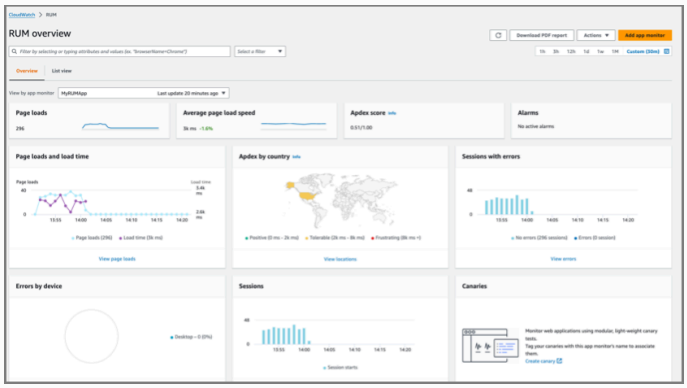
- 已验证宠物收养网站是否在 alpha 测试环境中运行。
- 已验证实验模板是否已配置为对 EKS 集群中运行的 PetSite 应用程序 pod 施加 CPU 压力。 应用程序容器由 Kubernetes 标签识别。`app=petsite`
- Load 已确认正在运行，每秒生成 85 个请求。

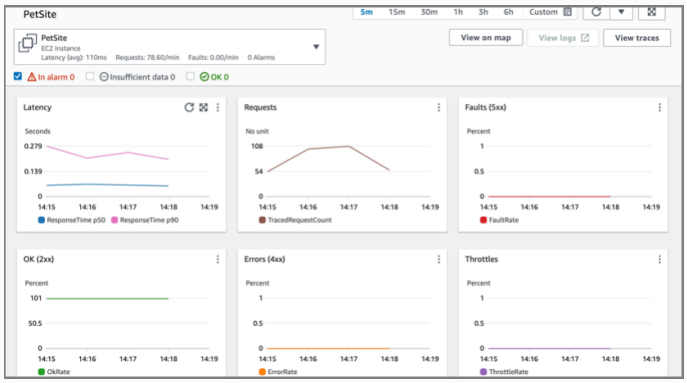
稳定状态

记录为实现稳定状态所采取的步骤以及您是如何验证稳定状态的。例如：

对于宠物收养场所的测试部署，正在生成 85 个 RPS 的负载以模拟稳定状态。在实验执行之前，对 CloudWatch RUM 和 CloudWatch 仪表板进行了审查，以验证所有业务和应用程序指标是否都在正常范围内。

可观测性数据：

预期	已观察
• P99 请求的 LCP 时间小于 4 秒。 • 响应延迟小于 500 毫秒。 • 没有 4XX 或 5XX 错误。	

预期	已观察
	

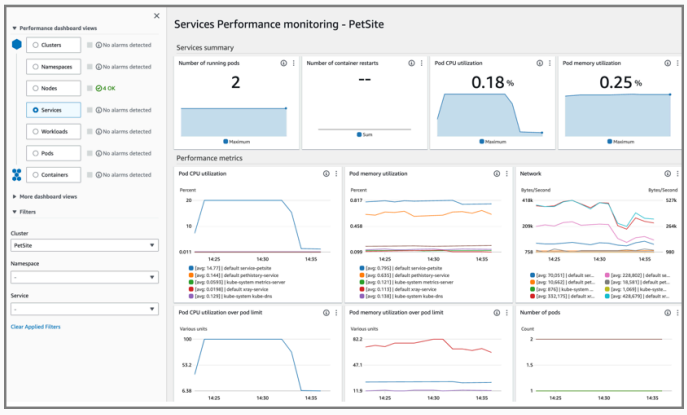
故障注入

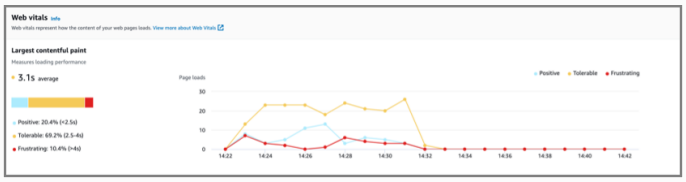
AWS FIS 用于使用实验模板注入故障（提供链接）。实验设置为运行 10 分钟，如果工作节点的 CPU 压力超过 60%，则配置回滚。

故障观察

对 CloudWatch RUM 和 CloudWatch 仪表板进行了审查，以跟踪应用程序的稳定状态（使用 LCP 指标定义）。 屏幕截图如下表所示。

可观测性数据：

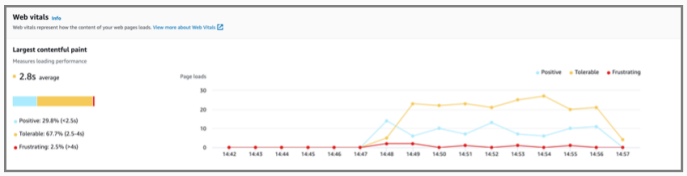
预期	已观察
• 对于 P99，LCP 应保持在 4 秒以内。 • 响应时间应保持在 500 毫秒以下。 • 不应遇到 4XX 或 5XX 错误。	

预期	已观察
	

恢复

消除压力后（ AWS FIS 实验完成并消除了 pod 的 CPU 压力），应用程序应恢复其正常的稳定状态。无需手动干预。

可观测性数据：

预期	观察（屏幕截图）
LCP P99 应低于 4 秒，平均值应低于 2.5 秒。	

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSS 源。

变更	说明	日期
初次发布	—	2025 年 4 月 4 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。