亚马逊云科技

亚马逊云科技规范性指南

韧性系统生命周期建设框架

Amazon Prescriptive Guidance：韧性系统生命周期建设框架

版权所有©2023 亚马逊云科技及/或其附属公司。保留所有版权。

亚马逊云科技的商标和商业外观不得用于任何非亚马逊云科技的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁亚马逊云科技的方式使用。所有非亚马逊云科技拥有的其它商标均为各自所有者的财产，这些所有者可能附属于亚马逊云科技、与亚马逊云科技有关联或由亚马逊云科技赞助，也可能不是如此。

目录

引言 1
术语和定义 2
持续韧性 3
第 1 阶段：设定目标 4
确定关键应用 4
确定用户故事 4
设定衡量指标 5
创建额外的衡量指标 5
第 2 阶段：设计和实施 7
Amazon Well-Architected Framework 7
理解依赖关系 7
灾难恢复策略 8
制定持续集成和持续交付（CI/CD）策略 8
开展运行就绪性检查 9
了解亚马逊云科技故障隔离边界 9
选择响应方式 9
韧性建模 10
故障安全 10
第 3 阶段：评估和测试 11
部署前活动 11
环境设计 11
集成测试 11
自动化部署管线 12
负载测试 12
部署后活动 12
开展韧性评估 12
灾难恢复测试 13
漂移检测 13
合成测试 13
混沌工程 13
第 4 阶段：运营 15
可观察性 15
事件管理 15
持续韧性 16
第 5 阶段：响应和学习 17
创建事件分析报告 17
实施运营审查 18
审查警报性能 18
警报精确度 18
假阳性 18
假阴性 18
重复的警报 19
实施指标审查 19
提供培训和赋能 19
创建事件知识库 19
深入实施韧性 20
结论和资源 21
撰稿人 22
文档历史 23

韧性系统生命周期建设框架：实现韧性优化的持续方法

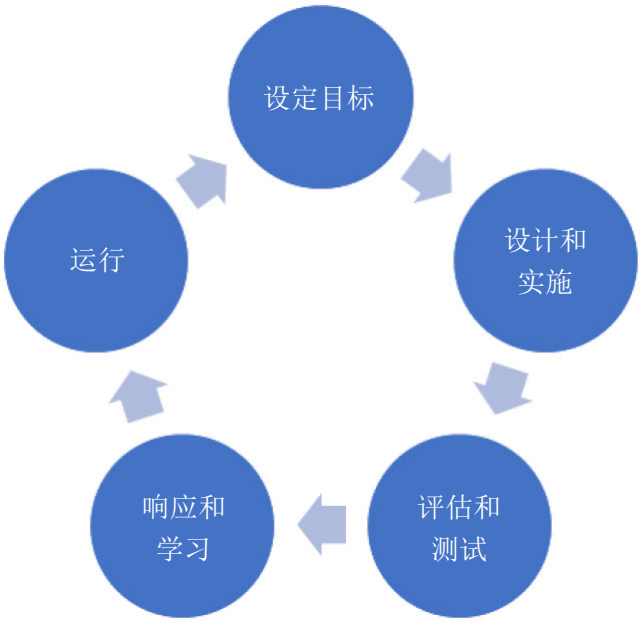
亚马逊云科技

2023 年 10 月（文档历史（23 页））

如今，现代公司面临着越来越多与韧性相关的挑战，这在客户日益期望服务“永远在线、永远可用”的背景下尤其如此。公司需要构建远程团队和复杂的分布式应用，同时也需满足客户对新应用程序不断增长的需求。因此，公司及其应用均需比以往更具韧性。

根据亚马逊云科技的定义，韧性是指应用程序抵抗中断（包括与基础设施、依赖服务、错误配置和瞬态网络问题等相关的中断）或从中断中恢复的能力（参见《Amazon Well-Architected Framework 可靠性支柱文件》中的“韧性和可靠性组件”部分）。然而，为了达到期望的韧性水平，公司通常需要进行权衡，需要对操作复杂性、工程复杂性和成本进行评估和相应调整。

在与客户和内部团队展开多年合作的基础上，亚马逊云科技开发了一个韧性系统生命周期建设框架，其中包括了韧性知识和最佳实践。该框架概述了下图所示的五个关键阶段。在每个阶段，您都可以运用相应的策略、服务和机制来优化韧性状态。



这些阶段将在本指南后续章节中予以讨论：

- 第 1 阶段：设定目标（第 4 页）
- 第 2 阶段：设计和实施（第 7 页）
- 第 3 阶段：评估和测试（第 11 页）
- 第 4 阶段：运行（第 15 页）
- 第 5 阶段：响应和学习（第 17 页）

术语和定义

每个阶段的韧性概念适用于从单个组件到整个系统的不同层面。概念的实施需要明确定义几个术语：

- 组件 是执行功能的元素，由软件和技术资源组成。组件包括代码配置、网络等基础设施，或者甚至服务器、数据存储和多因素身份验证（MFA）设备等外部依赖项。
- 应用程序 是交付商业价值的组件集合，比如面向客户的网络店铺或优化机器学习模型的后台流程。一个应用程序可能由单个亚马逊云科技帐户中的组件子集构成，也可能是跨多个亚马逊云科技帐户和地区的多组件集合。
- 系统 是管理既定业务功能所需的应用程序、人员和流程的集合。它包括功能运行所需的应用程序；运行流程（比如持续集成和持续交付（CI/CD）、可观测性、配置管理、事件响应和灾难恢复）；以及管理这些任务的操作人员。
- 中断 是指阻止应用程序正常实现其业务功能的事件。
- 受损 是指如果中断得不到缓解而将对应用程序产生的影响。如果应用程序遭受一系列中断，它们可能会因此受损。

持续韧性

韧性生命周期是一个持续的过程。即便在同一家公司中，应用团队对每个阶段的执行完整程度也有所不同，具体取决于应用程序的要求。不过，每个阶段越完整，应用程序的韧性也就越大。

您应将韧性生命周期视为公司可运行的标准流程。在对韧性生命周期建模时，亚马逊云科技有意使其类似于软件开发生命周期（SDLC），目的是在开发和运行应用程序时，可将规划、测试和学习融入整个运行流程。与许多敏捷开发流程一样，韧性生命周期会随着开发过程的迭代而重复出现。我们建议您在生命周期的各个阶段逐步深化实践。

第 1 阶段：设定目标

了解需要什么级别的韧性以及如何进行衡量是目标设定阶段的基础。如果你没有目标或无法对其进行衡量，那么改进将难以实现。

并非所有的应用程序都需要相同级别的韧性。在设定目标时，请考虑所需的韧性水平，以便做出正确的投资和权衡。在这里，汽车是一个很好的类比：汽车有四个轮胎，但只有一个备用轮胎。这是因为，在单次行驶中出现多个轮胎漏气的几率很低，并且配备更多备用轮胎可能会影响汽车的其他功能，比如货物空间或燃油效率等等。所以，这种配置是合理权衡的结果。

目标设定后，您需要在后续阶段（第 2 阶段：设计和实施（第 7 页）和第 4 阶段：运行（第 15 页））进行观测和控制，以了解是否达到目标。

确定关键应用

设定韧性目标不应仅限于技术对话。相反，您需要从业务目标入手，去了解应用程序需提供的服务以及受损的后果。然后，再将对业务目标的这种理解扩展到架构、工程和运行等领域。设定的任何韧性目标都可能适用于所有的应用程序，不过衡量目标的方式通常会因应用程序的功能而异。您当前运行的某个应用程序可能对业务至关重要。如果该应用程序受损，贵公司可能会损失大量收入或遭到名誉损害。也有可能的是，您当前运行的某个应用程序可能对业务没那么重要，因此公司可以容忍一段故障停机时间，同时也不会对业务能力产生负面影响。

以一家零售公司的订单管理应用程序为例。如果该订单管理应用程序的组件受损并且不能正常运行，那么新的销售将无法进行。此外，这家零售公司在一处办公楼内设有员工咖啡厅。咖啡厅提供线上菜单，员工可以通过静态网页进行访问。如果该网页不可访问，那么一些员工可能会有怨言，但未必会给公司带来经济损失。基于这个例子，企业可能会选择为订单管理应用程序设定更加积极的韧性目标，但不会进行大量投资来确保网页应用的韧性。

衡量应用程序在生产中的韧性十分重要，确定最关键的应用程序、需投入非常多精力的领域以及需权衡的事项同样重要。为了更好地了解应用受损的影响，您可以进行业务影响分析（BIA）。这种分析提供了一种结构化、系统化的方法，可以帮助您确定关键业务应用程序及其优先级别、评估潜在风险和影响并识别相互依赖关系。业务影响分析有助于量化贵公司重要应用程序的停机成本，即当特定应用程序受损而无法完成功能时的成本。在前面的例子中，如果订单管理应用程序受损，那么零售企业可能会损失大量收入。

确定用户故事

在业务影响分析过程中，您可能会发现某个应用程序可实现多项业务功能，或者某项业务功能需要多个应用程序的支撑。仍以前面的零售公司为例，订单管理功能可能需要结账、促销和定价等多个应用程序共同实现。如果其中的一个应用程序出现故障，那么公司及其用户都会受到影响。例如，公司可能会无法添加新订单、提供促销和折扣或者更新产品价格。订单管理所需的这些不同功能可能依赖于多个应用程序，还可能依赖多个外部应用程序，这样，以组件为核心来实现韧性的过程将变得十分复杂。应对这种情境的更好方法是以用户故事为核心，即概括用户在与一个

或一组应用程序互动时期望获得的体验。

关注用户故事可以帮助您了解哪些客户体验尤为重要，这样，您就可以构建相应的机制来防范特定的威胁。在前面的例子中，用户故事可以是结账，它受结账应用程序所支撑，并且依赖定价应用程序。用户故事也可以是查看促销，这涉及到促销应用程序。在确定了关键应用程序及用户故事之后，您便可以开始设定用来衡量用户故事韧性的指标。这些指标可以应用于整个故事组合，也可以应用于单个用户故事。

设定衡量指标

恢复点目标（RPO）、恢复时间目标（RTO）和服务级别目标（SLO）是用来评估既定系统韧性的标准行业指标。恢复点指标是指在发生故障时，企业可以容忍的数据丢失量，而恢复时间目标衡量的是应用程序在停机后恢复可用的速度。两个指标均以时间为单位，包括秒、分钟和小时。您也可以用它们来衡量应用程序正常工作的时间，这里的正常工作是指应用程序执行设计功能，并且能够为用户所访问。服务级别目标详述了客户将获得的预期服务级别，衡量指标包括在不到一秒的响应时间内获得无差错服务的请求的百分比（%）（例如，每月有 99.99% 的请求可得到响应）等等。恢复点目标和恢复时间目标与均灾难恢复策略相关，假设前提是从恢复备份到重定向用户流量的应用程序运行和恢复过程会出现中断。服务级别目标通过实施高可用性控制来实现，有助于减少应用程序的停机时间。

服务级别目标指标一般用来定义服务级别协议（SLA），后者是指在服务提供商和最终用户之间签订的合同。服务级别协议通常附带经济方面的承诺，列出了服务提供商在不遵循协议的情况下将需向用户支付的罚金。不过，服务级别协议并不是公司韧性的衡量指标，因此增加服务级别协议并不会让应用程序变得更具韧性。

现在，您可以开始根据服务级别目标、恢复点目标和恢复时间目标来设置您的韧性目标。在设定韧性目标并清楚了解恢复点目标和恢复时间目标之后，您可以利用 Amazon Resilience Hub 对应用程序架构进行评估，以便发现与韧性相关的潜在弱点。Amazon Resilience Hub 将根据 Amazon Well-Architected Framework 最佳实践来评估应用程序架构，并就需要改进的具体方面给出补救指导，以帮助您实现恢复点目标和恢复时间目标。

创建额外的衡量指标

恢复点目标、恢复时间目标和服务级别目标是衡量韧性的良好指标，不过，您也可以从业务角度考虑总体目标，并围绕应用程序的功能设定具体目标。例如，您的目标可以是：即便公司前台和后台之间的时延增加 40%，每分钟的成功订单量也可保持在 98% 以上；或者是：即便某个组件丢失，每秒流量也将保持在平均值的标准偏差范围之内。您也可以创建目标，以缩短各种已知故障类型的平均恢复时间（MTTR）。例如，如果发生某种已知问题，恢复时间将缩短 x%。创建符合业务需求的目标可帮助您预测应用程序可容忍的故障类型，还可以帮助您确定降低应用程序受损可能性的方法。

如果在丢失 5% 应用实例的情况下设定继续运行的目标，您可能会决定对应用进行预扩展或者让应用具有快速扩展的能力，以支持该事件期间产生的额外流量。或者，您也可能会决定采用第 2 阶段：设计和实施（第 7 页）一节中所描述的不同架构模式。

您还应该针对特定业务目标实施可观测性指标。例如，您可以跟踪平均订单率、平均订单价格、平均订购数量或其他指标，这些指标可以根据应用程序的行为提供对业务健康状况的洞察。基于应用程序的可观测性功能，您可以创建警报，并在这些指标超出您定义的界限时采取措施。可观测性在第 4 阶段：运行（第 15 页）一节中有更详细的介绍。

第 2 阶段：设计和实施

在前一阶段，您设定了韧性目标。现在，进入设计和实施阶段，您将尝试预测故障模式，并根据您在前一阶段设定的目标选择设计方案。您还需要制定变更管理策略、开发软件代码并配置基础设施。以下章节内容突出了在权衡成本、复杂性和运行费用等因素时应考虑的亚马逊云科技最佳实践。

Amazon Well-Architected Framework

在根据期望的韧性目标构建应用程序时，您需要评估多项因素，并基于最优架构进行权衡。为了构建高韧性的应用程序，您必须综合考虑设计、构建、部署、安全和运行等多个方面。Amazon Well-Architected Framework 提供了一整套最佳实践、设计原则和架构模式，可帮助您在亚马逊云科技平台上设计高韧性的应用程序。Amazon Well-Architected Framework 的六大支柱提供了设计和运行安全、高效、经济、可持续和高韧性系统的最佳实践。该框架提供了一种方法，可以帮助您参照最佳实践连贯地衡量应用程序架构，并确定需要改进的地方。

以下是 Amazon Well-Architected Framework 如何帮助您设计和实施符合韧性目标的应用程序的示例：

- **可靠性支柱：**可靠性支柱强调了构建即便在故障或中断期间也能正确、连贯运行的应用程序的重要性。例如，Amazon Well-Architected Framework 建议采用微服务架构，使应用程序变得更小、更简单，这样您就可以区分应用程序中不同组件的可用性需求。您还可以找到最佳实践的详细描述，通过使用节流、指数回退重试、快速故障（减载）、等幂、恒定工作、断路器和静态稳定性来构建应用程序。
- **全面检查：**Amazon Well-Architected Framework 鼓励参照最佳实践和设计原则对架构进行全面检查。该框架提供了一种方法，可以帮助您参照最佳实践连贯地衡量应用程序架构，并确定需要改进的地方。
- **风险管理：**Amazon Well-Architected Framework 可帮助您识别和管理可能影响应用程序可靠性的风险。通过主动解决潜在的故障，您可以降低故障发生的可能性或由此造成的损害。
- **持续改进：**韧性是一个持续的过程，因此 Amazon Well-Architected Framework 强调持续优化。您可以根据 Amazon Well-Architected Framework 的指导定期检查并改进应用程序架构和流程，这样就可以确保系统在面对不断发展的挑战和需求时始终保持韧性。

理解依赖关系

理解系统的依赖关系是实现韧性的关键。依赖关系包括应用程序内部各组件之间的关联，以及应用程序与外部组件（比如第三方应用程序编程接口和企业自有的共享服务）之间的关联。理解这些关联有助于隔离和管理中断，因为一个组件受损可能会影响其他组件的运行。这些知识可帮助工程师评估损害的影响并相应地制定计划，从而确保资源得到有效利用。理解依赖关系可以帮助您制定备用策略、协调恢复过程。它还可以帮助您确定可以用软依赖替换硬依赖的情况，这样，

当存在依赖损害时，应用程序可以继续执行其业务功能。依赖性还会影响关于负载平衡和应用程序扩展的决策。在更改应用程序时，理解依赖关系是至关重要的，因为这有助于您确定潜在的风险和影响。这些知识可以帮助您创建稳定、有韧性的应用程序，协助故障管理、影响评估、损害恢复、负载平衡、扩展和变更管理。您可以手动或者使用 Amazon X-Ray 等工具和服务来跟踪依赖关系，进而理解分布式应用程序的依赖关系。

灾难恢复策略

灾难恢复（DR）策略在构建和运行韧性应用程序的过程中扮演着关键角色，主要的作用是确保业务的连续性。即便在灾难性事件中，灾难恢复策略也能保证关键业务以最小的受损程度持续运行，从而最大限度地减少了停机时间和潜在的收入损失。灾难恢复策略对于数据保护至关重要，因为它们常常涉及多地的定期数据备份和数据复制，这有助于保护有价值的业务信息，同时防止灾难期间数据的全部丢失。而且，许多行业都受到相关政策的监管，这些政策要求企业制定灾难恢复策略来保护敏感数据，并确保服务在灾难期间保持可用。通过最大程度地减少服务的受损程度，灾难恢复策略还提高了客户的信任度和满意度。一项实施良好且屡屡实践的灾难恢复策略可以缩短灾难后的恢复时间，并可以帮助应用程序快速地重新上线。此外，灾难会带来巨大的成本，这不仅包括停机造成的收入损失，还包括恢复应用程序和数据的费用。设计良好的灾难恢复策略有助于减少这些经济损失。

策略的选择取决于您对应用程序的特定需求、您设定的恢复时间目标和恢复点目标以及您的预算。Amazon Elastic Disaster Recovery 是一种定制化的韧性服务，可用于实施本地和云上应用程序的灾难恢复策略。

有关更多信息，请参见亚马逊云科技网站上的 Disaster Recovery of Workloads on Amazon Web Services 和 Amazon Multi-Region Fundamentals。

制定持续集成和持续交付（CI/CD）策略

导致应用程序受损的一个常见原因是代码或其他变更改变了应用程序之前的已知工作状态。如果变更管理不够细致，那么这种变更可能会导致应用程序频繁受损。变更频率越高，产生影响的可能性也就越大。不过，降低变更的频率会产生更大的变更集合，由于其高度复杂，因此更有可能使应用程序受损。持续集成和持续交付（CI/CD）实践旨在保持小规模和频繁的变更（从而提高生产力），同时通过自动化对每项变更进行高级别的检查。一些基本策略包括：

- **完全自动化：**持续集成和持续交付的根本目的是尽可能实现应用构建和部署流程的自动化。整个流程包括应用程序的构建、测试、部署甚至监控。自动化管线有助于降低人为错误的可能性，确保一致性，并使流程更加可靠和高效。
- **测试驱动开发（TDD）：**在编写应用程序代码之前编写测试。这样可以确保所有的代码都有相关的测试，因此提高了代码的可靠性和自动化检查的质量。这些测试在持续集成管线中运行，以便验证变更。
- **频繁提交和集成：**鼓励开发人员频繁提交代码并进行集成。小规模且频繁的更改更容易测试和调试，进而降低了出现重大问题的风险。自动化降低了提交和部署的成本，使得频繁集成成为可能。

- **不可变基础设施：**将服务器和其他基础设施组件视为静态的不可变实体。尽可能替换而不是修改基础设施；利用经测试的代码构建新的基础设施，并通过管线进行部署。
- **回滚机制：**如果出现问题，确保能有一个简单、可靠且经频繁测试的更改回滚方式。能够快速返回到先前已知的良好状态对于部署安全至关重要。回滚可以由一个简单的按钮实现，轻轻一按便可以恢复到以前的状态，或者也可以完全自动化，警报一响便可启动。
- **版本控制：**将所有应用程序代码、配置甚至基础设施即代码保存在受版本控制的存储库中。这可以帮助您轻松地跟踪更改，并在需要时恢复更改。
- **金丝雀部署和蓝/绿部署：**首先将应用程序的新版本部署于基础架构的一个子集，或者维持蓝/绿两种环境，这可以帮助您在生产中验证更改行为，并在必要时快速回滚。

持续集成和持续交付不仅是工具，也构成了一种文化。创造一种重视自动化、测试和从失败中学习的文化与实施正确的工具和过程同样重要。如果回滚速度很快且影响极小，那么这种回滚不应被视为是一种失败的经历，而是一种学习的体验。

开展运营就绪审查

运营就绪性审查（ORR）有助于确定运营和和程序上的差距。在亚马逊云科技，我们确立了运营就绪审查制度，从数十年大规模服务运营中提取经验，精选问题，以提供最佳实践指导。运营就绪性审查总结了以前的经验教训，并要求新团队在应用中考虑了这些经验教训。运营就绪性审查可以提供一个故障模式或故障原因的清单，在下面韧性建模一节描述的韧性建模活动中，可以输入这些故障模式或故障原因。有关更多信息，请参见 Amazon Well-Architected Framework 网站上的运行就绪检查（ORRs）。

了解亚马逊云科技故障隔离边界

亚马逊云科技提供多种故障隔离边界，以帮助您实现韧性目标。您可以充分利用这些边界提供的可预测的影响遏制范围，也应该熟悉如何利用这些边界来设计亚马逊云科技服务，这样就可以有意识地为应用程序选择依赖项。有关如何在应用程序中利用故障隔离边界，请参阅亚马逊云科技网站上的 Amazon Fault Isolation Boundaries。

选择响应方式

系统可以以多种方式对警报做出响应。某些警报可能需要运营团队的响应，而某些警报可能会触发应用程序中的自我修复机制。您可能决定将可以自动化的响应保留为手动操作，以控制自动化的成本或管理工程约束。警报响应类型可被选做响应实施成本、警报预期频率、警报准确性以及根本不对警报做出响应而造成的潜在商业损失的函数。

例如，当服务器进程崩溃时，操作系统可能会重新启动该进程，或者会有新的服务器代替旧的服务器，或者可能会指示操作员远程连接并重启服务器。这些响应结果相同，即都是重启应用服务器进程，但是实施和维护成本有所不同。

注意

您可以选择多种响应方式，以便构建深入的韧性方法。例如，在前面的场景中，应用团队可能选择实施所有三种响应方式，不过在各种响应方式之间有一定的时间延迟。如果服务器进程崩溃指示器在 30 秒后仍然处于报警状态，那么团队可以认为操作系统无法重启应用服务器。因此，他们可能会创建一个自动扩展组，设置一个新的虚拟服务器来恢复应用服务器进程。如果指示器在 300 秒后仍处于报警状态，则可能会向操作人员发送警报，要求他们连接到原始服务器并尝试恢复进程。

应用程序团队和企业选择的响应方式应反映出企业希望通过前期工程时间的投入来抵消运营开支的偏好。您应该仔细考虑每种响应选择的约束和预期维护成本，进而选择恰当的响应方式，比如静态稳定性之类的架构模式、断路器之类的软件模式或是操作程序等等。对于应用程序团队来说，可能存在一些标准的相应方式指导，因此，您可以输入集中式架构功能管理的数据库和模式，然后做出相应考虑。

韧性建模

韧性建模记录了应用程序响应各种预期中断的过程。在预测中断的基础上，您的团队可以实现应用的可观测性、自动化控制和故障恢复，从而减轻或阻止中断造成的损害。亚马逊云科技利用韧性分析框架为韧性模型的开发提供指导。该框架可以帮助您预测中断及其可能对应用程序产生的影响。在预测中断的基础上，您可以确定构建可靠、高韧性的应用程序所需的中断缓解措施。我们建议您采用韧性分析框架，以便在应用程序迭代时更新韧性模型。在每次迭代中使用韧性分析框架——也就是在设计阶段预测中断并在生产部署前后测试应用程序——有助于减少事故发生；利用韧性分析框架开发韧性模型也有助于实现韧性目标。

安全的故障

如果你无法避免中断，那就确保故障是安全的。考虑利用默认的故障安全操作模式来创建应用程序，这样就不会导致重大的业务损失。数据库故障安全状态的一个例子是默认为只读操作，即不允许用户创建或修改任何数据。针对敏感性数据，您甚至可能希望应用程序默认为关闭状态，甚至不执行只读查询。请考虑应用程序的故障安全状态应该是什么，并在极端条件下默认使用这种操作模式。

第 3 阶段：评估和测试

在应用程序生命周期的评估和测试阶段，您已完成应用程序的设计或更改，但尚未发布于生产环境中。在这个阶段，您将开展一些活动来测试之前阶段的实践，并对结果进行评估。应用程序可能仍在积极开发中，或者主要开发可能已经完成，应用程序正在接受发布到生产环境之前的测试。在这个阶段，您将重点关注开发和运行测试，这些测试将确认应用程序能否达到设定的韧性目标。此外，您还要开发和测试系统的操作程序。您将应用在第 2 阶段：设计和实施（第 7 页）中开发的部署程序，并对结果进行评估。虽然这些测试和评估活动在这一阶段开始，不过并非在此结束。在第 4 阶段：运行阶段（第 15 页）中，测试和评估将继续进行。

评估和测试阶段进一步分为两个阶段，即部署前活动（第 11 页）和部署后活动（第 12 页）。部署前活动包括在将应用程序部署于任何环境之前应该完成的各种任务，比如部署软件的新版本以及在测试环境中的初始部署。部署后活动在将软件部署于测试或生产环境之后进行。以下章节将对这些阶段进行详细讨论。

部署前活动

环境设计

测试和评估应用程序的环境会影响测试的彻底程度，也会影响您对测试结果准确反映生产环境的信心。借助 Amazon DynamoDB（参见 DynamoDB 文档中的本地设置 DynamoDB）之类的服务，您可以在开发人员的机器上本地执行一些集成测试。但是，在某些时候，为了对结果有更强的信心，您需要在模拟生产环境中进行测试。这种环境会产生更高的成本，因此，我们建议您就测试环境采用分阶段或管线方法，模拟生产环境将出现在管线的后期阶段。

集成测试

集成测试是测试应用程序中定义明确的组件在使用外部依赖项时能否正确执行其功能的过程。这些外部依赖项可以是其他定制开发的组件、应用程序使用的亚马逊云科技服务、第三方依赖项和本地依赖项。本指南侧重于针对应用程序韧性的集成测试，并且假设已经存在证明软件功能准确性的单元和集成测试。

我们建议您针对已实施的韧性模式（比如断路器模式或减载模式（参见第 2 阶段：设计和实现（第 7 页））设计专门的集成测试。针对韧性的集成测试通常需要向应用程序施加特定的负载，或是利用诸如 Amazon Fault Injection Simulator （Amazon FIS）之类的功能，有意地在测试环境中制造中断场景。理想状态是，您需将所有的集成测试作为持续集成/持续交付管线的一部分，并确保每次提交代码时都运行测试。这可以帮助您快速检测违背韧性目标的任何代码或配置变更并做出响应。大规模的分布式应用程序十分复杂，即便是微小的变化也会对应用程序中看似不相关的组件的韧性产生重大影响。尽量在每次提交代码时都运行测试。亚马逊云科技提供了一整套出色的工具，可帮助您运行持续集成/持续交付管线和其他 DevOps 工具。有关更多信息，请参见亚马逊云科技网站上的“亚马逊云科技平台上 DevOps 简介”。

自动化部署管线

在生产前环境中进行部署和测试是一项反复而复杂的任务，还是能够自动化完成得好。自动化可以解放人力资源，减少出错的几率。实现流程自动化的机制通常被称为管线。在创建管线时，我们建议您设置一系列逐步接近生产配置的测试环境。您可以利用这一系列环境来反复测试应用程序。与生产环境相比，第一种环境提供的功能比较有限，但成本却低得多。后续环境应该不断添加服务并进行扩展，以更好地反映生产环境。

首先，在第一个环境中进行测试。如果部署的应用程序通过了第一个测试环境中的所有测试，那么让它在一定的负载下运行一段时间，看看随着时间的推移是否会出现任何问题。请确认已经正确地配置了可观测性（请参阅本指南后面的“报警精度”），这样您就可以检测到出现的任何问题。如果应用程序成功地通过了这段观测期，则可以将其部署到下一个测试环境中，然后重复这个过程，添加环境支持的额外测试或负载。在以这种方式充分测试应用程序之后，您便可以使用之前设置的部署方法将应用程序部署到生产环境中（请参阅本指南前面的“制定持续集成/持续交付策略”）。Amazon Builders' Library 中的《自动化安全、无需干预的部署》一文是很好的资源，描述了亚马逊云科技如何实现代码部署的自动化。生产部署之前的环境数量会有所不同，具体取决于应用程序的复杂程度及其依赖项的类型。

压力测试

从表面上看，压力测试类似于集成测试。为了验证应用程序及其外部依赖项是否按预期运行，您会对其离散性进行测试。不过，测试又超越了集成测试，关注的是应用程序如何在明确的负载下运行。压力测试需要验证功能的正确性，因此必须在成功的集成测试之后进行。了解应用程序在预期负载下的响应情况以及在负载超出预期时的行为非常重要。这有助于验证是否已经实施了必要的机制来确保应用程序在极端负载下的韧性。有关亚马逊云科技压力测试的全面指南，请参见 Amazon Solutions Library 中“亚马逊云科技平台上的分布式负载测试“。

部署后活动

韧性是一个持续的过程，因此，部署应用程序之后，必须继续评估应用程序的韧性。根据部署后活动（比如持续的韧性评估）的结果，您可能需要重新评估并更新在韧性生命周期早期阶段执行的一些活动。

开展韧性评估

在将应用程序部署到生产环境中之后，韧性评估并没有结束。在真实的生产环境中，即便您有定义良好的自动化部署管线，变更也时有发生。此外，可能还存在一些您在部署前韧性验证中尚未考虑的因素。Amazon Resilience Hub 中有一处核心区域，您可以在此评估已部署的架构能否达成设定的恢复点目标和恢复时间目标。您也可以利用这项服务，根据需求对应用程序的韧性进行自动化评估，甚至将它们集成到持续集成/持续交付工具中，正如亚马逊云科技博客文章《使用 Amazon Resilience Hub 和 Amazon CodePipeline 连续评估应用程序韧性》所述。自动化评估是最佳实践，因为它有助于确保您在生产中持续评估韧性状态。

灾难恢复测试

在第 2 阶段：设计和实施（第 7 页）中，您已制定了灾难恢复（DR）策略，并将其作为系统的一部分。在第 4 阶段，您需要测试灾难恢复程序，以确保团队为灾难事件做好充分准备，同时确保程序按预期工作。您应定期测试所有的灾难恢复程序，包括故障转移和故障恢复，并查看每次测试的结果，以确定是否需要更新以及如何更新系统程序，从而获得尽可能很好的结果。当启动灾难恢复测试开发时，请提前做好规划，同时确保整个团队都知晓预期结果，而且知道如何衡量结果以及应采用何种反馈机制来更新程序。在熟练运行有计划的灾难恢复测试之后，可以考虑运行突击的灾难恢复测试。真正的灾难是不会按计划发生的，所以需要做好随时演练的准备。不过，突击不代表没有计划。关键利益相关方仍需要就灾难事件做好规划，以确保监控到位，并且客户和关键应用程序都不会受到不利影响。

偏差检测

即使自动化和明确定义的程序已经就位，生产应用程序中的配置也可能会发生意外变化。为了检测应用程序配置的变化，您需确立偏差检测机制，这里的“偏差”是指与基线配置的偏差。关于如何检测 Amazon CloudFormation 堆栈中的偏差，请参阅 Amazon CloudFormation 文档中的“检测堆栈和资源的非托管配置更改”。关于如何检测应用程序在亚马逊云科技环境中的偏差，请参阅 Amazon Control Tower 文档中的“检测和解决 Amazon Control Tower 中的偏差”。

合成测试

合成测试是创建可配置软件的流程，该软件按计划在生产中运行，以模拟最终用户体验的方式测试应用程序的编程接口。这些测试有时被称为“金丝雀”，该术语最早是在煤矿开采中使用。当应用程序遭受中断时，即便损害是局部或间歇性的（灰色故障通常就是这种情况），合成测试也可以提供早期警告。

混沌工程

混沌工程是一个系统化的流程，包括以降低风险的方式故意使应用程序遭受破坏性事件，然后密切监视其响应，并实施必要的改进。其目的是验证或质疑应用程序应对此类中断的能力假设。混沌工程不是任这些事件随意发展，而是让工程师能够在一个受控的环境中安排实验；实验通常安排在低流量期间，并且随时可以获得有效的工程支持。

混沌工程的第一步是理解应用程序的正常操作条件，即所谓的稳态。在此基础上，您需要做出一项假设，详细描述应用程序在出现中断时的成功行为。然后，您开始运行实验，需要故意引入中断，包括但不限于网络延迟、服务器故障、硬盘错误和外部依赖项受损等等。最后，您将对实验结果进行分析，并通过学习增强应用程序的韧性。混沌工程实验是一项颇有价值的工具，可以帮助您改进应用程序的性能及其他各个方面，并且发现潜在的问题。此外，混沌工程有助于揭示和完善可观测性和警报工具的缺陷，缩短恢复时间，提高运行技能。混沌工程还可加速最佳实践的采用，培养持续改进的心态。最终，它使团队能够通过定期的重复实践来构建和提升运行技能。

亚马逊云科技建议您在非生产环境中开始混沌工程实验。您可以利用 Amazon Fault Injection Simulator （Amazon FIS） 来运行混沌工程，处理通用故障以及亚马逊云科技特有的故障。这项完全托管的服务包括停止条件警报和完全许可控制，因此您可以安全、放心、轻松地开展混沌工程实验。

第 4 阶段：运营

完成第 3 阶段：评估和测试（第 11 页）后，就可以将应用程序部署到生产环境中了。在*运营*阶段，需要将应用程序部署到生产中，并管理客户体验。应用程序的设计和实施决定着其自身的许多韧性结果，但这一阶段侧重于系统用于维护和提高弹性的操作实践。搭建卓越运营文化有助于为这些实践制定标准和一致性。

可观察性

监控和警报是了解客户体验的首要方式。您需要对应用程序进行检测，以了解其状态，而且需要从不同的角度进行检测，这意味着需要从服务器侧和客户侧进行测量 - 通常使用金雀丝。测量指标应包括应用程序的交互数据及其依赖关系，以及与故障隔离界限相一致的范围。此外，还应该生成日志，以便提供有关应用程序所执行的每个工作单元的更多详细信息。您可以考虑使用诸如 Amazon CloudWatch 嵌入式指标格式 之类的解决方案，将指标和日志结合起来。您可能会发现，您总是希望获得更多的可观察性，因此请考虑实现预期检测水平所需要的成本、工作量和复杂性折中。

以下链接提供了检测应用程序和创建警报的最佳实践：

- 监控 Amazon 的生产服务（Amazon Web Services re:Invent 2020 演示）
- Amazon Builders' Library：Amazon 的卓越运营（Amazon Web Services re:Invent 2021 演示）
- Amazon 的可观察性最佳实践（Amazon Web Services re:Invent 2022 演示）
- 对分布式系统进行检测以实现运营可视性 （Amazon Builders' Library 文章）
- 构建仪表板以实现运营可视性（Amazon Builders' Library 文章）

事件管理

为了在警报（或者更糟糕的是，您的客户）提示出现问题时处理受损情况，应该制定事件管理流程。该流程应包括聘用值班操作人员、上报相关问题、以及为一致的故障排除方法编写运行手册，帮助消除人为错误。但是，通常不会孤立发生损坏；单个应用程序可能会影响依赖于其的其他多个应用程序。通过了解受影响的所有应用程序，并经由一次电话会议将多个团队的操作人员召集起来，可迅速解决问题。不过，根据企业的规模和架构，这一过程可能需要一个集中的运营团队。

除了建立事件管理流程外，您还应通过仪表板定期审查指标。定期审查有助于了解客户体验和应用程序性能的长期趋势。这可帮助您识别问题和瓶颈，以免对生产造成重大影响。以一致、标准化的方式审查指标，可实现显著效益，但需要自上而下的支持和时间投入。

以下链接提供了关于建立仪表板和运营指标审核的最佳实践：

- 构建仪表板以实现运营可视性 （Amazon Builders' Library 文章）
- Amazon 成功应对故障的方法（Amazon Web Services re:Invent 2019 演示）

持续韧性

在第 2 阶段：设计和实施（第 7 页）和第 3 阶段：评估和测试（第 11 页）期间，您在将应用程序部署到生产环境之前启动了审查和测试活动。在*运营*阶段，您应继续迭代生产中的这些活动。您应通过 Amazon Well-Architected Framework reviews、Operational Readiness Reviews (ORRs)、以及韧性分析框架，定期审查应用程序的韧性状态。这有助于确保您的应用程序不偏离既定的基线和标准，并让您及时了解新的或更新后的指南。这些持续的韧性活动有助于让您发现以前未曾预料到的中断，并帮助您找到新的化解措施。

在预生产环境中成功运行 game day 和混沌工程实验后，您可能还会考虑在生产环境中运行这些实验。game day 用于模拟已知事件，而针对这些事件，您已经建立起加以缓解的韧性机制。例如，game day 可能会模拟亚马逊云科技区域服务受损并实施多区域故障转移。虽然实施这些活动可能需要大量努力，但这两种实践都有助于让您相信您的系统能够承受您所设计的故障模式。

在运营应用程序、遭遇运营事件、审查指标和测试应用程序过程中，您将遇到加以应对和学习的多种机会。

第 5 阶段：响应和学习

应用程序应对破坏性事件的方式会影响到其可靠性。吸取经验教训，了解应用程序在过去如何应对破坏性事件，这对提高可靠性也至关重要。

*响应和学习*阶段侧重于实践，您可以通过实施这些实践而更好地响应应用程序中的破坏性事件。其中还包括帮助您从运营团队和工程师的经验中提炼出极多知识和教训的实践。

创建事件分析报告

事件发生后，首要任务是尽快防止对客户和业务造成进一步损害。应用程序恢复后，下一步是了解发生了什么，并确定防止再次发生的措施。这种事后分析通常会编写成报告，记录导致应用程序受到破坏的一系列事件，以及中断对应用程序、客户和业务造成的影响。此类报告将成为宝贵的学习成果，并应在业务部门间广泛共享。

注

在进行事件分析时，至关重要的是不要指责任何一方。要假设所有操作人员都根据所掌握的信息优选非常适合的行动方案。请勿在报告中使用操作人员或工程师的姓名。将人为错误当作造成损害的原因，可能会导致团队成员为了自我保护而心怀戒备，从而导致获取的信息有误或不完整。

出色的事件分析报告 - 例如 Amazon Correction of Error（COE）程序中记录的报告 - 应采用标准化格式，并尽可能详细地记录导致应用程序受到破坏的各种状况。该报告详述一系列带有时间戳的事件，并收集定量数据（通常是来自监控仪表板的指标和屏幕截图），这些数据描述了应用程序随时间而变化的可测量状态。该报告应记录操作人员和工程师采取行动的思维过程，以及导致他们得出结论的信息。该报告还应详细说明不同指标的性能 - 例如，发出了哪些警报、这些警报是否准确地反映了应用程序的状态、事件与由此产生的警报之间的时间间隔、以及解决事件的时间。时间轴还记录所启动的运营手册或自动操作，以及它们说如何帮助应用程序恢复有用状态的。时间轴上的这些要素有助于您的团队了解自动化响应和操作人员响应的有效性，包括他们解决问题的速度以及在化解破坏方面的有效程度。

这种详细的历史事件描述是一种强大的教育工具。团队应将这些报告存放到中央存储库中，供整个企业查阅，以便其他人可以查看相关事件并从中学习。针对生产环境中可能会出现的问题，这可以提高团队的直觉。

详细事件报告库还可以成为操作人员的培训材料来源。团队可以使用事故报告为桌面或现场 game day 提供灵感，在这种情况下，将为团队提供报告中按时间轴再现的信息。操作人员可以利用时间轴中的部分信息进行情景排练，并描述他们将会采取的行动。然后，针对应用程序将会如何响应，game day 的主持人可以根据操作人员的行动而提供指导。这可以培养操作人员的故障排除技能，令其更轻松地预测和排解相关问题。

负责应用程序可靠性的中央团队应将这些报告保存在可供整个企业查阅的中心库中。该团队还应负责维护报告模板，并培训团队如何完成事件分析报告。可靠性团队应定期审查报告，以便发现整个业务中可通过软件库、架构模式或团队流程变更而加以解决的趋势。

实施运营审查

如第 4 阶段：运营（第 15 页）所述，运营审查是审查近期功能特性发布、事件和运营指标的机会。另外，还可借运营审查之机，与企业内更广泛的工程社区分享从功能发布和事件中所获得的经验教训。在运营审查期间，各团队会审查回滚的功能特性部署、所发生的事件以及处理方式。这为整个企业内的工程师提供了学习他人经验和提出问题的机会。

向公司内的工程社区分享运营审查结果，让他们更多地了解运行业务的 IT 应用程序以及可能会遇到的问题类型。在他们为业务设计、实施和部署其他应用程序时，将也记得这些知识。

审查警报性能

如*运营*阶段所述，警报可能会引发仪表板告警、创建 Ticket、发送电子邮件或呼叫操作人员。一个应用程序会设置许多警报，用以监控其运营的各个方面。随着时间推移，应该对这些警报的准确性和有效性进行审查，提高警报精确度，降低假阳性并合并重复的警报。

警报精确度

警报应尽可能明确，以减少解读或诊断引起警报的具体破坏所需的时间。当因响应应用受损而触发警报时，接收和响应警报的操作人员必须首先解读警报所传递的信息。这些信息可能是一个与操作步骤（例如恢复程序）相对应的简单错误代码，也可能包括应用程序日志中的多行内容，您必须查看日志才能了解警报触发原因。当您的团队学会更有效地运行应用程序时，他们应该改进这些警报，使其尽可能清晰且简洁。

您不可能预料到应用程序可能出现的所有受损情况，因此总会出现一些需要操作人员分析和诊断的一般警报。为了缩短响应时间和平均修复时间（MTTR），您的团队应努力减少一般警报的数量。理想情况下，警报与自动或人工响应之间应该是一一对应的关系。

假阳性

随着时间的推移，操作人员可能会忽略那些无需采取任何行动、但会以电子邮件、呼叫或 Ticket 形式发出告警的警报。定期或在事件分析的过程中审查警报，识别那些经常遭到忽略或无需操作人员采取任何措施的警报（*假阳性*）。为了向操作人员发出可执行的警报，您应努力删除警报或改进警报。

假阴性

在事件发生期间，如果某个事件以意想不到的方式影响到了应用程序，则设置为在事件发生时发出告警的警报可能会失效。在事件分析的过程中，您应该审查本应发出但实际上并未发出的警报。您应努力改进这些警报，以便其更好地反映事件可能引发的状况。或者，您可能需要创建更多警报，用于映射由不同的征兆引发的相同受损情况。

重复的警报

损害应用程序的破坏可能会引发多种征兆，并可能导致多个警报。应定期或在事件分析的过程中，查看所发出的警报和告警。如果操作人员收到重复警报，则应创建警报集合，将其合并为单个警报消息。

实施指标审查

您的团队应收集有关应用程序的运行指标，例如每月按严重程度分类的事件数量、探测事件所需要的时间、确定原因所需要的时间、补救时间以及创建的 Ticket、发送的警报和发出的呼叫数量。至少每月审查一次这些指标，借以了解操作人员的负担，他们处理的信噪比（例如，信息性警报与可执行警报），以及对于其控制下的应用程序，团队是否正在提高操作能力。通过这种审查，了解运营团队可量度方面的趋势。针对如何改进这些指标，征求团队的意见。

提供培训和赋能

引发事件或意外行为的应用程序及其环境难以详细描述。此外，对应用程序的韧性进行建模以预测此类情况并不总是简单明了。您的企业应投资于培训和赋能材料，以便运营团队和开发人员参与到韧性建模、事件分析、game day 和混沌工程实验等活动之中。这将提高团队所编制报告以及他们所获取知识的准确性。同时，团队的能力也能得到提升，从而在无需依赖经验丰富的工程师小组（工程师须通过定期审查而分享见解）的情况下预测故障。

创建事件知识库

事件报告是事件分析的标准输出结果。即使应用程序并未受损，您也应该使用相同或类似的报告，记录所检测到的应用程序异常行为情况。使用相同的标准化报告结构，记录混沌工程实验和 game day 结果。该报告代表着导致事件或其他意外行为的应用程序及其环境的快照。您应该将这些标准化报告存放在中心库中，供企业内的所有工程师查阅。

这样，运营团队和开发人员可以搜索该知识库，了解过去哪些情况曾导致应用程序受损，哪类状况可能会导致破坏，以及哪些情况可以防止应用程序受损。该知识库将成为提高运营团队和开发人员技能的加速器，使他们能够共享知识和经验。此外，您还可以将这些报告用作培训材料以及 game day 或混沌工程实验的场景，借以提高运营团队排解故障的直觉和能力。

注

标准化的报告格式还能给予读者一种熟悉感，有助于他们更快速地找到所需要的信息。

深入实施韧性

如前所述，先进的企业会对警报实施多种响应。单一一种响应无法保证有效，因此，通过逐级响应，应用程序可以更从容地应对故障。我们建议您为每个指标至少实施两种响应，确保单一响应不会成为可能导致灾难恢复场景的单点故障。这些响应层应按顺序创建，这样只有在前一种响应无效时才会执行后一种响应。不应对单个警报运行多个逐级响应。相反，应使用警报指示响应是否未取得成功，如果没有成功，则启动下一级响应。

结论和资源

通过在 5 个阶段实施最佳实践，本指南展示了一个有助于您不断提高应用程序韧性的生命周期：*设定目标*、*设计和实施*、*评估和测试*、*运营以及响应和学习*。

关于本指南中所述服务和概念的更多信息，请参见以下资源。

亚马逊云科技服务：

- Amazon Backup
- Amazon Elastic Disaster Recovery
- Amazon Fault Injection Simulator (Amazon FIS)
- Amazon Resilience Hub
- Amazon Route 53 Application Recovery Controller
- Amazon X-Ray

博文和文章：

- 可用性及其他：了解并提高基于亚马逊云科技的分布式系统韧性
- 亚马逊云科技故障隔离界限
- 亚马逊云科技多区域基础
- 云端混沌工程
- 利用 Amazon Resilience Hub 和 Amazon CodePipeline 持续评估应用程序韧性
- 将内部应用程序灾难恢复至亚马逊云科技
- 可靠性支柱 – Amazon Well-Architected Framework
- 韧性分析框架

撰稿人

本指南的撰稿人包括：

- Bruno Emer，亚马逊云科技首席解决方案架构师
- Clark Richey，亚马逊云科技首席解决方案架构师
- Elaine Harvey，亚马逊云科技可靠性服务总经理
- Jason Barto，亚马逊云科技首席解决方案架构师
- John Formento，亚马逊云科技首席解决方案架构师
- Lisi Lewis，亚马逊云科技高级产品营销经理
- Michael Haken，亚马逊云科技首席解决方案架构师
- Neeraj Kumar，亚马逊云科技首席解决方案架构师
- Wangechi Doble，亚马逊云科技首席解决方案架构师

文档历史

下表描述了本指南的重要变更。如果您想要接收关于未来更新的通知，请订阅 RSS 源。

变更	说明	日期
首次发布(第 23 页)	—	2023 年 10 月 6 日

亚马逊云科技

亚马逊云科技高可用性云服务
扫码免费试用，限时体验