AWS Well-Architected Framework

可靠性支柱

可靠性支柱: AWS Well-Architected Framework

Table of Contents

可靠性支柱 – AWS Well-Architected Framework

发布日期： 2024 年 6 月 27 日 (文档修订)

本白皮书重点介绍 AWS Well-Architected Framework的可靠性支柱。文中提供了指导，可帮助客户在 Amazon Web Services（AWS）环境的设计、交付和维护过程中应用最佳实践。

简介

此 AWS Well-Architected Framework 有助于您了解您在 AWS 上构建工作负载时所做决策的优缺点。通过使用此框架，您将了解有关在云中设计和运行可靠、安全、高效、经济实惠且可持续的工作负载的架构最佳实践。它提供了一种统一的方法，使您能够根据最佳实践衡量架构，并确定需要改进的方面。我们相信，拥有架构完善的工作负载能够大大提高实现业务成功的可能性。

AWS Well-Architected Framework 基于六大支柱：

- 卓越运营
- 安全性
- 可靠性
- 性能效率
- 成本优化
- 可持续性

本白皮书重点介绍了可靠性支柱，以及如何将其应用于您的解决方案。在传统本地环境中，由于单点故障、缺乏自动化和缺乏弹性，实现可靠性可能具有挑战性。通过采用本白皮书中的实践，您将会构建具有强大的基础、韧性的架构，一致的变更管理和经过验证的故障恢复流程的架构。

本白皮书面向技术人员，例如首席技术官（CTO）、架构师、开发人员和运维团队成员。阅读本白皮书后，您将了解可在设计云架构以实现可靠性时使用的 AWS 最佳实践和策略。本白皮书包括高层次实施详情和架构模式，以及对其他资源的引用。

可靠性

可靠性支柱涵盖相关工作负载按照计划正确而稳定执行其预期功能的能力。它包括在其全部生命周期内运行和测试工作负载的能力。本白皮书深度介绍了有关在 AWS 中实施可靠工作负载的最佳实践指导。

主题

弹性的责任共担模式

弹性是 AWS 和您共同承担的一项责任。您要了解作为弹性一部分的灾难恢复（DR）和可用性在这个共担模式下如何运行，这很重要。

AWS 责任 - 云的弹性

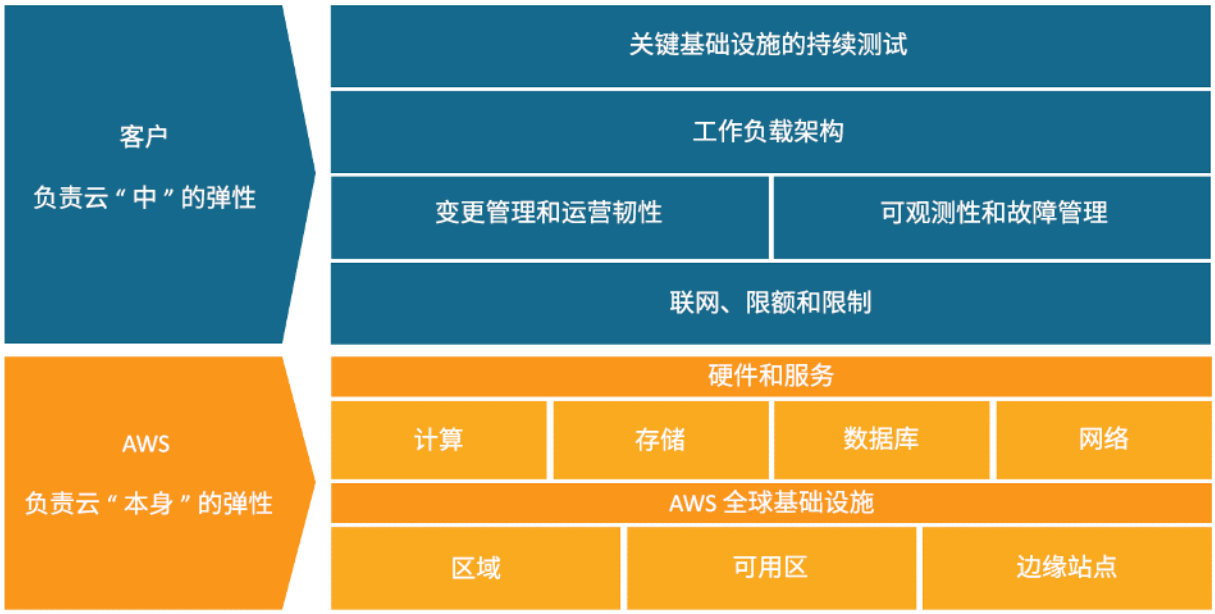
对于运行 AWS Cloud 中提供的所有服务的基础设施，AWS 负责维持其弹性。此基础设施包括运行 AWS Cloud 服务的硬件、软件、网络产品和设施。AWS 作出在商业上合理的努力，提供这些 AWS Cloud 服务，确保服务可用性符合或超过 AWS 服务等级协议（SLA）。

AWS 全球云基础设施旨在使客户可以构建高弹性工作负载架构。每个 AWS 区域完全隔离，包含多个可用区，可用区是在物理上隔离的基础设施分区。可用区隔离会影响工作负载弹性的故障，防止它们影响区域内的其他可用区。但同时，AWS 区域中的所有可用区都通过完全冗余的专用城域光纤，通过高带宽、低延迟的网络互联，从而在可用区之间实现高吞吐量、低延迟的联网。加密可用区之间的所有流量。网络性能足够，可以完成可用区之间的同步复制。跨可用区对应用程序进行分区时，公司可以更好地隔离并防止受到断电、雷击、龙卷风、飓风等灾害的影响。

客户责任 - 云中的弹性

您的责任由您选择的 AWS Cloud 服务决定。这决定了您承担弹性责任时必须执行的配置工作量。例如，Amazon Elastic Compute Cloud（Amazon EC2）等服务要求客户执行所有必要的弹性配置和管理任务。部署 Amazon EC2 实例的客户负责在多个地点部署 Amazon EC2 实例（例如 AWS 可用区）、使用 Auto Scaling 等服务实施自我修复，以及对实例上安装的应用程序使用弹性工作负载架构最佳实践。对于托管服务（例如 Amazon S3 和 Amazon DynamoDB），AWS 运行基础设施层、操作系统和平台，而客户访问端点以存储和检索数据。您负责管理数据的弹性，包括备份、版本控制和复制策略。

在 AWS 区域中的多个可用区部署工作负载是高可用性策略的一部分，通过将问题隔离在一个可用区，使用其他可用区的冗余来继续提供请求，旨在保护工作负载。多可用区架构也是 DR 策略的一部分，旨在更好地隔离工作负载并免受诸如停电、雷击、龙卷风、地震等问题的影响。DR 策略也可以使用多个 AWS 区域。例如，在主动/被动配置中，如果活动区域不再提供请求，则工作负载的服务从其活动区域失效转移到其灾难恢复区域。



客户和 AWS 对云中和云本身的弹性所承担的责任。

您可以使用 AWS 服务来实现弹性目标。作为客户，您负责管理系统的以下方面，以便实现云中的弹性。有关具体到每项服务的更多详细信息，请参阅 AWS 文档。

联网、限额和限制

- 基础部分下面详细说明了责任共担模式这一方面的最佳实践。
- 根据预期的负载请求增长情况（如果适用），为您的架构规划足够的空间来扩展和了解所包含服务的服务限额和限制。
- 将网络拓扑设计为高可用、冗余和可扩展。

变更管理和运营韧性

- 变更管理包括如何在环境中引入和管理变更。实施变更需要为应用程序和基础设施编制运行手册并使运行手册保持最新，还要制定部署策略。
- 监控工作负载资源的弹性策略考虑到所有组件，包括技术和业务指标、通知、自动化和分析。

- 云中的工作负载必须适应需求的变化，根据使用量的减损或波动进行扩展。

可观测性和故障管理

- 需要通过监控来观测故障，进而自动修复，以便工作负载在组件发生故障时仍能正常运行。
- 故障管理需要备份数据、应用最佳实践以使工作负载能够在组件发生故障时继续运行，以及为灾难恢复制定计划。

工作负载架构

- 工作负载架构包括如何根据业务领域来设计服务、应用 SOA 和分布式系统设计来防止故障，以及内置功能，如节流、重试、队列管理、超时和紧急杠杆。
- 依赖已被证明有效的 AWS 解决方案、Amazon Builders Library 和无服务器模式来遵循最佳实践并快速开始实施。
- 通过持续改进将系统分解为分布式服务，以便更快地扩展和创新。使用 AWS 微服务指导和托管服务选项来简化和加快引入变化和创新。

关键基础设施的持续测试

- 测试可靠性是指在功能、性能和混沌级别进行测试，以及采用事件分析和实际试用实践来构建专业知识，解决没有很好地理解的问题。
- 对于全云部署和混合应用程序，如果知道在出现问题或组件故障时应用程序会有怎样的表现，您就可以快速和可靠地从故障中恢复。
- 创建和记录可重复的试验，了解在事情并不像预期的那样发展时，您的系统会怎样表现。这些测试会证明您的整体弹性的有效性，并在面对真正的故障场景之前，为您的运营过程提供反馈循环。

设计原则

在云中，有许多原则可帮助您提高可靠性。在讨论最佳实践时，请记住以下几点：

- 自动从故障中恢复：通过监控工作负载的关键性能指标（KPI），您可以在指标超过阈值时触发自动化响应机制。这些 KPI 应该是对业务价值（而不是服务运营的技术方面）的一种度量。这包括自动发送故障通知和跟踪故障，以及启动解决或修复故障的自动恢复流程。借助更高级的自动化功能，您可以在故障发生之前预测和修复故障。

- 测试恢复过程：在本地环境中，经常会通过执行测试来证明工作负载能够在特定场景中正常运作。通常不会利用测试来验证恢复策略。在云中，您可以测试工作负载的故障情况，并验证您的恢复程序。您可以采用自动化方式来模拟不同的故障，也可以重新建立之前导致故障的场景。此方式可以在实际的故障发生以前揭示您可以测试与修复的故障路径，从而降低风险。
- 横向扩展以提高聚合工作负载的可用性：使用多个小型资源取代一个大型资源，以降低单个故障对整个工作负载的影响。跨多个较小的资源分配请求，以确保它们不共用常见故障点。
- 无需预估容量：本地工作负载出现故障的常见原因是资源饱和，即对工作负载的需求超过该工作负载的容量（这通常是拒绝服务攻击的目标）。在云中，您可以监控需求和工作负载利用率，并自动添加或删除资源，以保持最佳水平来满足需求，而不会出现超额预置或预置不足的问题。虽然还有很多限制，但有些限额是可控的，其他限额也可以管理（请参阅“ 管理服务限额和限制 ）。
- 通过自动化来管理变更：使用自动化方式对基础设施进行变更。需要管理的变更包括，对自动化的变更，可对其进行跟踪与审查。

定义

本白皮书涵盖了云中的可靠性，对以下四个领域的最佳实践进行描述：

- 基础
- 工作负载架构
- 变更管理
- 故障管理

要实现可靠性，您必须从基础入手，而基础是服务限额和网络拓扑适应工作负载的环境。在设计时，分布式系统的工作负载架构必须能够预防与减少故障。工作负载必须处理需求或要求的变化，而且它的设计必须能够检测故障，并自动加以修复。

主题

- 弹性，以及可靠性的组件
- 可用性
- 灾难恢复（DR）目标

弹性，以及可靠性的组件

云中的工作负载的可靠性取决于多个因素，其中最主要的要属弹性：

- 弹性是工作负载从基础设施或服务中断中恢复、动态获取计算资源以满足需求以及减少诸如配置错误或暂时性网络问题等中断的能力。

会对工作负载可靠性产生影响的其他因素还有：

- 卓越运营，其中包括变更自动化，使用行动手册对故障做出响应，以及通过运维准备情况审查（ORR）确保应用程序已经为生产运营做好准备。
- 安全性，其中包括杜绝恶意行为者破坏数据或基础设施，进而影响可用性。例如，使用加密备份以确保数据安全。
- 性能效率，其中包括通过设计在最大程度上提高工作负载的请求速率，并且将延迟最小化。
- 成本优化，其中包括权衡取舍，如确定要在 EC2 实例上投入更多以实现静态稳定性，还是在需要更大容量时依赖自动扩展。

弹性是本白皮书的主要关注点。

其他四个方面也很重要，我们将在讨论 AWS Well-Architected Framework 的对应支柱时加以介绍。这里的许多最佳实践也解决了可靠性在这些方面的问题，但重点是弹性。

可用性

可用性（也被称作服务可用性）既是定量衡量弹性的常用指标，也是需要达成的弹性目标。

- 可用性是工作负载可供使用的时间百分比。

可供使用指的是它会在有需要时成功履行其约定的功能。

这里计算的是一段时间内的百分比，例如一个月、一年或之后的三年。最严格地来说，当应用程序不能正常运行（包括计划的和计划外的中断）时，可用性就会降低。我们按如下方式定义可用性：

$$可用性 = \frac{可用时间}{总时间}$$

- 可用性是一段时间（通常是一个月或一年）内正常运行时间的百分比（例如 99.9%）
- 常见的简单表达方式仅指“9 的数量”；例如，“5 个 9”表示 99.999% 可用

- 在公式中，有些客户选择从总时间中排除计划的维护停机时间（例如，计划的维护）。但不建议采用这种方法，因为您的用户可能会在这些时间希望使用您的服务。

下表列出了常见的应用程序可用性设计目标，以及在仍然达到目标的同时，一年内可能会出现的中断的最大时长。该表包含我们在每个可用性层常常会看到的应用类型示例。在本文档中，我们会引用这些值。

可用性	最大不可用性（每年）	应用程序类别
99%	3 天 15 小时	批处理、数据提取、传输和负载作业
99.9%	8 小时 45 分钟	内部工具，如知识管理、项目跟踪
99.95%	4 小时 22 分钟	网上商务、销售点
99.99%	52 分钟	视频传输、广播工作负载
99.999%	5 分钟	ATM 交易、电信工作负载

根据请求衡量可用性。 对于您的服务，计算成功和失败的请求数可能比计算“可用时间”更容易。在这种情况下，可以如下计算：

$$可用性 = \frac{成功响应}{有效请求}$$

这通常以一分钟或五分钟为周期进行测量。然后，可以根据这些时间段的平均值计算每月正常运行时间百分比（基于时间的可用性测量）。如果在给定时间段内未收到任何请求，则该时间段内的可用性为 100%。

针对硬依赖关系计算可用性。 许多系统对其他系统具有硬依赖关系，依赖的系统中的中断会直接转换为调用系统的中断。这与软依赖关系相反，其中依赖的系统的故障会在应用程序中得到弥补。在出现此类硬依赖关系的情况下，调用系统的可用性是依赖的系统可用性的结果。例如，如果您有一个旨在实现

99.99% 可用性的系统，它对两个其他独立系统具有硬依赖关系，这两个系统都旨在实现 99.99% 的可用性，则工作负载在理论上可以实现 99.97% 的可用性：

$可用_{调用} \times 可用_{依赖项\ 1} \times 可用_{依赖项\ 2} = 可用_{工作负载}$

99.99% × 99.99% × 99.99% = 99.97%

因此，在计算您自己的可用性时，一定要了解您的依赖项及其可用性设计目标。

针对冗余组件计算可用性。 当系统涉及到使用独立的冗余组件（例如，不同可用区中的冗余资源）时，从理论上讲，可用性的计算方法是：100% 减去组件故障率的乘积。例如，如果系统使用了两个独立的组件，每个组件都具有 99.9% 的可用性，此依赖项的有效可用性为 99.9999%：

$可用_{高效} = 可用_{最大值} - ((100\% - 可用性_{依赖项}) \times (100\% - 可用性_{依赖项}))$

99.9999% = 100% − (0.1%×0.1%)

简化算法：如果您的计算中所有组件的可用性都只包含数字 9，那么您可以将数字 9 的个数相加得到您的答案。在上面的示例中，两个具有 3 个 9 可用性的冗余独立组件得到 6 个 9 可用性。

计算依赖项的可用性。 有些依赖项会提供有关其可用性的指导，包括许多 AWS 服务的可用性设计目标。但在没有相关指导的情况下（例如，制造商未发布可用性信息的组件），一个估算方式是确定平均故障间隔时间（MTBF）和平均修复时间（MTTR）。可以通过以下公式来确定可用性估算值：

$$可用_{预计} = \frac{MTBF}{MTBF + MTTR}$$

例如，如果 MTBF 为 150 天，且 MTTR 为 1 小时，则可用性估算值是 99.97%。

有关额外详细信息，请参阅可用性及其他内容：了解和提高 AWS 上分布式系统的弹性，它可帮助您计算您的可用性。

实现可用性的成本。 设计应用程序以实现更高级别的可用性通常导致成本的增加，因此在开始应用程序设计之前，应该确定真正的可用性需求。高级别的可用性对彻底失败场景下的测试和验证提出了更严格的要求。它们要求从各种故障中自动恢复，并要求系统运营的所有方面都按照相同的标准进行类似的构建和测试。例如，容量的添加或删除、更新软件或配置更改的部署或回滚，或者系统数据的迁移都必须依照预期的可用性目标来进行。可用性级别非常高时，软件部署的成本会增加，相应地，创新会受到影响，因为在部署系统时需要放慢行动速度。因此，这里的指导方针是，在系统运营的整个生命周期内，在应用标准和考虑适当的可用性目标时，要做得彻底。

在具有更高可用性设计目标的系统中，成本增加的另一种方式与依赖项的选择有关。在目标较高的情况下，可以选择作为依赖项的软件或服务集减少，具体取决于其中哪些服务已具备我们前面所说的深度投资。随着可用性设计目标的增加，通常要少找一些多用途服务（例如关系数据库），多找一些专用服务。这是因为后者更易于评估、测试和自动化，与包括在内但未使用的功能发生意外交互的可能性也较低。

灾难恢复（DR）目标

除了可用性目标之外，您的弹性策略还应包括灾难恢复（DR）目标，这些目标基于在发生灾难事件时恢复工作负载的策略。灾难恢复侧重于一次性恢复目标，以应对自然灾害、大规模技术故障或人为威胁（如攻击或错误）。这与可用性不同，可用性衡量的是一段时间内响应组件故障、负载峰值或软件错误的平均弹性。

恢复时间目标（RTO）由组织定义。RTO 是指服务中断和服务恢复之间的最大可接受延迟。这可以确定在服务不可用时被视为可接受的时间窗口。

恢复点目标（RPO）由组织定义。RPO 是指自上一个数据恢复点以来的最大可接受时间。这可以确定在上一个恢复点和服务中断之间可接受的数据丢失程度。

RPO（恢复点目标）、RTO（恢复时间目标）和灾难事件之间的关系。

RTO 和 MTTR（平均恢复时间）相似，两者都测量中断开始到工作负载恢复之间的时间。但 MTTR 取的是一段时间内多次影响可用性的事件的平均值，而 RTO 则是单次可用性影响事件允许的目标或最大值。

了解可用性需求

人们最初会认为应用程序的可用性是整个应用程序的单一目标，这种想法很常见。但是，经过仔细检查后，我们经常发现应用程序或服务的某些方面具有不同的可用性要求。例如，比起检索现有数据，某些系统可能会优先实现接收和存储新数据的功能。又或者，比起更改系统配置或环境的操作，有些系统可能会优先执行实时操作。服务可能会在一天中的某些时段具有非常高的可用性要求，但可以容忍这些时段之外的更长时间的中断。您可以通过这些方法来将单个应用程序分解成各个组成部分，并评估每个部分的可用性要求。这样做的好处是可以根据特定需求集中投入可用性方面的精力（和费用），而不是根据最严格的要求设计整个系统。

推荐

批判性地评估您的应用程序的独特方面，并在适当的情况下区分可用性和灾难恢复设计目标，以反映您的业务需求。

在 AWS，我们通常会将服务分为“数据面板”和“控制面板”。 数据平面负责交付实时服务，控制平面则用于配置环境。例如，Amazon EC2 实例、Amazon RDS 数据库和 Amazon DynamoDB 表的读/写操作都是数据平面操作。相反，启动新的 EC2 实例或 RDS 数据库，或者在 DynamoDB 中添加或更改表元数据，都属于控制平面操作。虽然高水平的可用性对所有这些功能来说都很重要，但数据平面的可用性设计目标通常比控制平面更高。因此，具有高可用性需求的工作负载应该避免运行时依赖于控制面板操作。

很多 AWS 客户会采用类似的方法批判性地评估其应用程序，并识别具有不同可用性需求的子组件。然后，针对不同的方面量身定制可用性设计目标，并执行适当的工作来设计系统。AWS 拥有根据一系列可用性设计目标设计应用程序的丰富经验，包括设计具有 99.999% 或更高可用性的服务。AWS 解决方案架构师（SA）可帮助您根据可用性目标进行合理设计。在设计过程中尽早让 AWS 参与有助于我们更好地帮助您实现可用性目标。并不是只有在启动工作负载前才要针对可用性进行规划。还应该持续不断地进行规划，从而在获得运营经验的过程中细化设计，从实际事件中吸取经验教训，并能承受不同类型的故障。然后，您可以投入适当的工作来改进实施。

工作负载所需的可用性需求必须与业务需求和关键性相符。使用定义的 RTO、RPO 和可用性定义业务关键性框架，然后您就可以对每个工作负载进行评估。此方法要求参与工作负载实施的人员对该框架，及其框架对业务需求的影响有所了解。

基础

基础要求是指其范围超出单个工作负载或项目的因素。在为任何系统设计架构之前，您应确定影响可靠性的基本要求。例如，您必须为数据中心提供足够的网络带宽。

在本地环境中，由于存在依赖关系，这些要求会导致花费较长的准备时间，因此必须在初始规划期间就考虑在内。不过，在您使用 AWS 时，这些基础要求中的大部分已经包含在内，并且您还可以根据需要进行处理。云环境在设计层面拥有几乎无限的资源，因此 AWS 要负责满足对足够联网和计算容量的需求，让您可以根据需求随意更改资源大小和分配。

下文将针对此类可靠性注意事项的最佳实践进行说明。

主题

- 管理服务限额和限制
- 计划网络拓扑

管理服务限额和限制

基于云的工作负载架构存在服务限额（也被称作服务限制）。存在这些配额是为了防止意外预置超过您所需的资源，并对 API 操作的请求速率进行限制，以保护服务不会遭到滥用。还存在资源限制，例如，将比特推入光缆的速率，或物理磁盘上的存储量。

如果您使用的是 AWS Marketplace 应用程序，则必须了解这些应用程序的限制。如果使用第三方 Web 服务或软件即服务，您也必须了解它们的限制。

最佳实践

- REL01-BP01 了解服务限额和约束
- REL01-BP02 跨多个账户和区域管理服务限额
- REL01-BP03 通过架构适应固定服务限额和限制
- REL01-BP04 监控和管理限额
- REL01-BP05 自动管理限额
- REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移

REL01-BP01 了解服务限额和约束

了解您的工作负载架构的默认限额并管理限额提高请求。了解哪些云资源约束（如磁盘或网络）可能会对您产生影响。

期望结果：客户可以通过实施正确的准则，来监视关键指标、基础设施审查和自动化补救步骤，以确认没有达到服务限额和约束（这可能导致服务降级或中断），从而防止其 AWS 账户中的服务降级或中断。

常见反模式：

- 在不了解所用服务的硬限额或软限额及其限制的情况下部署工作负载。
- 在未分析和重新配置必要限额或未事先联系支持部门的情况下，部署替代工作负载。
- 假设云服务没有限制，并且认为可以在不考虑费率、限制、计数、数量的情况下使用服务。
- 假设限额会自动增加。
- 不了解限额请求的流程和时间表。
- 假设每个服务的默认云服务限额在不同区域都是相同的。
- 假设可以突破服务约束，并且系统会自动扩展或提高限制以超出资源约束。
- 没有在流量高峰期测试应用程序，以便对资源的利用率进行压力测试。
- 在没有分析所需资源规模的情况下配置资源。
- 通过选择远远超出实际需求或预期峰值的资源类型来过量配置容量。
- 在新的客户事件或部署新技术之前，不评估新流量水平的容量需求。

建立此最佳实践的好处：对服务限额和资源约束的监视和自动化管理可以主动减少故障。如果不遵循最佳实践，客户服务的流量模式变化可能会导致中断或降级。通过监视和管理所有区域和所有账户的这些值，应用程序可以在出现不利或意外事件时具有更好的复原能力。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

Service Quotas 是一项 AWS 服务，可帮助您在一个位置管理 250 多项 AWS 服务的限额。除了查找限额值，您还可以在 Service Quotas 控制台或使用 AWS SDK 请求增加限额并跟踪。AWS Trusted Advisor 提供服务限额检查，显示您的服务使用情况，以及某些服务在某些方面的限额。有关每个服务的默认服务限额，请查看相应服务的 AWS 文档，例如，请参阅 Amazon VPC 限额。

通过配置使用计划，可在 Amazon API Gateway 内设置某些服务限制，例如限流 API 的速率限制。可通过配置对应的服务进行设置的一些限制包括预置 IOPS、已分配的 Amazon RDS 存储，以及 Amazon EBS 卷分配等。Amazon Elastic Compute Cloud 有自身的服务限制控制面板，可帮助您管理您的实例、Amazon Elastic Block Store 和弹性 IP 地址限制。如果在某用例中，服务限额会对您应用程序的性能造成影响，而且无法按您的需求进行调整，请联系 AWS Support 了解是否有解决的办法。

服务限额可以是区域特定的，也可以是全局性的。使用达到其限额的 AWS 服务将不会具有正常使用中预期的行为，并且可能会导致服务中断或降级。例如，服务限额可限制一个区域中使用的 DL Amazon EC2 数量，在使用 Auto Scaling 组（ASG）的流量扩展事件中，可能会达到该限制。

应定期评估每个账户的服务限额的使用情况，以确定适合该账户的适当服务限制。这些服务限额是作为操作防护机制存在的，以防止意外地配置超出所需数量的资源。它们还用于限制 API 操作的请求率，以保护服务不被滥用。

服务约束与服务限额不同。服务约束代表由特定资源类型定义的该资源的限制，可能是存储容量（例如，gp2 的大小限制为 1GB - 16TB）或磁盘吞吐量（10,0000 iops）。必须对资源类型的约束进行设计，并不断评估可能达到其限制的使用量。如果意外地达到约束条件，账户的应用程序或服务可能会降级或中断。

如果在某用例中，服务限额对应用程序的性能造成影响，而且无法根据所需要求进行调整，请联系 AWS Support 了解是否有解决办法。有关调整固定限额的更多详情，请参阅REL01-BP03 通过架构适应固定服务限额和限制。

有一些 AWS 服务和工具可以帮助监视和管理 Service Quotas。应该利用这些服务和工具来自动或手动检查限额水平。

- AWS Trusted Advisor 提供服务限额检查，显示您的服务使用情况，以及某些服务在某些方面的限额。它可以帮助识别接近限额的服务。
- AWS Management Console提供了显示服务限额值，管理、请求新限额，监视限额请求状态以及显示限额历史记录的方法。
- AWS CLI 和 CDK 提供了通过编程方式自动管理和监视服务限额水平和使用情况的方法。

实施步骤

对于 Service Quotas：

- 审核 AWS Service Quotas。

- 为了了解您现有的服务限额，请确定使用的服务（如 IAM Access Analyzer）。大约有 250 个 AWS 服务由服务限额控制。然后，确定每个账户和区域内可能使用的具体服务限额名称。每个区域大约有 3000 个服务限额名称。
- 使用 AWS Config 来增强这种限额分析，以查找您的 AWS 账户中使用的所有 AWS 资源。
- 使用 AWS CloudFormation 数据来确定使用的 AWS 资源。查看 AWS Management Console中创建的资源或通过 `list-stack-resources` AWS CLI 命令创建的资源。您还可以查看配置为要在模板自身部署的资源。
- 通过查看部署代码来确定工作负载所需的所有服务。
- 确定适用的服务限额。通过 Trusted Advisor 和 Service Quotas 使用能够以编程方式访问的信息。
- 建立一个自动监视方法（请参阅REL01-BP02 跨多个账户和区域管理服务限额和REL01-BP04 监控和管理限额），以便在服务限额接近或已达到其限制时发出提醒和通知。
- 建立一个自动化编程方法，以检查服务限额是否在一个区域已更改，但在同一账户的其他区域没有更改（请参阅REL01-BP02 跨多个账户和区域管理服务限额和REL01-BP04 监控和管理限额）。
- 自动扫描应用程序日志和指标，以确定是否存在任何限额或服务约束错误。如果存在这些错误，则向监视系统发送警报。
- 一旦确定特定服务需要更高限额，则制定工程设计步骤来计算所需的限额变化（请参阅REL01-BP05 自动管理限额）。
- 创建一个配置和批准工作流，以请求更改服务限额。这应该包括在请求被拒绝或部分批准情况下的例外工作流。
- 创建一个工程设计方法，在配置和使用新的 AWS 服务之前，以及在推出到生产环境或加载的环境之前（例如，负载测试账户），审查服务限额。

对于服务约束：

- 建立监视和度量方法，以提醒资源接近其资源约束。适当地利用 CloudWatch 进行指标或日志监视。
- 为每个具有对应用程序或系统有意义的约束的资源建立警报阈值。
- 创建工作流和基础设施管理过程，以在约束条件接近利用率时更改资源类型。该工作流应包括负载测试作为最佳实践，以验证新类型是新约束条件下的正确资源类型。
- 使用现有的过程和流程，将确定的资源迁移到推荐的新资源类型。

资源

相关最佳实践：

- REL01-BP02 跨多个账户和区域管理服务限额
- REL01-BP03 通过架构适应固定服务限额和限制
- REL01-BP04 监控和管理限额
- REL01-BP05 自动管理限额
- REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移
- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性

相关文档：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 服务限制
- 什么是 Service Quotas？
- 如何请求增加限额
- 服务终端节点和限额
- Service Quotas 用户指南
- AWS 的限额监控
- AWS 故障隔离界限
- 通过冗余实现可用性
- AWS 数据解决方案
- 什么是持续集成？
- 什么是持续交付？
- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- 在 AWS 上的每个租户一个账户的 SaaS 环境中管理账户生命周期
- 管理和监控工作负载中的 API 节流

- 使用 AWS Organizations 大规模查看 AWS Trusted Advisor 建议
- 使用 AWS Control Tower 自动提升服务限制并实现企业支持

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 查看和管理 AWS 服务的限额
- AWS IAM 限额演示

相关工具：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP02 跨多个账户和区域管理服务限额

如果您目前使用多个账户或区域，请确保在运行生产工作负载的所有环境中都请求适当的限额。

期望结果：对于跨账户或区域的配置，或具有使用扩展区、区域或账户失效转移的弹性设计的配置，服务和应用程序不应受到服务限额耗尽的影响。

常见反模式：

- 允许一个隔离区域内的资源利用率增加，但没有相关机制保持其他隔离区域中的容量。
- 手动单独设置隔离区域中的所有限额。

- 没有考虑到在非主要区域出现降级期间，弹性架构（如主动或被动）对未来限额需求的影响。
- 不定期评估限额，并且不在工作负载运行的每个区域和账户中进行必要更改。
- 不利用限额请求模板来请求提高多个区域和账户的限额。
- 不更新服务限额，因为错误地认为提高限额会像计算预留请求一样产生影响成本。

建立此最佳实践的好处：验证在区域服务不可用时，您能否在辅助区域或账户中处理您当前的负载。这可以帮助降低在区域丢失期间发生的错误数量或降级水平。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

每个账户的服务限额都可被跟踪。除非另有说明，否则每个限额都针对的是特定的 AWS 区域。除生产环境以外，还要管理所有适用的非生产环境中的限额，以避免妨碍测试与开发。保持高度复原能力需要持续评估服务限额（无论是自动还是手动）。

由于实施的设计采用主动/主动、主动/被动 – 常用、主动/被动 – 非常用和主动/被动 – 指示灯方法，会有更多工作负载跨越区域，因此了解所有区域和账户限额水平至关重要。如果服务限额设置正确，过去的流量模式并不总是一个好的指标。

同样重要的是，服务限额名称限制并非对每个区域都始终相同。在一个区域，该值可能是 5，而在另一个区域，该值可能是 10。必须跨越所有相同的服务、账户和区域来管理这些限额，以便在负载状态下提供一致的复原能力。

协调不同区域（主动区域或被动区域）之间的所有服务限额差异，并创建流程来持续协调这些差异。被动区域失效转移的测试计划很少扩展到能够满足峰值主动容量，这意味着实际试用或桌面演练可能无法发现区域之间的服务限额差异，因此也无法保持正确的限制。

服务限额偏移对于跟踪和评估非常重要，它是指特定命名限额在一个区域而非所有区域发生更改的情况。应考虑更改在有流量或可能有流量的区域的限额。

- 根据您的服务要求、延迟、法规和灾难恢复（DR）要求选择相关账户和区域。
- 确定跨所有相关账户、区域和可用区的服务限额。限制的范围具体到账户和区域。应比较这些值以了解差异。

实施步骤

- 审查可能已经超出风险使用水平的 Service Quotas 值。AWS Trusted Advisor 会在超出 80% 和 90% 阈值时提供警报。

- 审查任何被动区域（在主动/被动设计中）的服务限额值。验证在主区域发生故障时，负载能否在辅助区域成功运行。
- 自动评估同一账户的不同区域之间是否发生了任何服务限额偏移，并采取相应的行动来更改限制。
- 如果客户的组织单位（OU）以受支持的方式构建，则应更新服务限额模板，以反映任何限额的变化，这些变化应该应用于多个区域和账户。
  - 创建模板并将区域与限额变化关联起来。
  - 审查所有现有的服务限额模板，看看是否需要进行任何更改（区域、限制和账户）。

资源

相关最佳实践：

- REL01-BP01 了解服务限额和约束
- REL01-BP03 通过架构适应固定服务限额和限制
- REL01-BP04 监控和管理限额
- REL01-BP05 自动管理限额
- REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移
- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性

相关文档：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 服务限制
- 什么是 Service Quotas？
- 如何请求增加限额
- 服务终端节点和限额

- Service Quotas 用户指南
- AWS 的限额监控
- AWS 故障隔离界限
- 通过冗余实现可用性
- AWS 数据解决方案
- 什么是持续集成？
- 什么是持续交付？
- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- 在 AWS 上的每个租户一个账户的 SaaS 环境中管理账户生命周期
- 管理和监控工作负载中的 API 节流
- 使用 AWS Organizations 大规模查看 AWS Trusted Advisor 建议
- 使用 AWS Control Tower 自动提升服务限制并实现企业支持

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 查看和管理 AWS 服务的限额
- AWS IAM 限额演示

相关服务：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP03 通过架构适应固定服务限额和限制

了解不可更改的服务限额、服务限制和物理资源限制。为应用程序和服务设计架构，以防止这些限制影响可靠性。

其中的示例包括网络带宽、无服务器功能调用有效负载大小、API Gateway 的节流突发速率，以及并发用户连接至数据库。

期望结果：应用程序或服务在正常条件下和高流量条件下按预期执行。它们设计为在该资源的固定约束或服务限额的限制范围内工作。

常见反模式：

- 选择使用一项服务资源的设计时，没有意识到设计存在限制，这些限制将导致扩展时设计失败。
- 执行不现实的基准测试，并且在测试期间将达到服务固定限额。例如，以突发限制运行测试，但运行时间较长。
- 选择在超过固定服务限额时无法扩展或修改的设计。例如，SQS 有效负载大小为 256KB。
- 没有设计和实施可观测性，以便监控在高流量事件期间可能面临风险的服务限额阈值并发出警报。

建立此最佳实践的好处：确认应用程序将在所有预计的服务负载水平下运行，而不会中断或降级。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

与软服务限额或替换为更高容量单位的资源不同，无法更改 AWS 服务的固定限额。这意味着，在应用程序设计中使用所有这些类型的 AWS 服务时，必须评估是否存在潜在的硬容量限制。

Service Quotas 控制台中显示了硬限制。如果列中显示 `ADJUSTABLE = No`，则服务具有硬限制。一些资源配置页中也显示了硬限制。例如，Lambda 具有无法调整的特定硬限制。

例如，在设计 Python 应用程序以在 Lambda 函数中运行时，应评估应用程序，以确定 Lambda 是否有可能运行超过 15 分钟。如果代码可能运行超过此服务限额限制，则必须考虑替代技术或设计。如果在生产部署后达到此限制，则应用程序将遭受降级和中断，直到可以补救为止。与软限额不同，即使在紧急的严重性 1 事件下，也无法更改这些限制。

应用程序部署到测试环境之后，应使用策略来查明是否会达到任何硬限制。引入测试计划中应包括压力测试、负载测试和混沌测试。

实施步骤

- 查看可在应用程序设计阶段使用的 AWS 服务的完整列表。
- 查看所有这些服务的软限额限制和硬限额限制。Service Quotas 控制台中并没有显示所有限制。有些服务描述了备用位置的这些限制。
- 在设计应用程序时，检查工作负载的业务和技术驱动因素，例如业务成果、使用案例、相依系统、可用性目标和灾难恢复对象。让业务和技术驱动因素指导流程，以确定适合工作负载的分布式系统。
- 分析各个区域和账户的服务负载。服务的许多硬限制基于区域。但有些限制基于账户。
- 分析弹性架构在可用区故障和区域故障期间的资源使用情况。在使用“主动/主动”、“主动/被动 – 热”、“主动/被动 - 冷”和“主动/被动 - 指示灯”方法的多区域设计过程中，这些故障情况会导致使用率更高。这会形成达到硬限制的潜在使用案例。

资源

相关最佳实践：

- REL01-BP01 了解服务限额和约束
- REL01-BP02 跨多个账户和区域管理服务限额
- REL01-BP04 监控和管理限额
- REL01-BP05 自动管理限额
- REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移
- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性

相关文档：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 服务限制
- 什么是 Service Quotas？

- 如何请求增加限额
- 服务终端节点和限额
- Service Quotas 用户指南
- AWS 的限额监控
- AWS 故障隔离界限
- 通过冗余实现可用性
- AWS 数据解决方案
- 什么是持续集成？
- 什么是持续交付？
- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- 在 AWS 上的每个租户一个账户的 SaaS 环境中管理账户生命周期
- 管理和监控工作负载中的 API 节流
- 使用 AWS Organizations 大规模查看 AWS Trusted Advisor 建议
- 使用 AWS Control Tower 自动提升服务限制并实现企业支持
- Service Quotas 的操作、资源和条件键

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 查看和管理 AWS 服务的限额
- AWS IAM 限额演示
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统

相关工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor

- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP04 监控和管理限额

评估您的可能使用情况，并适当提高您的限额，支持使用量按计划增长。

期望结果：部署了可进行管理和监控的主动和自动化系统。这些操作解决方案可确保接近达到限额使用阈值。根据请求的限额更改主动修复这些问题。

常见反模式：

- 未配置监控以检查服务限额阈值
- 没有为硬限制配置监控，即使这些值不能更改。
- 假定请求和确立软限额变化所需的时间是即时或短时间。
- 配置警报，以在快达到服务限额时发出警报，但没有关于如何对提醒做出响应的流程。
- 只为 AWS Service Quotas 支持的服务配置警报，不监控其他 AWS 服务。
- 不考虑多区域弹性设计（如“主动/主动”、“主动/被动 – 热”、“主动/被动 - 冷”和“主动/被动 - 指示灯”方法）的限额管理。
- 不评估区域之间的限额差异。
- 不评估每个区域对特定限额增加请求的需求。
- 不利用模板进行多区域限额管理。

建立此最佳实践的好处：自动跟踪 AWS Service Quotas，并根据这些限额监控您的使用情况，使您可以了解何时达到限额。您还可以使用此监控数据帮助限制由于限额耗尽而导致的降级。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

对于支持的服务，您可以通过配置各种可以进行评测的不同服务，然后发送警报，从而监控限额。这有助于监控使用情况，并可在接近限额时提醒您。这些警报可以从 AWS Config、Lambda 函数、Amazon CloudWatch 或从 AWS Trusted Advisor 触发。您还可以使用 CloudWatch Logs 上的指标筛选条件来搜索与提取日志中的模式，确定使用量是否快达到限额阈值。

实施步骤

对于监控：

- 获取当前资源使用量（例如存储桶或实例）。使用服务 API 操作（例如 Amazon EC2 `DescribeInstances` API）来收集当前资源使用量。
- 使用以下项获得必要且适用于服务的当前限额：
  - AWS Service Quotas
  - AWS Trusted Advisor
  - AWS 文档
  - AWS 服务特定页面
  - AWS Command Line Interface（AWS CLI）
  - AWS Cloud Development Kit (AWS CDK)
- AWS Service Quotas 是一项 AWS 服务，使用该服务可帮助您在一个地方管理超过 250 项 AWS 服务的限额。
- 使用 Trusted Advisor 服务限制来监控在各种阈值下的当前服务限制。
- 使用服务限额历史记录（控制台或 AWS CLI）来检查区域增长情况。
- 如果需要，比较每个区域和每个账户中的服务限额变化，以形成等效性。

对于管理：

- 自动：设置 AWS Config 自定义规则以扫描各个区域的服务限额，并比较它们之间的差异。
- 自动：设置计划好的 Lambda 函数以扫描各个区域的服务限额，并比较它们之间的差异。
- 手动：通过 AWS CLI、API 或 AWS 控制台来扫描各个区域的服务限额，并比较它们之间的差异。报告差异。
- 如果在不同区域之间发现限额差异，如有需要，请求限额更改。
- 检查所有请求的结果。

资源

相关最佳实践：

- REL01-BP01 了解服务限额和约束
- REL01-BP02 跨多个账户和区域管理服务限额

- REL01-BP03 通过架构适应固定服务限额和限制
- REL01-BP05 自动管理限额
- REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移
- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性

相关文档：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 服务限制
- 什么是 Service Quotas？
- 如何请求增加限额
- 服务终端节点和限额
- Service Quotas 用户指南
- AWS 的限额监控
- AWS 故障隔离界限
- 通过冗余实现可用性
- AWS 数据解决方案
- 什么是持续集成？
- 什么是持续交付？
- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- 在 AWS 上的每个租户一个账户的 SaaS 环境中管理账户生命周期
- 管理和监控工作负载中的 API 节流
- 使用 AWS Organizations 大规模查看 AWS Trusted Advisor 建议

- 使用 AWS Control Tower 自动提升服务限制并实现企业支持
- Service Quotas 的操作、资源和条件键

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 查看和管理 AWS 服务的限额
- AWS IAM 限额演示
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统

相关工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP05 自动管理限额

实施工具以便在接近阈值时向您发送提醒。您可以自动发出限额提高请求：通过使用 AWS Service Quotas API，您可以自动发出限额提高请求。

如果将您的配置管理数据库 (CMDB) 或票证系统与 Service Quotas 集成，您可以自动跟踪配额提高请求和当前配额。除了 AWS 开发工具包之外，Service Quotas 还使用 AWS Command Line Interface（AWS CLI）提供自动化。

常见反模式：

- 以电子表格的形式跟踪配额和使用情况。

- 每天、每周或每月运行使用情况报告，然后将使用情况与配额进行比较。

建立此最佳实践的好处： 自动跟踪 AWS 服务限额，并根据这些限额监控您的使用情况，从而让您了解何时接近限额。您可以设置自动流程，帮助您在需要时提出配额提高请求。当使用情况趋向于相反的方向时，您可能需要考虑降低一些限额，以实现降低风险（如果凭据被盗）和节省成本的效果。

未建立此最佳实践暴露的风险等级： 中

实施指导

- 设置自动监控：通过开发工具包实施各种工具，以便在接近阈值时向您发出提醒。
  - 利用 Service Quotas，通过自动限额监控解决方案（例如 AWS Limit Monitor 或从 AWS Marketplace 获得的产品）来增强服务。
    - 什么是 Service Quotas？
    - AWS 上的限额监控 – AWS 解决方案
  - 使用 Amazon SNS 和 AWS Service Quotas API 来根据限额阈值来设置触发响应。
  - 测试自动化。
    - 配置限制阈值。
    - 与来自 AWS Config、部署管道、Amazon EventBridge 或第三方的更改事件集成。
    - 人工设置低限额阈值来测试响应。
    - 设置触发器以根据通知采取适当措施，并在必要时联系 AWS Support。
    - 人工触发更改事件。
    - 运行实际测试以测试限额提高更改流程。

资源

相关文档：

- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- AWS Marketplace：可以帮助跟踪限制的 CMDB 产品
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（见“服务限制”部分）
- AWS 上的限额监控 – AWS 解决方案
- Amazon EC2 服务限制

- 什么是 Service Quotas？

相关视频：

- AWS Live re:Inforce 2019 – Service Quotas

REL01-BP06 确保在当前限额与最大使用量之间存在足够的差距，以便应对失效转移

当资源出现故障或无法访问时，该资源可能仍会被计入限额，直到成功终止资源。确认您的限额涵盖出现故障或无法访问的资源及其替换资源的重叠部分。在计算此差距时，应考虑网络故障、可用区故障或区域故障等使用案例。

期望结果：在当前服务阈值内可以覆盖资源或资源可访问性方面的或大或小的故障。在资源规划中已考虑到可用区故障、网络故障或甚至是区域故障。

常见反模式：

- 根据当前需求设置服务限额，而不考虑失效转移场景。
- 在计算服务的峰值限额时不考虑静态稳定性原则。
- 在计算每个区域所需的总限额时，不考虑可能无法访问资源的情况。
- 不考虑某些服务的 AWS 服务故障隔离边界及其可能的异常使用模式。

建立此最佳实践的好处：当服务中断事件影响应用程序可用性时，您可以借助云实施策略来缓解影响或从这些事件中恢复。此类策略通常包括创建额外资源来替换出现故障或无法访问的资源。您的限额策略会适应这些失效转移条件，并且不会由于服务限制耗尽而导致额外的降级。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

在评估限额限制时，请考虑由于某些降级而可能发生的失效转移情况。应考虑以下类型的失效转移情况：

- VPC 中断或无法访问。
- 子网无法访问。
- 可用区已显著降级，导致影响许多资源的可访问性。

- 系统阻止或更改了各种网络路由或入口点和出口点。
- 区域已显著降级，导致影响许多资源的可访问性。
- 有多个资源，但并非所有资源都受到区域或可用区故障的影响。

如上所列的故障可能是启动失效转移事件的触发器。因为业务影响可能会有很大差异，每种情况和每个客户的失效转移决策都是独特的。但是，当从操作上决定失效转移应用程序或服务时，必须在事件发生之前解决失效转移位置中资源的容量规划及其相关限额。

检查每个服务的服务限额，要考虑到可能会发生高于正常峰值的情况。这些峰值可能与由于网络或权限而可以访问但仍处于活动状态的资源有关。未终止的活动资源仍将计入服务限额限制。

实施步骤

- 确认您的服务限额与最高使用量之间有足够的差距，以便适应失效转移或失去可访问性。
- 根据您的部署模式、可用性要求和使用量增长情况确定服务限额。
- 根据需要请求增加限额。预计完成限额提高请求所需的时间。
- 确定可靠性要求（也称为“X 个 9”）。
- 构建故障场景（例如组件、可用区或区域缺失）。
- 确定部署方法（例如金丝雀部署、蓝/绿部署、红/黑部署或滚动部署）。
- 在当前限制中包含适当的缓冲区（例如 15%）。
- 在适当情况下包括静态稳定性（可用区和区域）的计算。
- 预计使用量增长（例如监控使用量趋势）。
- 考虑静态稳定性对最关键工作负载的影响。评估所有区域和可用区中适应静态稳定系统的资源。
- 考虑使用按需容量预留，以便在发生任何失效转移之前安排容量。在最关键的业务计划期间，这是一种有用的策略，可以在失效转移期间获得正确数量和类型的资源时降低潜在的风险。

资源

相关最佳实践：

- REL01-BP01 了解服务限额和约束
- REL01-BP02 跨多个账户和区域管理服务限额
- REL01-BP03 通过架构适应固定服务限额和限制
- REL01-BP04 监控和管理限额

- REL01-BP05 自动管理限额
- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性

相关文档：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（以前称为服务限制）
- AWS Trusted Advisor 最佳实践检查（请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 服务限制
- 什么是 Service Quotas？
- 如何请求增加限额
- 服务终端节点和限额
- Service Quotas 用户指南
- AWS 的限额监控
- AWS 故障隔离界限
- 通过冗余实现可用性
- AWS 数据解决方案
- 什么是持续集成？
- 什么是持续交付？
- APN 合作伙伴：可帮助进行配置管理的合作伙伴
- 在 AWS 上的每个租户一个账户的 SaaS 环境中管理账户生命周期
- 管理和监控工作负载中的 API 节流
- 使用 AWS Organizations 大规模查看 AWS Trusted Advisor 建议
- 使用 AWS Control Tower 自动提升服务限制并实现企业支持
- Service Quotas 的操作、资源和条件键

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 查看和管理 AWS 服务的限额
- AWS IAM 限额演示
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统

相关工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

计划网络拓扑

工作负载通常存在于多个环境中。其中包括多个云环境（可公开访问云和私有云），可能还包括现有数据中心基础设施。相关计划必须涵盖网络注意事项，如系统内部和系统间连接、公有 IP 地址管理、私有 IP 地址管理以及域名解析。

在使用基于 IP 地址的网络构建系统时，您必须规划网络拓扑并预计可能的故障，从而应对未来的增长以及与其他系统及其网络的集成。

Amazon Virtual Private Cloud（Amazon VPC）让您可以在 AWS 云中预置一个私有隔离的部分，并在虚拟网络中启动 AWS 资源。

最佳实践

- REL02-BP01 为工作负载公共端点使用高度可用的网络连接
- REL02-BP02 为云环境和本地环境之间的私有网络预置冗余连接

- REL02-BP03 确保 IP 子网分配考虑扩展和可用性
- REL02-BP04 轴辐式拓扑优先于多对多网格
- REL02-BP05 在互相连接的所有私有地址空间中强制实施非重叠的私有 IP 地址范围

REL02-BP01 为工作负载公共端点使用高度可用的网络连接

构建与工作负载的公共端点的高可用性网络连接有助于减少因连接丢失而导致的停机，并提高工作负载的可用性和改进 SLA。为此，需使用高度可用的 DNS、内容分发网络（CDN）、API Gateway、负载均衡或反向代理。

期望结果：为公共端点规划、构建和实施高可用性网络连接至关重要。如果您的工作负载由于连接中断而无法访问，即使您的工作负载正在运行且可用，您的客户也会看到您的系统处于关闭状态。通过将工作负载的公共端点的高可用性和弹性网络连接与工作负载本身的弹性架构结合起来，您可以为客户提供最佳的可用性和服务级别。

AWS Global Accelerator、Amazon CloudFront、Amazon API Gateway、AWS Lambda 函数 URL、AWS AppSync API 和 Elastic Load Balancing（ELB）均提供高可用性公共端点。Amazon Route 53 为域名解析提供高可用性 DNS 服务，以便确认可以解析公共端点地址。

您还可以评估 AWS Marketplace 软件设备是否适用于负载均衡和代理。

常见反模式：

- 在没有规划 DNS 和网络连接以实现高可用性的情况下设计高可用性工作负载。
- 在各个实例或容器中使用公有互联网地址并使用 DNS 管理与它们的连接。
- 使用 IP 地址而非域名来查找服务。
- 没有测试与公共端点的连接丢失的场景。
- 没有分析网络吞吐量需求和分发模式。
- 没有测试和计划与工作负载的公共端点的互联网连接可能中断的情况。
- 为较大地理区域提供内容（如网页、静态资产或媒体文件），而不使用内容分发网络。
- 没有为分布式拒绝服务（DDoS）攻击制定计划。DDoS 攻击会引发关闭您的用户的合法流量并降低可用性的风险。

建立此最佳实践的好处：设计高可用性和弹性网络连接，确保用户可以访问和使用您的工作负载。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

构建与公共端点的高可用性网络连接的核心是流量的路由。为了验证您的流量是否能够到达端点，DNS 必须能够将域名解析为其相应的 IP 地址。使用 Amazon Route 53 等高可用性和可扩展域名系统（DNS）来管理域的 DNS 记录。您还可以使用 Amazon Route 53 提供的运行状况检查。运行状况检查确认您的应用程序可访问、可用且正常运行，并且通过特殊的方式设置它们，使它们可以模仿用户的行为，例如请求网页或特定 URL。如果发生故障，Amazon Route 53 会响应 DNS 解析请求，并仅将流量定向到运行正常的端点。您还可以考虑使用 Amazon Route 53 提供的 Geo DNS 和基于延迟的路由功能。

为了确认您的工作负载本身具有高可用性，请使用 Elastic Load Balancing（ELB）。Amazon Route 53 可用于将流量定向到 ELB，然后它将流量分发到目标计算实例。您还可以将 Amazon API Gateway 与 AWS Lambda 结合使用，以实现无服务器解决方案。客户也可以在多个 AWS 区域 中运行工作负载。借助多站点主动/主动模式，工作负载可以从多个区域提供流量。借助多站点主动/被动模式，工作负载可以从活动区域提供流量，而数据复制到辅助区域，在主区域发生故障时它变为活动。然后可使用 Route 53 运行状况检查来控制 DNS 从主区域中的任何端点失效转移到辅助区域中的端点，确认用户可以访问和使用您的工作负载。

Amazon CloudFront 提供一个简单的 API，通过使用世界各地的边缘站点网络提供请求，从而分发具有低延迟和高数据传输速率的内容。内容分发网络（CDN）提供位于或缓存在用户附近位置的内容，从而为客户提供服务。因为内容的加载从您的服务器转移到 CloudFront 的边缘站点，所以这也可以提高应用程序的可用性。边缘站点和区域边缘高速缓存在靠近查看者的位置保存内容的缓存副本，以便可以快速检索并提高工作负载的可访问性和可用性。

对于用户在地理上分散的工作负载，AWS Global Accelerator 可帮助您提高应用程序的可用性和性能。AWS Global Accelerator 提供任播静态 IP 地址，它充当在一个或多个 AWS 区域 中托管的应用程序的固定入口点。这样就可以让流量在尽可能靠近用户的位置进入 AWS 全球网络，从而提高工作负载的可访问性和可用性。AWS Global Accelerator 还使用 TCP、HTTP 和 HTTPS 运行状况检查来监控应用程序端点的运行状况。端点的运行状况或配置发生任何更改，都会触发用户流量重定向到正常运行的端点，为用户提供最佳性能和可用性。此外，AWS Global Accelerator 采用故障隔离设计，使用由独立网络区域提供的两个静态 IPv4 地址，提高了应用程序的可用性。

为帮助客户防范 DDoS 攻击，AWS 提供了 AWS Shield Standard。Shield Standard 会自动启用并防范 SYN/UDP 泛洪和反射攻击等常见的基础设施（第 3 层和第 4 层）攻击，以便在 AWS 上支持应用程序的高可用性。要对更复杂和更大规模的攻击（如 UDP 泛洪）、状态耗尽攻击（如 TCP SYN 泛洪）提供额外保护，以及为了帮助保护 Amazon Elastic Compute Cloud（Amazon EC2）、Elastic Load Balancing（ELB）、Amazon CloudFront、AWS Global Accelerator 和 Route 53 上运行的应用程序，您可以考虑使用 AWS Shield Advanced。为了防范 HTTP POST 或 GET 泛洪等应用程序层攻

击，请使用 AWS WAF。AWS WAF 可使用 IP 地址、HTTP 标头、HTTP 主体、URI 字符串、SQL 注入和跨站点脚本条件来确定是应该阻止还是允许请求。

实施步骤

1. 设置高可用性 DNS：Amazon Route 53 是高可用性和可扩展的域名系统（DNS）Web 服务。Route 53 将用户请求连接到在 AWS 上或在本地运行的互联网应用程序。有关更多信息，请参阅将 Amazon Route 53 配置为 DNS 服务。
2. 设置运行状况检查：当使用 Route 53 时，确认只有正常运行的目标才可解析。首先创建 Route 53 运行状况检查和配置 DNS 故障转移。在设置运行状况检查时，必须考虑以下方面：
   a. Amazon Route 53 如何确定运行状况检查是否正常
   b. 创建、更新和删除运行状况检查
   c. 监控运行状况检查状态和获得通知
   d. Amazon Route 53 DNS 的最佳实践
3. 将 DNS 服务连接到端点。
   a. 当使用 Elastic Load Balancing 作为流量的目标时，使用指向负载均衡器的区域端点的 Amazon Route 53 来创建别名记录。在创建别名记录期间，将评估目标运行状况选项设置为“是”。
   b. 对于无服务器工作负载或私有 API，当使用 API Gateway 时，使用 Route 53 将流量引向 API Gateway。
4. 决定内容分发网络。
   a. 要使用更靠近用户的边缘站点传输内容，首先了解 CloudFront 如何传输内容。
   b. 开始使用简单 CloudFront 分发。然后，CloudFront 知道您希望从何处传输内容，还知道有关如何跟踪和管理内容传输的详细信息。在设置 CloudFront 分发时，必须了解和考虑以下方面：
      i. 缓存如何与 CloudFront 边缘站点配合使用
      ii. 提高直接从 CloudFront 缓存提供的请求的比例（缓存命中率）
      iii. 使用 Amazon CloudFront Origin Shield
      iv. 使用 CloudFront 来源失效转移优化高可用性
5. 设置应用程序层保护：AWS WAF 帮助您免受可能会影响可用性、危及安全性或消耗过多资源的常见 Web 漏洞和机器人的攻击。若要更深入了解，请查看 AWS WAF 的工作原理，并且在您准备好实施保护措施来防范应用程序层 HTTP POST AND GET 泛洪时，请查看开始使用 AWS WAF。您还可以使用 AWS WAF 和 CloudFront，请参阅有关 AWS WAF 如何与 Amazon CloudFront 功能配合使用的文档。
6. 设置额外的 DDoS 防护：默认情况下，所有 AWS 客户都可以免费使用 AWS Shield Standard 获得保护，防范针对您的网站或应用程序的常见、最常发生的网络和传输层 DDoS 攻击。对于

在 Amazon EC2、Elastic Load Balancing、Amazon CloudFront、AWS Global Accelerator 和 Amazon Route 53 上运行，面向互联网的应用程序，要获得额外保护，您可以考虑 AWS Shield Advanced 并查看 DDoS 弹性架构的示例。要保护您的工作负载和公共端点，防止受到 DDoS 攻击，请参阅开始使用 AWS Shield Advanced。

资源

相关最佳实践：

- REL10-BP01 将工作负载部署到多个位置
- REL10-BP02 为您的多位置部署选择合适的位置
- REL11-BP04 恢复期间依赖于数据平面而不是控制平面
- REL11-BP06 当事件影响可用性时发出通知

相关文档：

- AWS 合作伙伴：可帮助您规划联网的合作伙伴
- 适用于网络基础设施的 AWS Marketplace
- 什么是 AWS Global Accelerator？
- 什么是 Amazon CloudFront？
- 什么是 Amazon Route 53？
- 什么是 Elastic Load Balancing？
- 网络连接能力 - 建立您的云基础
- 什么是 Amazon API Gateway？
- 什么是 AWS WAF、AWS Shield 和 AWS Firewall Manager？
- 什么是 Amazon Route 53 Application Recovery Controller？
- 配置自定义运行状况检查来进行 DNS 故障转移

相关视频：

- AWS re:Invent 2022 - 使用 AWS Global Accelerator 提高性能和可用性
- AWS re:Invent 2020：使用 Amazon Route 53 进行全球流量管理
- AWS re:Invent 2022 - 运行高可用性多可用区应用程序

- AWS re:Invent 2022 - 深入了解 AWS 网络基础设施
- AWS re:Invent 2022 - 建立弹性网络

相关示例：

- 使用 Amazon Route 53 Application Recovery Controller (ARC) 进行灾难恢复
- 可靠性研讨会
- AWS Global Accelerator 研讨会

REL02-BP02 为云环境和本地环境之间的私有网络预置冗余连接

在云环境和本地环境中的专用网络之间设置冗余连接，以实现连接的韧性。这可以通过部署两条或更多链路和流量路径来实现，从而在网络出现故障时仍然保持连接。

常见的反面模式：

- 仅依赖一个网络连接，这会造成单点故障。
- 仅使用一个 VPN 隧道，或者使用多个隧道，但是多个隧道又连接到同一个可用区。
- 依赖一家互联网服务提供商（ISP）来提供 VPN 连接，这会导致在 ISP 中断期间彻底故障。
- 未实施像 BGP 这样的动态路由协议，这些协议对于在网络中断期间重新路由流量至关重要。
- 忽略了 VPN 隧道的带宽限制，高估了 VPN 隧道的备份能力。

建立此最佳实践的好处：通过在云环境和企业或本地环境之间实施冗余连接，两个环境之间的依赖服务能够可靠通信。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 AWS Direct Connect 将您的本地网络连接到 AWS 时，如果使用不同的连接来连接到多个本地位置和多个 AWS Direct Connect 位置中的不同设备，则可以实现出色的网络韧性（SLA 为 99.99%）。这种拓扑结构可抵御设备故障、网络连接问题以及彻底的位置中断。或者，您可以通过使用两个单独的连接与多个位置相连（每个本地位置连接到一个 Direct Connect 位置）来实现较高的韧性（SLA 达到 99.9%）。这种方法可以防止因光纤中断或设备故障而导致的连接中断，并有助于缓解完全的位置故障。AWS Direct Connect 韧性工具包有助于您设计 AWS Direct Connect 拓扑。

您也可以考虑使用与 AWS Transit Gateway 相连的 AWS Site-to-Site VPN，以经济实惠的方式备份至主 AWS Direct Connect 连接。这种设置支持跨多个 VPN 隧道的等价多路径（ECMP）路由，即使每个 VPN 隧道的吞吐量上限为 1.25 Gbps，也能实现高达 50 Gbps 的吞吐量。但值得注意的是，AWS Direct Connect 仍然是大幅减少网络中断并实现稳定连接的极佳选择。

在通过互联网使用 VPN 将您的云环境连接到本地数据中心时，将两个 VPN 隧道配置为单个 Site-to-Site VPN 连接的一部分。为了实现高可用性，每条隧道都应连接到不同的可用区，并使用冗余硬件来防止本地设备故障。此外，可以考虑使用不同互联网服务提供商（ISP）的多个互联网连接，来连接到您的本地位置，以避免因单个 ISP 中断而导致彻底中断 VPN 连接。选择具有不同路由和基础设施的 ISP，尤其是那些具有单独物理路径通往 AWS 端点的 ISP，可实现高连接可用性。

除了通过多个 AWS Direct Connect 连接和/或多个 VPN 隧道实现物理冗余外，实现边界网关协议（BGP）动态路由也至关重要。动态 BGP 可根据实时网络状况和配置的策略，自动将流量从一条路径重新路由到另一条路径。这种动态行为特别有助于在链路或网络出现故障时，保持网络可用性和服务连续性，进而快速选择替代的路径，增强网络的韧性和可靠性。

实施步骤

- 在 AWS 和本地环境之间获取高度可用的连接。
  - 在单独部署的专用网络之间使用多个 AWS Direct Connect 连接或 VPN 隧道。
  - 使用多个 AWS Direct Connect 位置来实现高可用性。
  - 如果使用多个 AWS 区域，请至少在其中两个区域中创建冗余。
- 如果可能，请尽量将您的 VPN 连接连接到 AWS Transit Gateway。
- 评估将 VPN 连接到 AWS Marketplace 设备，或者使用这些设备将 SD-WAN 扩展到 AWS 的方案。如果您使用 AWS Marketplace 设备，请在不同的可用区中部署冗余实例以实现高可用性。
- 提供面向本地环境的冗余连接。
  - 您可能需要面向多个 AWS 区域的冗余连接，来满足可用性需求。
  - 使用 AWS Direct Connect 韧性工具包开始操作。

资源

相关文档：

- AWS Direct Connect Resiliency Recommendations
- Using Redundant Site-to-Site VPN Connections to Provide Failover
- 路由策略和 BGP 社区

- Active/Active and Active/Passive Configurations in AWS Direct Connect
- AWS 合作伙伴：可帮助您规划联网的合作伙伴
- 适用于网络基础设施的 AWS Marketplace
- Amazon Virtual Private Cloud Connectivity Options 白皮书
- 构建可扩展且安全的多 VPC AWS 网络基础设施
- Using redundant Site-to-Site VPN connections to provide failover
- Using the AWS Direct Connect Resiliency Toolkit to get started
- VPC 端点和 VPC 端点服务（AWS PrivateLink）
- Amazon VPC 是什么？
- What is a transit gateway?
- What is AWS Site-to-Site VPN?
- 使用 Direct Connect 网关

相关视频：

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs

REL02-BP03 确保 IP 子网分配考虑扩展和可用性

Amazon VPC IP 地址范围必须足够大，以满足工作负载的要求，包括考虑未来的扩展以及跨可用区为子网分配 IP 地址。这包括负载均衡器、EC2 实例和基于容器的应用程序。

当您规划网络拓扑时，第一步是定义 IP 地址空间本身。应（按照 RFC 1918 准则）为每个 VPC 分配私有 IP 地址范围。作为此流程的一部分，要满足以下要求：

- 在每个区域中为多个 VPC 留出 IP 地址空间。
- 在 VPC 内，为多个子网留出空间，这样您就可以跨多个可用区。
- 请考虑在 VPC 内保留未使用的 CIDR 块空间以用于未来扩展。
- 确保有 IP 地址空间以满足您可能使用的任何 Amazon EC2 实例临时性实例集的需求，如用于机器学习的竞价型实例集、Amazon EMR 集群或 Amazon Redshift 集群。对于 Amazon Elastic Kubernetes Service（Amazon EKS）等 Kubernetes 集群应该也采取类似的考虑，因为默认情况下，每个 Kubernetes 容器组（pod）都会从 VPC CIDR 块中分配一个可路由的地址。
- 注意，每个子网 CIDR 块中的前四个 IP 地址和最后一个 IP 地址将被预留而无法供您使用。

- 注意，最初被分配到您的 VPC 的 VPC CIDR 块无法被更改或删除，但您可以向 VPC 添加额外的非重叠的 CIDR 块。虽然无法更改子网 IPv4 CIDR，但可以更改 IPv6 CIDR。
- 可以使用的最大 VPC CIDR 块为 /16，最小为 /28。
- 考虑其它互联网络（VPC、本地部署或其它云提供商），并确保 IP 地址空间不重叠。有关更多信息，请参阅 REL02-BP05 在互相连接的所有私有地址空间中强制实施非重叠的私有 IP 地址范围。

期望的结果： 可扩展的 IP 子网有助于您适应未来的增长，并避免不必要的浪费。

常见反模式：

- 没有考虑未来的增长，导致 CIDR 块过小且需要重新配置，这可能会造成停机。
- 错误估计弹性负载均衡器可以使用的 IP 地址数量。
- 在相同子网中部署多个高流量负载均衡器。
- 使用自动扩缩机制，但未能监控 IP 地址使用情况。
- 定义过大的 CIDR 范围，远远超出未来的增长预期，这会导致地址范围重叠，难以与其它网络建立对等连接。

建立此最佳实践的好处： 这可确保您能适应工作负载增长要求，并在扩展过程中继续提供可用性。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

规划您的网络以适应增长、符合监管合规性以及实现与其它服务的集成。如果没有合理的规划，则增长可能会被低估、监管合规性可能会发生变化并且收购或私有网络连接可能难以实施。

- 根据您的服务要求、延迟、法规和灾难恢复（DR，Disaster Recovery）要求，选择相关 AWS 账户和区域。
- 确定您的区域 VPC 部署需求。
- 确定 VPC 的大小。
  - 确定您是否要部署多 VPC 连接。
    - 什么是 Transit Gateway？
    - 单区域多 VPC 连接
  - 确定您是否需要隔离网络以满足法规要求。
  - 使用适当大小的 CIDR 块创建 VPC，以满足您当前和未来的需求。

  - 如果您的增长预测不明朗，则可能需要偏向更大的 CIDR 块，以减少未来重新配置的可能性
- 在双堆栈 VPC 中，考虑为子网使用 IPv6 寻址 。IPv6 非常适合用于这样的私有子网：包含大量临时实例集或临时容器集，因此需要大量 IPv4 地址。

资源

相关的 Well-Architected 最佳实践：

- REL02-BP05 在互相连接的所有私有地址空间中强制实施非重叠的私有 IP 地址范围

相关文档：

- APN 合作伙伴：可帮助您规划联网的合作伙伴
- 适用于网络基础设施的 AWS Marketplace
- Amazon Virtual Private Cloud 连接选项白皮书
- 多数据中心 HA 网络连接
- 单区域多 VPC 连接
- 什么是 Amazon VPC？
- AWS 上的 IPv6
- 参考架构上的 IPv6
- Amazon Elastic Kubernetes Service 启动 IPv6 支持

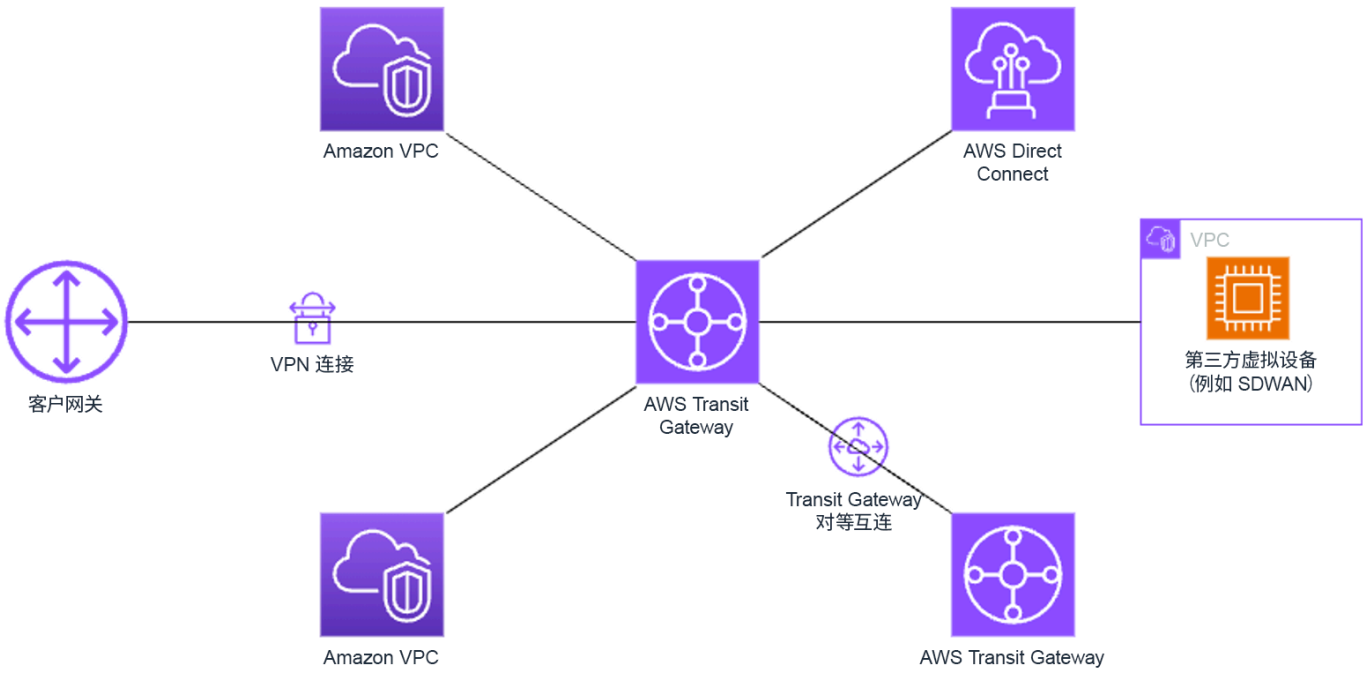
相关视频：

- AWS re:Invent 2018：Amazon VPC 的高级 VPC 设计和新功能（NET303）
- AWS re:Invent 2019：适用于众多 VPC 的 AWS Transit Gateway 参考架构（NET406-R1）
- AWS re:Invent 2023：AWS 准备好迎接下一步发展 设计具备扩展性和灵活性的网络（NET310）

REL02-BP04 轴辐式拓扑优先于多对多网格

连接多个私有网络时，例如连接虚拟私有云（VPC）与本地网络，应优先选择轴辐式拓扑而不是网格拓扑。在网格拓扑中，每个网络直接连接到其他网络，这会增加复杂性和管理开销，而轴辐式拓扑则不同，这种拓扑架构通过单个中心枢纽来集中连接。这种集中式方法简化了网络结构，并且可以增强可操作性、可扩展性和控制能力。

AWS Transit Gateway 是一项托管服务，可扩展且能够提供高可用性，专为在 AWS 上构建轴辐式网络而设计。该服务充当网络的中心枢纽，提供网络分段、集中路由功能，并可简化与云端以及与本地环境的连接。下图说明了如何使用 AWS Transit Gateway 来构建轴辐式拓扑。



常见反面模式：

- 您在轴辐式架构中使用过于复杂的路由策略，这会降低网络效率，并且导致故障排除和主动管理工作复杂化。
- 中心枢纽内没有基于路由充分地分段，这会造成漏洞，可能导致网络遭受未经授权的访问。
- 没有认真地进行优化，通过中心枢纽的流量可能会导致更高的数据传输成本，尤其是跨可用区和区域的流量。有效的流量管理策略对于控制开支至关重要。

建立此最佳实践的好处：随着所连接网络数量的增加，网格拓扑连接的管理和扩展会越来越困难。AWS Transit Gateway 为构造和运营轴辐式拓扑提供可扩展且可靠的托管式中心枢纽。使用 AWS Transit Gateway 时，您可以对多个网络建立连接，并集中管理跨这些网络的流量路由。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

- 规划您的网络。

- 创建 AWS Transit Gateway。
- 连接 VPC。
- 根据需要，创建 VPN 连接或 Direct Connect 网关，并将其关联到 Transit Gateway。
- 通过配置 Transit Gateway 路由表，定义如何在连接的 VPC 和其他连接之间路由流量。
- 使用 Amazon CloudWatch 监控并根据需要调整配置，以优化性能和成本。

资源

相关文档：

- What Is a Transit Gateway?
- 构建可扩展且安全的多 VPC AWS 网络基础设施
- Building a global network using AWS Transit Gateway Inter-Region peering
- Amazon Virtual Private Cloud Connectivity Options
- AWS 合作伙伴：可帮助您规划联网的合作伙伴
- 适用于网络基础设施的 AWS Marketplace

相关视频：

- AWS re:Invent 2023 - AWS networking foundations
- AWS re:Invent 2023 - Advanced VPC designs and new capabilities

REL02-BP05 在互相连接的所有私有地址空间中强制实施非重叠的私有 IP 地址范围

当多个 VPC 对等连接、通过 Transit Gateway 连接或者通过 VPN 连接时，各个 VPC 的 IP 地址范围不得重叠。请避免 VPC 与本地环境之间或者与所使用的其它云提供商之间出现 IP 地址冲突。您还必须能够在需要时分配私有 IP 地址范围。IP 地址管理（IPAM）系统有助于实现这一操作的自动化。

期望结果：

- VPC、本地环境或其它云提供商之间不存在 IP 地址范围冲突。
- 适当的 IP 地址管理可以更轻松地扩展网络基础设施，来适应不断增长和变化的网络需求。

常见的反面模式：

- 在 VPC 中使用与本地、企业网络或者其它云提供商相同的 IP 范围。
- 不追踪用于部署工作负载的 VPC 的 IP 范围。
- 依赖手动 IP 地址管理流程，例如电子表格。
- CIDR 块过大或过小，这往往会导致 IP 地址浪费或地址空间不足以容纳您的工作负载。

建立此最佳实践的好处：主动规划网络可确保您不会遇到互连网络中相同 IP 地址多次出现的情况。这可防止使用不同应用程序的工作负载部分出现路由问题。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

使用 IPAM（例如 Amazon VPC IP Address Manager）来监控和管理您的 CIDR 使用情况。AWS Marketplace 也提供了几个 IPAM。评估您在 AWS 上的可能使用量、将 CIDR 范围添加到现有 VPC，并且在创建 VPC 时要考虑到计划的使用量增长情况。

实施步骤

- 捕获当前的 CIDR 使用量数据（例如，VPC 和子网）。
  - 使用服务 API 操作收集当前的 CIDR 使用量数据。
  - 使用 Amazon VPC IP Address Manager 来发现资源。
- 捕获您当前的子网使用量数据。
  - 使用服务 API 操作在每个区域中按 VPC 收集子网。
  - 使用 Amazon VPC IP Address Manager 来发现资源。
- 记录当前使用量数据。
- 确定是否创建了任何重叠的 IP 范围。
- 计算备用容量。
- 确定重叠的 IP 范围。您可以迁移到新的地址范围，或者如果您需要连接重叠的范围，也可以考虑使用私有 NAT 网关或 AWS PrivateLink 等技术。

资源

相关最佳实践：

- 保护网络

相关文档：

- AWS 合作伙伴：可帮助您规划联网的合作伙伴
- 适用于网络基础设施的 AWS Marketplace
- Amazon Virtual Private Cloud Connectivity Options 白皮书
- 多数据中心 HA 网络连接
- Connecting Networks with Overlapping IP Ranges
- Amazon VPC 是什么？
- 什么是 IPAM？

相关视频：

- AWS re:Invent 2023 - Advanced VPC designs and new capabilities
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs
- AWS re:Invent 2023 - Ready for what's next? Designing networks for growth and flexibility
- AWS re:Invent 2021 - {New Launch} Manage your IP addresses at scale on AWS

工作负载架构

可靠的工作负载始于前期的软件和基础设施设计决策。您的架构选择将影响所有六个 Well-Architected 支柱的工作负载行为。针对可靠性，您必须遵循特定的模式。

以下各节将介绍使用这些保证可靠性的模式时要遵循的最佳实践。

主题

- 设计您的工作负载服务架构
- 在分布式系统中设计交互以预防发生故障
- 在分布式系统中设计交互以缓解或经受住故障的影响

设计您的工作负载服务架构

使用面向服务的架构 (SOA) 或微服务架构构建高度可扩展的可靠工作负载。面向服务的架构 (SOA) 可通过服务接口使软件组件可重复使用。微服务架构则进一步让组件变得更小、更简单。

以服务为导向的架构 (SOA) 接口采用常见的通信标准，以便快速地融入到新的工作负载。SOA 取代了构建整体架构的做法，后者由相互依赖、不可分割的单元组成。

AWS 一直采用 SOA，但现在，我们会使用微服务构建我们的系统。虽然微服务有许多具有吸引力的特性，但就可用性而言，最重要的好处在于规模更小、更简单。它们可让您区分不同服务要求的可用性，从而更明确地专注于投资具有最大可用性需求的微服务。例如，要在 Amazon.com 上提供产品信息页面（“详情页面”），需要调用数百个微服务来构建页面的不同部分。虽然一定有一些服务可用于提供价格和产品详情，但如果服务不可用，页面上的绝大多数内容都可以直接排除在外。甚至不需要提供照片和评论等内容，客户也可以购买产品。

最佳实践

- REL03-BP01 选择如何划分工作负载
- REL03-BP02 构建专注于特定业务领域和功能的服务
- REL03-BP03 根据 API 提供服务合同

REL03-BP01 选择如何划分工作负载

在确定应用程序的弹性要求时，工作负载划分很重要。应尽可能避免使用整体式架构。相反，应仔细考虑哪些应用程序组件可以分解为多项微服务。根据您的应用程序要求，最终应尽可能采用服务导向型架构（SOA）与微服务组合的形式。能够实现无状态的工作负载更容易部署为微服务。

期望结果： 工作负载应该可支持、可扩展，并尽可能地松散耦合。

在选择如何划分工作负载时，要权衡其优点和复杂性。适用于即将首次发布的新产品的功能有别于从一开始就构建用于扩展的工作负载的需求。重构一个现有的整体架构时，您需要考虑应用程序对无状态分解的支持程度。通过将服务分解为较小的部分，可以让职责明确的小型团队来开发和管理它们。然而，较小的服务会带来复杂性，包括可能会增加延迟，调试变得更复杂，而且加重运营负担。

常见反模式：

- 如示例所示， 微服务 Death Star 是这样一种情况：原子组件变得高度相互依赖，牵一发而动全身，使组件像一块整体一样死板而又脆弱。

建立此实践的好处：

- 更多特定分段可以提高敏捷性、组织灵活性和可扩展性。
- 减小了服务中断的影响。
- 应用程序组件可能有不同的可用性要求，可通过更加原子化的分段来满足这些要求。
- 支持工作负载的团队职责分明。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

请根据工作负载的分段方式选择架构类型。选择 SOA 或微服务架构（极少数情况下是整体架构）。即便在刚开始选择整体架构，您必须确保它是模块化的，而且由于您的产品随着采用的用户增加而扩展，它最终也可转变成为 SOA 或微服务架构。SOA 和微服务各自提供较小的区段，它们是现代可扩展和可靠架构的首选，但您需要认真权衡利弊，尤其在部署微服务架构时。

一项主要的权衡是您现在使用的是分布式计算架构，可能更难实现用户延迟要求，还增加了调试和跟踪用户交互的复杂性。您可以使用 AWS X-Ray 来帮助解决此问题。需要考虑的另一个影响是，您管理的应用程序数量增加时，运营复杂性也会随之增加，这要求部署多个相互独立组件。

整体架构、服务导向型架构和微服务架构

实施步骤

- 确定构建或重构应用程序所需的适当架构。SOA 和微服务分别提供较小分段，这是现代可扩展的可靠架构的首选。要在实现较小分段的同时避免一些微服务复杂性，SOA 是很好的折中方案。有关更多详细信息，请参阅 微服务利弊权衡.
- 如果您的工作负载适合，并且您的组织可以支持，则应使用微服务架构来实现最佳敏捷性和可靠性。有关更多详细信息，请参阅 在 AWS 上实施微服务。
- 考虑遵循 Strangler Fig 模式， 将整体架构重构为较小的组件。这包括逐步用新的应用程序和服务替换特定的应用程序组件。AWS Migration Hub Refactor Spaces 作为增量重构的起点。有关更多详细信息，请参阅 使用绞杀者模式无缝迁移本地传统工作负载。
- 实施微服务可能需要采用一种服务发现机制，让这些分布式服务能够相互通信。AWS App Mesh 可用于服务导向型架构，以实现可靠的发现和服务访问。 AWS Cloud Map 也可用于动态的基于 DNS 的服务发现。
- 如果您要从整体架构迁移到 SOA，Amazon MQ 可作为服务总线来帮助弥合这一差距（在云中重新设计传统应用程序时）。
- 对于具有单个共享数据库的现有整体架构，请选择将数据重组为较小分段的方式。可以按业务部门、访问模式或数据结构来划分。在重构过程的这一阶段，应选择是使用关系型还是非关系型（NoSQL）数据库来继续操作。有关更多详细信息，请参阅 从 SQL 到 NoSQL.

实施计划的工作量级别： 高

资源

相关最佳实践：

- REL03-BP02 构建专注于特定业务领域和功能的服务

相关文档：

- Amazon API Gateway：使用 OpenAPI 配置 REST API
- 什么是服务导向型架构？
- 边界上下文（领域驱动设计的中心模式）
- 在 AWS 上实施微服务
- 微服务利弊权衡
- 微服务 – 一个全新架构术语的定义
- AWS 上的微服务
- 什么是 AWS App Mesh？

相关示例：

- 迭代应用程序现代化研讨会

相关视频：

- 利用 AWS 上的微服务实现卓越

REL03-BP02 构建专注于特定业务领域和功能的服务

服务导向型架构（SOA）采用按业务需求定义的、划分明确的功能来定义服务。微服务使用域模型和限界上下文，沿业务环境边界划定服务边界。通过将重点放在业务领域和功能上，有助于团队为其服务定义独立的可靠性要求。限界上下文隔离和封装业务逻辑，让团队能够更好地解释如何处理故障。

期望的结果： 工程师和业务利益相关者共同定义限界上下文，并使用它们将系统设计为实现特定业务功能的服务。这些团队使用事件风暴等既定的实践来定义需求。新的应用程序被设计为服务，具有明确定义的边界并采用松耦合。现有的整体式架构被分解为 限界上下文 ，而系统设计转向 SOA 或微服务架构。重构整体式架构时，会应用气泡上下文和整体式架构分解模式等既定方法。

面向领域的服务作为一个或多个进程执行，彼此之间不分享状态。这些进程独立应对需求的波动，并根据特定领域的要求处理故障情景。

常见反模式：

- 围绕特定技术领域（例如 UI 和 UX、中间件或数据库）组建团队，而不是根据特定的业务领域组建。
- 应用程序进行了领域职责划分。跨越限界上下文的服务可能更难于维护，需要更多测试工作，并且需要多个领域团队参与软件更新。
- 领域依赖关系（例如领域实体库）在服务之间共享，因此对一个服务领域进行更改也需要更改其他服务领域
- 服务合同和业务逻辑没有使用通用且一致的领域语言来描述实体，这会导致翻译层的存在，使得系统更加复杂并增加调试工作量。

建立此最佳实践的好处： 应用程序被设计为独立的服务，按照业务领域确定界限，并使用通用的业务语言。服务可独立测试和部署。对于所实施的领域，服务满足特定于领域的弹性要求。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

领域驱动型决策（DDD）是围绕业务领域设计和构建软件的基本方法。围绕业务领域构建服务时，使用现有框架会很有帮助。在使用现有的整体式应用程序时，您可以利用分解模式提供成熟的技术，将应用程序现代化改造为服务。

领域驱动型决策

实施步骤

- 团队可以举办 事件风暴 研讨会，以简便的便签格式，快速确定事件、命令、聚合和领域。
- 在领域上下文中构建领域实体和功能后，您可以使用 限界上下文将领域划分为服务，具有相似功能和属性的实体分组在一起。通过将模型细分为不同的上下文，即可得到如何确定微服务边界的模板。

    - 以 Amazon.com 网站为例，实体可能包括包装、配送、时间表、价格、折扣和货币。
    - 包装、配送和时间表分组到运输上下文中，而价格、折扣和货币分组到定价上下文中。
- 将整体式架构分解为微服务 概述了重构微服务的模式。使用按照业务功能、子领域或事务进行分解的模式，与领域驱动型方法非常吻合。
- 利用 气泡上下文 等战术性技巧，您可以在现有或旧版应用程序中引入 DDD，无需预先重新编写和承诺完全转向 DDD。在气泡上下文方法中，使用服务映射和协调来建立小型限界上下文，或者建立 防护层，该层保护新定义的领域模型免受外部影响。

在团队进行了领域分析并定义实体和服务合同后，他们可以利用 AWS 服务，将自己的领域驱动型设计作为云端服务来实施。

- 通过定义执行领域业务规则的测试来开始开发。测试驱动型开发（TDD）和行为驱动型开发（BDD），有助于团队将服务重点放在解决业务问题上。
- 选择最能满足您的业务领域要求的 AWS 服务 以及 微服务架构：
    - AWS 无服务器 让您的团队可以将精力集中于具体的领域逻辑上，而不是管理服务器和基础设施。
    - AWS 上的容器 可简化基础设施的管理，这样您便可以将精力集中于领域需求方面。
    - 专用数据库 有助于您将领域需求与最适合的数据库类型相匹配。
- 在 AWS 上构建六边形架构 中概述了一个框架，从业务领域出发，采用逆向工作方法，将业务逻辑构建到服务中，以满足功能需求，然后连接集成适配器。采用将接口详细信息从业务逻辑与 AWS 服务之间分离的模式，让团队能够专注于研究领域功能并提高软件质量。

资源

相关最佳实践：

- REL03-BP01 选择如何划分工作负载
- REL03-BP03 根据 API 提供服务合同

相关文档：

- AWS 微服务
- 在 AWS 上实施微服务
- 如何将整体式架构分解为多项微服务
- 在被遗留系统包围时通过 DDD 开始着手

- 领域驱动型设计：解决软件核心的复杂性
- 在 AWS 上构建六边形架构
- 将整体式架构分解为微服务
- 事件风暴
- 限界上下文之间的消息
- 微服务
- 测试驱动型开发
- 行为驱动型开发

相关示例：

- 企业云原生研讨会
- 在 AWS 上设计云原生微服务（来源：DDD/EventStormingWorkshop）

相关工具：

- AWS Cloud 数据库
- AWS 上的无服务器
- AWS 上的容器

REL03-BP03 根据 API 提供服务合同

服务合同是 API 生产者与使用者之间的书面协议，采用机器可读的 API 定义形式进行定义。合同版本控制策略让使用者能够继续使用现有的 API，并在更新的 API 准备就绪时，将其应用程序迁移到更新的 API。只要遵守合同，生产者部署可随时进行。服务团队可以使用自己选择的技术堆栈来满足 API 合同要求。

期望的结果：

常见反模式： 使用服务导向型架构或微服务架构所构建的应用程序能够独立运行，但具有集成的运行时系统依赖项。当双方都遵循共同的 API 合同时，向 API 使用者或生产者部署更改并不会影响整个系统的稳定性。通过服务 API 进行通信的组件能够独立执行功能发布、升级到运行时系统依赖项或者失效转移到灾难恢复（DR）站点，而彼此之间的影响很小，或者根本没有影响。此外，离散服务能够独立扩展来满足资源需求，无需统一扩展其他服务。

- 创建不使用强类型架构的服务 API。这样， API 不能用来生成无法通过编程方式验证的 API 绑定和有效负载。
- 不采用版本控制策略，会迫使 API 使用者在服务合同变化时进行更新和发布，否则会出现故障。
- 错误消息会泄露基础服务的实施细节，而不是按照域环境和语言描述集成故障。
- 不使用 API 合同开发测试用例和模拟 API 实施，以便对服务组件进行独立测试。

建立此最佳实践的好处： 分布式系统由通过 API 服务合同进行通信的组件组成，可以提高可靠性。开发人员可以在开发过程的早期发现潜在问题，在编译期间进行类型检查，来验证请求和响应是否符合 API 合同以及是否存在必填字段。API 合同为 API 提供了清晰的自描述接口，并在不同的系统和编程语言之间提供了更好的互操作性。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

确定业务领域并确定工作负载划分后，即可开发服务 API。首先，为 API 定义机器可读的服务合同，然后实施 API 版本控制策略。在准备好通过 REST、GraphQL 等常见协议或异步事件来集成服务时，您可以将 AWS 服务整合到架构中，从而将您的组件与强类型的 API 合同集成。

面向服务 API 合同的 AWS 服务

将 AWS 服务（包括 Amazon API Gateway、AWS AppSync和 Amazon EventBridge ）等整合到架构中，以便在您的应用程序中使用 API 服务合同。使用 Amazon API Gateway 有助于直接与原生 AWS 服务及其他 Web 服务集成。API Gateway 支持 OpenAPI 规范 和版本控制。AWS AppSync 是托管的 GraphQL 端点，在配置该端点时，您需要定义 GraphQL 架构，进而为查询、突变和订阅定义服务接口。Amazon EventBridge 使用事件架构来定义事件并为您的事件生成代码绑定。

实施步骤

- 首先，为您的 API 定义一个合同。合同将说明 API 的功能，并为 API 的输入和输出定义强类型的数据对象和字段。
- 在 API Gateway 中配置 API 时，您可以导入和导出端点的 OpenAPI 规范。
  - 导入 OpenAPI 定义 可简化 API 的创建过程，并可与 AWS 基础设施即代码工具（例如 AWS Serverless Application Model 和 AWS Cloud Development Kit (AWS CDK)）集成。
  - 导出 API 定义 可简化与 API 测试工具的集成，并为服务使用者提供集成规范。
- 要使用 AWS AppSync 定义和管理 GraphQL API，您可以 定义 GraphQL 架构 文件来生成合同接口，并简化与复杂的 REST 模型、众多数据库表或传统服务的交互。

- AWS Amplify 项目与 AWS AppSync 集成后，会生成强类型的 JavaScript 查询文件供应用程序使用，还会生成 AWS AppSync GraphQL 客户端库供 Amazon DynamoDB 表使用。
- 当您使用来自 Amazon EventBridge 的服务事件时，事件遵循架构注册表中已有的架构或您使用 OpenAPI 规范定义的架构。通过在注册表中定义架构，您还可以从架构合同生成客户端绑定，以将代码与事件集成。
- 扩展 API 或者实施 API 版本控制。在添加可以配置为可选字段的字段时，或者为必填字段添加默认值时，扩展 API 是一种相对比较简单的选项。
  - 对于 REST 和 GraphQL 等协议，基于 JSON 的合同可能非常适合合同扩展。
  - 对于 SOAP 等协议，基于 XML 的合同应与服务使用者一起进行测试，来确定合同扩展的可行性。
- 对 API 进行版本控制时，可以考虑实施代理版本控制，其中使用 Facade 模式来支持版本，这样就能够在单个代码库中维护逻辑。
  - 通过 API Gateway，您能够使用 请求和响应映射 ，通过建立 Facade 模式为新字段提供默认值，或者从请求或响应中除去已删除的字段，从而简化接受合同变更的过程。通过这种方法，底层服务可以维护单个代码库。

资源

相关最佳实践：

- REL03-BP01 选择如何划分工作负载
- REL03-BP02 构建专注于特定业务领域和功能的服务
- REL04-BP02 实施松耦合的依赖关系
- REL05-BP03 控制与限制重试调用
- REL05-BP05 设置客户端超时

相关文档：

- 什么是 API（应用程序编程接口）？
- 在 AWS 上实施微服务
- 微服务利弊权衡
- 微服务 – 一个全新架构术语的定义
- AWS 上的微服务
- 使用 OpenAPI 的 API Gateway 扩展

- OpenAPI 规范
- GraphQL：架构和类型
- Amazon EventBridge 代码绑定

相关示例：

- Amazon API Gateway：使用 OpenAPI 配置 REST API
- 从 Amazon API Gateway 到使用 OpenAPI 的 Amazon DynamoDB CRUD 应用程序
- 无服务器时代的现代化应用程序集成模式：API Gateway 服务集成
- 通过 Amazon CloudFront 实施基于标头的 API Gateway 版本控制
- AWS AppSync：构建客户端应用程序

相关视频：

- 在 AWS SAM 中使用 OpenAPI 来管理 API Gateway

相关工具：

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

在分布式系统中设计交互以预防发生故障

分布式系统依赖于通信网络实现组件（例如服务器或服务）的互联。尽管这些网络中存在数据丢失或延迟，但是您的工作负载必须可靠运行。分布式系统组件的运行方式不得对其他组件或工作负载产生负面影响。这些最佳实践能够预防故障，并改善平均故障间隔时间（MTBF）。

最佳实践

- REL04-BP01 确定您依赖的分布式系统的类型
- REL04-BP02 实施松耦合的依赖关系
- REL04-BP03 持续工作
- REL04-BP04 使所有响应幂等

REL04-BP01 确定您依赖的分布式系统的类型

分布式系统可以是同步系统、异步系统或批处理系统。同步系统必须尽可能快地处理请求，并使用 HTTP/S、REST 或远程过程调用（RPC，Remote Procedure Call）协议，同步地发出请求和响应调用，来彼此通信。异步系统在彼此通信时通过中间服务来异步交换数据，无需将各个系统耦合在一起。批处理系统则会接收大量输入数据，无需人工干预即可运行自动数据处理，并生成输出数据。

期望结果：设计可有效与同步、异步和批处理依赖项交互的工作负载。

常见反面模式：

- 工作负载无限期地等待其依赖项的响应，这可能导致工作负载客户端超时，不知道其请求是否已被接收。
- 工作负载使用会同步调用彼此的依赖系统链。这种模式要求每个系统都可用并能成功处理请求，整个链才能成功运行，这导致很容易出现崩溃行为，影响到整体可用性。
- 工作负载与其依赖项异步通信，并依赖于保证消息传递且仅传递一次的概念，但仍然会经常收到重复的消息。
- 工作负载没有使用正确的批处理调度工具，导致允许并行执行相同的批处理作业。

建立此最佳实践的好处：对于给定工作负载而言，实施同步、异步和批处理中的某种或多种通信方式是很常见的情况。此最佳实践可帮助您确定，选择每种通信方式所面对的不同权衡，以便让您的工作负载能够承受其任意依赖项中断所带来的干扰。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

以下各节针对每种依赖项，介绍了一般性实施指南和具体实施指南。

一般指导

- 确保您的依赖项提供的性能和可靠性服务级别目标（SLO，Service-Level Objective），能够满足工作负载的性能和可靠性要求。
- 使用 AWS 可观测性服务监控响应时间和错误率，确保您的依赖项提供的服务级别能够满足工作负载的需求。
- 确定您的工作负载在与其依赖项进行通信时，可能会遇到的潜在挑战。分布式系统会遇到各种各样的挑战，这些挑战可能会增加架构的复杂性、运营负担和成本。常见的挑战包括延迟、网络中断、数据丢失、可扩展性和数据复制延迟。

- 实施可靠的错误处理和日志记录措施，帮助您在依赖项遇到问题时解决问题。

同步依赖项

在同步通信中，您的工作负载向其依赖项发送请求并停止操作，来等待响应。当其依赖项收到请求时，依赖项会尝试尽快处理请求并将响应发送回工作负载。同步通信面临的一个巨大挑战是它会导致时间耦合，这要求您的工作负载及其依赖项同时可用。当您的工作负载需要与其依赖项以同步方式通信时，请考虑以下指南：

- 工作负载在执行单个功能时，不应依赖多个同步依赖项。这种依赖项链会导致整体更加脆弱，因为要想完成请求，路径中的所有依赖项都必须可用。
- 请确定当依赖项运行不正常或不可用时，您采用的错误处理和重试策略。避免使用双模态行为。双模态行为是指工作负载在正常模式和故障模式下表现出不同的行为。有关双模态行为更多详细信息，请参阅 REL11-BP05 使用静态稳定性来防止双模态行为。
- 请记住，快速失效机制比让工作负载等待要好。例如，AWS Lambda Developer Guide 描述了在调用 Lambda 函数时如何处理重试和失败。
- 设置工作负载调用其依赖项时的超时。这种技术可以避免等待太长时间或无限期等待响应。有关此问题的有用讨论，请参阅 Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications。
- 在完成单个请求时，尽可能减少从工作负载对依赖项进行的调用次数。在它们之间进行的频繁调用会增加耦合和延迟。

异步依赖项

要想临时解除工作负载与其依赖项之间的时间耦合关系，就应该采取异步通信。使用异步方法，您的工作负载无需等待其依赖项或依赖项链发送响应，即可继续进行任何其他处理。

当您的工作负载需要与其依赖项以异步方式通信时，请考虑以下指南：

- 根据您的应用场景和要求，确定是使用消息收发还是事件流。消息收发让工作负载可以通过消息代理来发送和接收消息，以此来与其依赖项通信。事件流则让工作负载及其依赖项可以使用流服务来发布和订阅事件，这些事件以连续数据流的形式交付，需要尽快处理。

- 消息收发和事件流以不同的方法来处理消息，因此您需要根据以下因素作出权衡决策：
  - 消息优先级：消息代理可以先处理高优先级消息，然后再处理普通消息。在事件流中，所有消息具有相同的优先级。

  - 消息使用：消息代理确保使用器收到消息。事件流使用器必须跟踪所读取的每一条消息。
  - 消息排序：对于消息收发，除非您使用先进先出（FIFO，First-In-First-Out）方法，否则无法保证完全按照发送的顺序接收消息。事件流则始终保留数据生成的顺序。
  - 消息删除：对于消息收发，使用器必须在处理消息后将其删除。事件流服务则将消息附加到流并一直保留在流中，直到消息的保留期到期。这种删除策略让事件流非常适合重放消息。
- 定义如何让工作负载在其依赖项完成工作时了解这一信息。例如，当您的工作负载异步调用 Lambda 函数时，Lambda 将事件置于队列中并返回成功响应，没有额外的信息。处理完成后，Lambda 函数可以将结果发送到目标，根据结果是成功还是失败，可以配置不同的目标。
- 利用幂等性，构建工作负载以处理重复的消息。幂等性意味着，即使您的工作负载针对同一消息多次生成结果，这些结果也会保持不变。需要指出的是，如果发生网络故障或未收到确认，消息收发或流服务将重新传送消息。
- 如果您的工作负载没有从其依赖项获得响应，则需要重新提交请求。请考虑限制重试次数，以保留工作负载的 CPU、内存和网络资源用于处理其他请求。AWS Lambda 文档介绍了如何处理异步调用的错误。
- 利用合适的可观测性、调试和跟踪工具，来管理和操作工作负载与其依赖项的异步通信。您可以使用 Amazon CloudWatch 来监控消息收发和事件流服务。您还可以使用 AWS X-Ray 检测工作负载，以快速获取见解用于对问题进行故障排除。

批处理依赖项

批处理系统获取输入数据，启动一系列作业来处理数据，然后生成一些输出数据，整个过程无需人工干预。根据数据大小，作业的运行时间可能只需要几分钟，而在某些情况下，也可能长达数天。当您的工作负载与其批处理依赖项通信时，请考虑以下指南：

- 定义工作负载应运行批处理作业的时间窗口。您的工作负载可以设置重复模式来调用批处理系统，例如，每小时或每月月底。
- 确定数据输入的位置，以及处理后数据输出的位置。选择可以让您的工作负载大规模读取和写入文件的存储服务，例如 Amazon Simple Storage Services（Amazon S3）、Amazon Elastic File System（Amazon EFS） 和 Amazon FSx for Lustre。
- 如果您的工作负载需要调用多个批处理作业，则可以利用 AWS Step Functions 来简化在 AWS 或本地运行的批处理任务的编排。此示例项目演示了使用 Step Functions、AWS Batch 和 Lambda 编排批处理作业。
- 监控批处理作业以发现异常情况，例如作业完成用时超过了应有的时间。您可以使用 CloudWatch Container Insights 等工具来监控 AWS Batch 环境和作业。在这种情况下，工作负载会从头停止下一个作业，并向相关人员通知异常情况。

资源

相关文档：

- AWS Cloud运维：监控和可观测性
- Amazon Builders' Library：分布式系统相关挑战
- REL11-BP05 使用静态稳定性来防止双模态行为
- AWS Lambda 开发人员指南：AWS Lambda 中的错误处理和自动重试
- Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications
- AWS Messaging
- 什么是流数据？
- AWS Lambda 开发人员指南：异步调用
- Amazon Simple Queue Service 常见问题：FIFO 队列
- Amazon Kinesis Data Streams Developer Guide: Handling Duplicate Records
- Amazon Simple Queue Service Developer Guide: Available CloudWatch metrics for Amazon SQS
- Amazon Kinesis Data Streams Developer Guide: Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch
- AWS X-Ray Developer Guide: AWS X-Ray concepts
- AWS Samples on GitHub: AWS Step functions Complex Orchestrator App
- AWS Batch User Guide: AWS Batch CloudWatch Container Insights

相关视频：

- AWS Summit SF 2022 - Full-stack observability and application monitoring with AWS (COP310)

相关工具：

- Amazon CloudWatch
- Amazon CloudWatch Logs
- AWS X-Ray
- Amazon Simple Storage Services（Amazon S3）
- Amazon Elastic File System（Amazon EFS）
- Amazon FSx for Lustre

- AWS Step Functions
- AWS Batch

REL04-BP02 实施松耦合的依赖关系

队列系统、流系统、工作流和负载均衡器等依赖关系是松耦合的。松耦合有助于隔离某个组件的行为与依赖于它的其他组件的行为，从而提升韧性和敏捷性。

在紧耦合的系统中，更改一个组件可能导致需要更改依赖该组件的其他组件，从而使所有组件的性能均降低。松耦合会打破这种依赖关系，使存在依赖关系的组件只需了解经过版本控制而且已发布的接口。在依赖项之间实施松耦合将隔离一个组件中的故障，防止对其他组件造成影响。

松耦合允许您修改代码或向组件添加功能，同时最大限度地降低依赖于该组件的其他组件的风险。它还允许在组件级别实现精细的韧性，您可以横向扩展或甚至改变依赖关系的底层实现。

要通过松耦合进一步提升韧性，在可能的情况下采用异步组件交互。若确定对请求进行注册已足够，则此模型适用于无需立即响应的任何交互。它包含一个生成事件的组件和另外一个使用事件的组件。两个组件不会通过直接点对点交互，但通常经由中间持久存储层集成，例如，Amazon SQS 队列或诸如 Amazon Kinesis 或 AWS Step Functions 流数据平台。

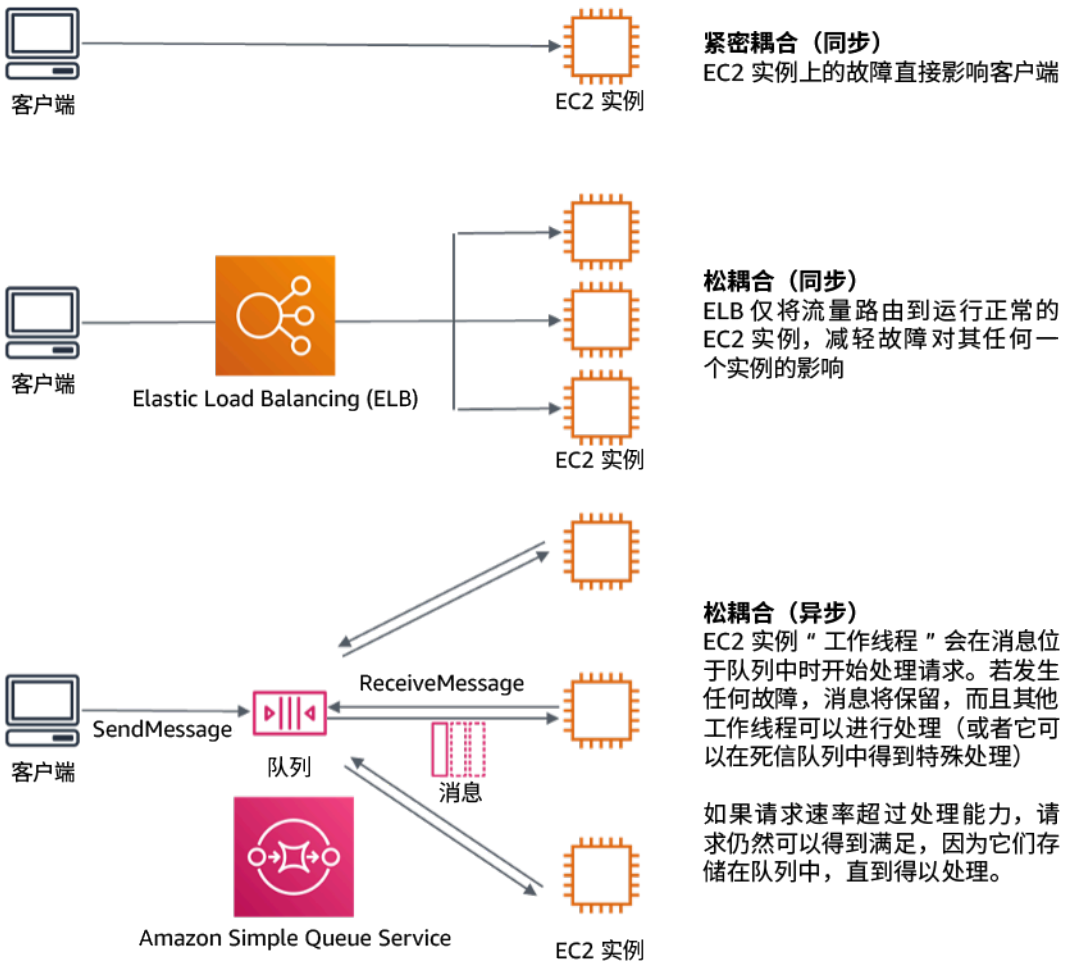


图 4：队列系统和负载均衡器等依赖关系是松耦合的。

Amazon SQS 队列和 Elastic Load Balancer 只是为松耦合增加中间层的两种方式。您还可以使用 Amazon EventBridge 在 AWS Cloud 中构建事件驱动型架构，而前者可从其依赖的服务（事件使用器）中提取客户端（事件产生器）。当您需要进行高吞吐量、基于推送的多对多消息收发时，Amazon Simple Notification Service（Amazon SNS）是可供选择的高效解决方案。通过 Amazon SNS 主题，您的发布者系统可以呈扇形将消息分发到大量订阅者端点以便进行并行处理。

虽然队列具有多项优点，但在大多数硬性实时系统中，早于阈值时间（通常为秒）的请求应被视为过时（客户端已放弃而且不再等待响应）而不被处理。因此，较新（而且可能依然有效）的请求会被处理。

预期结果：实现松耦合的依赖关系可以最大限度地减少组件级别的故障范围，这有助于诊断和解决问题。它还简化了开发周期，允许团队在模块级别实施更改，而不会影响依赖它的其他组件的性能。这种方法能够根据资源需求以及有助于提高成本效益的组件利用率，在组件层面进行扩展。

常见的反面模式：

- 部署整体式工作负载。
- 直接在工作负载层之间调用 API，不具备失效转移或异步处理请求的功能。
- 使用共享数据进行紧密耦合。松耦合的系统应避免通过共享数据库或其他形式的紧密耦合数据存储共享数据，这可能会重新引入紧耦合并阻碍可扩展性。
- 忽略背压。当组件无法以相同速度处理传入数据时，您的工作负载应该能够减慢或停止传入数据。

建立此最佳实践的好处：松耦合有助于隔离来自其他依赖于它的组件的行为，从而提升韧性和敏捷性。组件中的故障相互隔离。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

实施松耦合的依赖关系。可利用多种解决方案来构建松耦合的应用程序。其中包括用于实施全面托管的队列、自动化工作流、对事件的反应以及 API 等内容的服务，这些服务有助于将组件的行为相互隔离，从而提高韧性和敏捷性。

- 构建事件驱动型架构：Amazon EventBridge 有助于您构建松耦合的分布式事件驱动型架构。
- 在分布式系统中实现队列：可以使用 Amazon Simple Queue Service（Amazon SQS）来集成和解耦分布式系统。
- 将组件容器化为微服务：利用微服务，团队可以构建由小型独立组件组成的应用程序，这些组件通过明确定义的 API 进行通信。Amazon Elastic Container Service（Amazon ECS）和 Amazon Elastic Kubernetes Service（Amazon EKS）可以帮助您更快地开始使用容器。
- 使用 Step Functions 管理工作流程：Step Functions 有助于您将多种 AWS 服务协调成灵活的工作流程。
- 利用发布-订阅 (pub/sub) 消息架构：Amazon Simple Notification Service（Amazon SNS）提供从发布者到订阅者（也称为产生器和使用器）的消息传输。

实施步骤

- 事件驱动型架构中的组件由事件启动。事件是系统中发生的操作，例如用户将商品添加到购物车。操作成功后，将生成一个激活系统的下一个组件的事件。
  - 使用 Amazon EventBridge 构建事件驱动型应用程序
  - AWS re: Invent 2022 - 使用 Amazon EventBridge 设计事件驱动型集成
- 分布式消息系统主要有三个部分，需要为基于队列的架构实现这些部分。它们包括分布式系统的组件、用于解耦的队列（分布在 Amazon SQS 服务器上）以及队列中的消息。典型的系统有将消息发

送到队列中的产生器和从队列接收消息的使用器。该队列将消息存储在多台 Amazon SQS 服务器上以实现冗余。

  - 基本 Amazon SQS 架构
  - 使用 Amazon Simple Queue Service 在分布式应用程序之间发送消息
- 由于松耦合的组件由独立团队管理，因此微服务如果得到充分利用，可以增强可维护性并提高可扩展性。它还允许在发生变化时将行为隔离到单个组件。
  - 在 AWS 上实施微服务
  - 让我们来构造！ 使用容器构造微服务
- 借助 AWS Step Functions，您可以构建分布式应用程序、实现流程自动化、编排微服务等。将多个组件编排到一个自动化工作流程中，让您可以解耦应用程序中的依赖关系。
  - 使用 AWS Step Functions 和 AWS Lambda 创建无服务器工作流程
  - 开始使用 AWS Step Functions

资源

相关文档：

- Amazon EC2：确保幂等性
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：可靠性、持续工作和安然无忧
- 什么是 Amazon EventBridge？
- 什么是 Amazon Simple Queue Service？
- 拆分整体式应用程序
- 使用 AWS Step Functions 和 Amazon SQS 协调基于队列的微服务
- 基本 Amazon SQS 架构
- 基于队列的架构

相关视频：

- 2019 年 AWS 纽约峰会：介绍事件驱动型架构和 Amazon EventBridge（MAD205）
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统（ARC337）（包括松耦合、持续工作和静态稳定性）
- AWS re:Invent 2019：迁移到事件驱动型架构（SVS308）

- AWS re:Invent 2019：使用 Amazon SQS 和 Lambda 的可扩展无服务器事件驱动型应用程序（API304）
- AWS re:Invent 2019：使用 Amazon SQS 和 Lambda 的可扩展无服务器事件驱动型应用程序
- AWS re: Invent 2022 - 使用 Amazon EventBridge 设计事件驱动型集成
- AWS re: Invent 2017：Elastic Load Balancing 深入探讨和最佳实践

REL04-BP03 持续工作

系统会在负载中存在剧烈快速更改时失败。例如，如果您的工作负载执行的一项运行状况检查监控着数千个服务器的运行状况，每次都应发送相同大小的有效负载（当前状态的完整快照）。无论是否有服务器或有多少服务器发生故障，运行状况检查系统都会持续工作，而不会有剧烈、快速的变动。

例如，如果运行状况检查系统正在监控 100000 个服务器，它的标称负载低于通常而言较低的服务器故障率。但如果发生重大事件使一半的服务器运行不正常，则运行状况检查系统会因为尝试更新通知系统以及向其客户端传送状态而变得不堪重负。因此，运行状况检查系统每次应发送当前状态的完整快照。100000 个服务器的运行状况，若每个以一个比特代表，则仅需要 12.5-KB 有效负载。无论是没有服务器发生故障还是所有服务器都发生故障，运行状况检查系统都会持续工作，而大幅度骤变也不会威胁到系统的稳定性。这实际上是 Amazon Route 53 处理端点（例如 IP 地址）的运行状况检查来确定最终用户如何路由到这些端点的方式。

未建立这种最佳实践的情况下暴露的风险等级： 低

实施指导

- 持续工作，使系统不会在负载出现骤变时失败。
- 实施松耦合的依赖关系。队列系统、流系统、工作流和负载均衡器等依赖关系是松耦合的。松耦合有助于隔离某个组件的行为与依赖于它的其他组件的行为，从而提升弹性和敏捷性。
  - Amazon Builders' Library：可靠性、持续工作和安然无忧
  - AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统（ARC337）（包括持续工作）
    - 例如，如果运行状况检查系统正在监控 10 万台服务器工程设计工作负载，不论成功或失败的次数，有效负载大小均能保持稳定。

资源

相关文档：

- Amazon EC2：确保幂等性
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：可靠性、持续工作和安然无忧

相关视频：

- 2019 年 AWS 纽约峰会：介绍事件驱动型架构和 Amazon EventBridge（MAD205）
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统（ARC337）（包括持续工作）
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统（ARC337）（包括松耦合、持续工作和静态稳定性）
- AWS re:Invent 2019：迁移到事件驱动型架构（SVS308）

REL04-BP04 使所有响应幂等

幂等服务承诺每个请求只完成一次，因此发起多个相同请求与进行单个请求的效果相同。幂等服务使客户端可以轻松进行重试，而不必担心错误地处理多次。要执行此操作，客户端可以发出具有幂等性令牌的 API 请求，每当重复请求时都会使用同一令牌。幂等服务 API 使用令牌返回响应，该响应与首次完成请求时返回的响应相同。

在分布式系统中，至多（客户端仅发起一个请求）或至少（持续发起请求直到客户端收到成功确认）执行某项操作一次并不难。难就难在要保证某项操作具有幂等性，亦即它被 恰好 执行一次，从而使发起多个相同的请求与发起一个请求的效果相同。在 API 中使用幂等性令牌，服务可以一次或多次收到变异请求，而不会创建重复记录或产生副作用。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

- 使所有响应幂等。幂等服务承诺每个请求只完成一次，因此发起多个相同请求与进行单个请求的效果相同。
  - 客户端可以发出具有幂等性令牌的 API 请求，每当重复请求时都会使用同一令牌。幂等服务 API 使用令牌返回响应，该响应与首次完成请求时返回的响应相同。
    - Amazon EC2：确保幂等性

资源

相关文档：

- Amazon EC2：确保幂等性
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：可靠性、持续工作和安然无忧

相关视频：

- 2019 年 AWS 纽约峰会：介绍事件驱动型架构和 Amazon EventBridge（MAD205）
- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统（ARC337）（包括松耦合、持续工作和静态稳定性）
- AWS re:Invent 2019：迁移到事件驱动型架构（SVS308）

在分布式系统中设计交互以缓解或经受住故障的影响

分布式系统依赖于通信网络以便使组件互相连接（如服务器或服务等）。尽管这些网络中存在数据丢失或延迟，但是您的工作负载必须可靠运行。分布式系统组件的运行方式不得对其他组件或工作负载产生负面影响。这些最佳实践使工作负载能够承受压力或故障，从中更快地恢复，并且降低此类伤害的影响。其结果是缩短平均恢复时间（MTTR）。

这些最佳实践能够预防故障，并改善平均故障间隔时间（MTBF）。

最佳实践

- REL05-BP01 实施轻松降级以将适用的硬依赖关系转换为软依赖关系
- REL05-BP02 限制请求
- REL05-BP03 控制与限制重试调用
- REL05-BP04 快速失效机制和限制队列
- REL05-BP05 设置客户端超时
- REL05-BP06 尽可能使系统为无状态
- REL05-BP07 实施紧急杠杆

REL05-BP01 实施轻松降级以将适用的硬依赖关系转换为软依赖关系

即使依赖项不可用，应用程序组件也应继续执行其核心功能。应用程序组件可以提供稍微陈旧的数据、替代数据，甚至没有数据。这可确保在提供核心业务价值的同时，将局部故障对整体系统功能造成的障碍减至最少。

期望的结果： 某个组件的依赖项运行不正常时，该组件仍可在性能降低的条件下运行。组件的故障模式应视为正常运行。工作流在设计时，应确保此类故障不会导致完全失败，或者至少实现可预测和可恢复的状态。

常见反模式：

- 未确定所需的核心业务功能。即使在依赖项故障期间也不测试组件是否正常运行。
- 不论是出错时，还是当多个依赖项中只有一个不可用且仍可以返回部分结果时，不提供任何数据。
- 在事务部分失败时造成不一致的状态。
- 没有替代方法用于访问中央 Parameter Store。
- 在刷新失败时，使本地状态失效或清空，而没有考虑这样做的后果。

建立此最佳实践的好处： 轻松降级可以提高整个系统的可用性，即使在故障期间也能保持最重要功能的功能。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

实施轻松降级有助于最大限度地减少依赖项故障对组件功能的影响。理想情况下，组件检测依赖项故障，并以对其他组件或客户影响最小的方式解决这些故障。

为轻松降级设计架构意味着在依赖项设计期间，需要考虑潜在的故障模式。对于每种故障模式，都要有办法向调用方或客户提供组件的大部分功能，或者至少提供最关键的功能。这些注意事项可以作为额外的要求进行测试和验证。理想情况下，即使一个或多个依赖项出现故障，一个组件也能够以可接受的方式执行其核心功能。

这既是商业议题，也是技术议题。所有业务要求都很重要，应尽可能满足。但是，确定在无法满足所有要求时会出现什么情况，这种做法同样很有意义。系统可以设计为具备可用性和一致性，但是如果必须放弃一个要求，那么哪个要求更重要？ 对于付款处理而言，可能一致性更重要。对于实时应用程序，可能可用性更重要。对于面向客户的网站，这个回答可能取决于客户的期望值。

其具体的意义取决于组件的要求，以及将什么功能视为其核心功能。例如：

- 在登录页面上，电子商务网站可能会显示来自多个不同系统的数据，例如个性化推荐、最热销的产品和客户订单状态。当一个上游系统出现故障时，合理的做法是显示其余的所有信息，而不是向客户显示一个错误页面。
- 对于执行批量写入的组件，如果某个单独的操作失败，它仍应继续执行批处理。实施重试机制应该很简单。要实施重试机制，可以向调用方返回哪些操作成功、哪些操作失败的信息，或者将失败的请求放入死信队列中来实施异步重试。同时还应记录有关失败操作的信息。
- 处理事务的系统必须确保，要么执行了所有更新，要么未执行任何更新。对于分布式事务，在同一个事务后面的操作失败时，可以使用 Sega 模式来回滚先前的操作。这里的核心功能是保持一致性。
- 时间关键型系统应能够处理未及时响应的依赖项。在这些情况下，可以使用断路器模式。当来自依赖项的响应开始超时时，系统可以切换为关闭状态，不进行额外的调用。
- 应用程序可以从 Parameter Store 中读取参数。创建具有默认参数集的容器镜像，并在 Parameter Store 不可用时使用这些镜像，这种做法会很有用。

请注意，需要对组件出现故障时所采取的途径进行测试，而且这一途径应该比主要途径简单得多。通常，应避免使用回退策略。

实施步骤

确定外部和内部依赖项。考虑它们可能出现什么样的故障。思考在故障期间，尽可能减少对上游和下游系统以及客户的负面影响的方法。

以下是依赖项列表以及在依赖项故障时如何轻松降级：

1. 依赖项部分故障： 一个组件可以向下游系统发出多个请求，这可以是向一个系统发出多个请求，也可以是向多个系统发出一个请求。根据具体的业务环境，可能需要采用不同的处理方式（有关更多详细信息，请参阅实施指南前文中的示例）。
2. 由于高负载，下游系统无法处理请求： 如果发送到下游系统的请求持续失败，则继续重试就毫无意义。这可能会对已经过载的系统造成额外负载，使得系统更难于恢复。这时可以使用断路器模式，该模式监视对下游系统的失败调用。如果大量调用失败，它会停止向下游系统发送更多请求，仅不定期让调用通过，以测试下游系统是否再次可用。
3. Parameter Store 不可用： 要转换 Parameter Store，可以使用容器或机器镜像中包含的软依赖项缓存或合理默认值。请注意，这些默认值需要保持为最新并包含在测试套件中。
4. 监控服务或其他非功能性依赖项不可用： 如果某个组件间歇性地无法将日志、指标或跟踪发送到中央监控服务，通常最好还是正常执行业务功能。长时间静默地不进行日志记录或不推送指标通常不可接受。此外，某些使用场景可能需要完整的审计条目才能满足合规性要求。

5. 关系数据库的主实例可能不可用： 与几乎所有关系数据库一样，Amazon Relational Database Service 只能有一个主写入器实例。对于写入工作负载，这会造成单点故障，并增加扩缩的难度。使用多可用区配置来实现高可用性，或者使用 Amazon Aurora 无服务器架构来实现更好的扩展能力，可以部分缓解这种情况。对于非常高的可用性要求，完全不依赖于主写入器是有意义的。对于只读查询，可以使用只读副本，这提供了冗余和横向扩展能力，而不仅仅是纵向扩展。对写入操作可以进行缓冲，例如缓冲在 Amazon Simple Queue Service 队列中，这样即使主写入器暂时不可用，仍然可以接受来自客户的写入请求。

资源

相关文档：

- Amazon API Gateway：限制 API 请求以提高吞吐量
- CircuitBreaker（对《发布它！》一书中的“断路器”部分进行的总结）
- AWS 中的错误重试和指数回退
- Michael Nygard《发布它！ 设计和部署生产就绪的软件》（Release It! Design and Deploy Production-Ready Software）
- Amazon Builders' Library：避免在分布式系统中回退
- Amazon Builders' Library：避免无法克服的队列积压
- Amazon Builders' Library：缓存挑战和策略
- Amazon Builders' Library：为超时、重试和回退引入抖动

相关视频：

- 重试、回退和抖动：AWS re:Invent 2019：介绍 Amazon Builders’ Library（DOP328）

相关示例：

- Well-Architected 实验室：第 300 级：实施运行状况检查和管理依赖项以提高可靠性

REL05-BP02 限制请求

限制请求，以防范因需求意外增加而导致的资源耗尽的情况。系统将处理未超过限制速率的请求，而超过所定义限制的请求将被拒绝，并返回一条消息，指出请求已受限制。

期望的结果： 使用请求限制可以缓解客户流量突增、泛洪攻击或重试风暴所造成的大量容量峰值情况，让工作负载能够继续正常处理支持的请求量。

常见反模式：

- 未实施 API 端点限制，或者未考虑预期容量即保留默认值。
- API 端点未经过负载测试，也未测试节流限制。
- 限制请求速率而未考虑请求大小或复杂性。
- 测试最大请求速率或最大请求大小，但未同时测试两者。
- 资源预置的限制与测试中确定的限制不同。
- 尚未为应用程序到应用程序的（A2A）API 使用者配置或考虑使用量计划。
- 横向扩展的队列使用者没有配置最大并发设置。
- 没有基于每个 IP 地址实施速率限制。

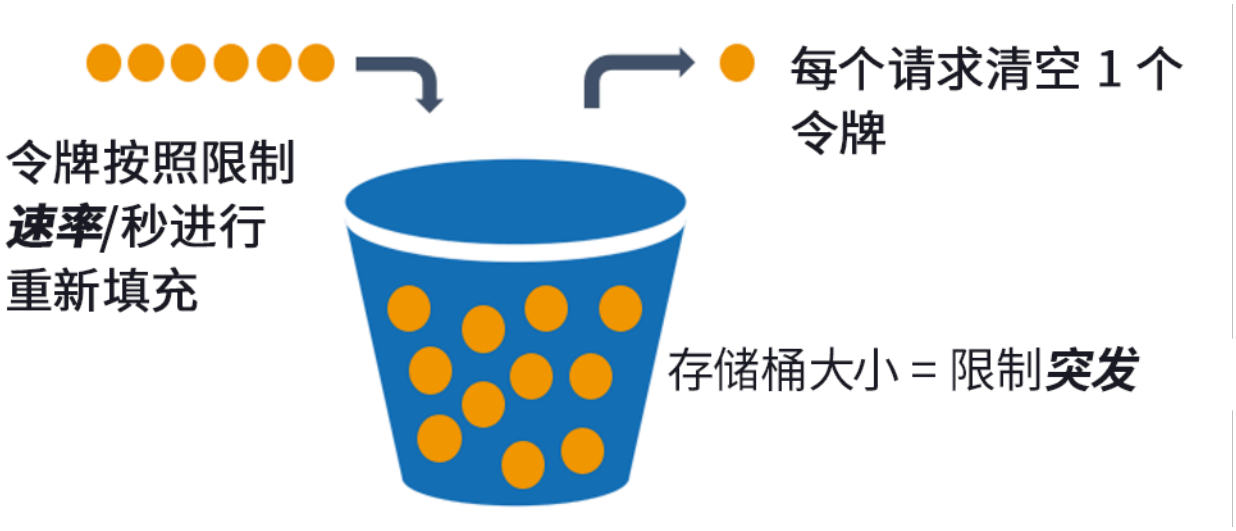
建立此最佳实践的好处： 在遇到意外的容量峰值时，设置了节流限制的工作负载能够正常运行，并成功处理已接受的请求负载。API 和队列上突然或持续出现的请求峰值会受到限制，不会耗尽请求处理资源。速率限制会限制单独的请求者，这样来自单个 IP 地址或 API 使用者的大量流量就不会耗尽资源，从而不会影响其他使用者。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

服务应设计为处理已知的请求容量；这种容量可以通过负载测试来确立。如果请求到达速率超过限制，则会发出相应的响应，表示请求已被限制。这让使用者可以处理错误并稍后重试。

当您的服务需要实施节流时，可以考虑实施令牌存储桶算法，每个令牌对应于一个请求。令牌按照每秒的限制速率重新填充，并按照每个请求一个令牌的模式异步清空。



令牌存储桶算法。

Amazon API Gateway 根据账户和区域限制实施令牌存储桶算法，可通过使用量计划为每个客户端配置。此外，Amazon Simple Queue Service（Amazon SQS）和 Amazon Kinesis 可以缓冲请求以稳定请求速率，并在有足够的请求处理能力时实现更高的限制速率。最后，您可以通过 AWS WAF 实施速率限制，以限制产生过高负载的特定 API 使用者。

实施步骤

您可以为 API 配置具有节流限制的 API Gateway，并在超出限制时返回“429 ####”错误。您可以将 AWS WAF 与 AWS AppSync 和 API Gateway 端点结合使用，根据各个 IP 地址来启用速率限制。此外，如果您的系统能够接受异步处理，则可以将消息放入队列或流中，以加快对服务客户端的响应，这样您便可以突增到更高的限制速率。

使用异步处理，当您配置 Amazon SQS 作为 AWS Lambda 的事件源时，您可以 配置最大并发度 ，以避免高速率的事件消耗账户中可用的并发执行配额，因为您的工作负载或账户中的其他服务会需要这些配额。

虽然 API Gateway 提供了令牌存储桶的托管实施，但在无法使用 API Gateway 的情况下，您可以针对服务利用具体语言的令牌存储桶开源实施（参见“资源”中的相关示例）。

- 了解和配置 API Gateway 节流限制 ：对于区域在账户级别，对于阶段在 API 级别，而对于使用量计划级别则为 API 密钥。
- 应用 AWS WAF 速率限制规则 到 API Gateway 和 AWS AppSync 端点，以阻止泛洪攻击并屏蔽恶意 IP。对于 A2A 使用者，也可以在 AWS AppSync API 密钥上配置速率限制规则。

- 对于 AWS AppSync API，请考虑您需要的节流控制是否超过了速率限制，如果超过，请在 AWS AppSync 端点前面配置 API Gateway。
- 将 Amazon SQS 队列设置为 Lambda 队列使用者的触发器时，请将 最大并发度 设置为一个值，使其处理量足以满足服务等级目标，同时使用量又不会超过并发度限制而影响其他 Lambda 函数。当您通过 Lambda 使用队列时，请考虑为相同账户和区域中的其他 Lambda 函数设置预留并发度。
- 将 API Gateway 与 Amazon SQS 或 Kinesis 的原生服务集成结合使用，以缓冲请求。
- 如果您无法使用 API Gateway，请查看具体语言的库，以便为工作负载实施令牌存储桶算法。查看示例部分，然后自行研究以查找合适的库。
- 对您计划设置的限制或者您计划允许增加的限制进行测试，并记录测试后的限制值。
- 不要将限制值提高到超出您在测试中确立的限制值。增加限制时，请先确认预置的资源是否已经等于或大于测试场景中预置的资源，然后再进行增加。

资源

相关最佳实践：

- REL04-BP03 持续工作
- REL05-BP03 控制与限制重试调用

相关文档：

- Amazon API Gateway：限制 API 请求以提高吞吐量
- AWS WAF：基于速率的规则语句
- 引入了使用 Amazon SQS 作为事件源时 AWS Lambda 的最大并发度
- AWS Lambda：最大并发度

相关示例：

- 三条最重要的基于 AWS WAF 速率的规则
- Java Bucket4j
- Python token-bucket
- Node token-bucket
- .NET 系统线程速率限制

相关视频：

- 使用 AWS AppSync 实施 GraphQL API 安全最佳实践

相关工具：

- Amazon API Gateway
- AWS AppSync
- Amazon SQS
- Amazon Kinesis
- AWS WAF

REL05-BP03 控制与限制重试调用

使用指数回退来重试请求，每次重试之间的间隔会逐渐延长。在两次重试之间引入抖动，以随机调整重试间隔。限制最大重试次数。

期望的结果： 分布式软件系统中的常见组件包括服务器、负载均衡器、数据库和 DNS 服务器。在正常运行期间，这些组件对请求的响应可能是临时错误或者受限制错误，也能是无论如何重试都会持续存在的错误。当客户端向服务发出请求时，请求会消耗资源，包括内存、线程、连接、端口或任何其他有限的资源。控制和限制重试策略用于释放资源并最大限度地减少资源消耗，这样就可以避免承受压力的系统组件不堪重负。

当客户端请求超时或收到错误响应时，客户端应决定是否重试。如果进行重试，则会按照最大重试次数值，采用指数回退和抖动方法进行重试。因此，后端服务和进程可以缓解负载并缩短自我修复时间，从而加快恢复速度并成功处理服务请求。

常见反模式：

- 实施重试，但没有添加指数回退、抖动和最大重试次数值。指数回退和抖动有助于避免因无意间按照相同间隔协调进行重试，从而导致的人为流量峰值。
- 实施重试但没有测试重试的效果，或者假设 SDK 中已经内置了重试而不测试重试场景。
- 无法理解依赖项发布的错误代码，导致重试所有错误，包括那些有明确原因的错误，这些错误指出缺乏权限、配置错误或其他预计需要手动干预才能解决的情况。
- 没有解决可观测性实践，包括监控反复出现的服务故障并发出警报，以便了解和解决潜在问题。
- 在内置或第三方重试功能便已足够时，开发自定义重试机制。

- 在应用程序堆栈的多层进行重试，而重试方法导致重试尝试复杂化，进一步加剧了重试风暴中的资源消耗。一定要了解这些错误对您的应用程序以及所依赖的依赖项有何影响，然后仅在一个级别实施重试。
- 重试非幂等的服务调用，导致意外的副作用，例如重复的结果。

建立此最佳实践的好处： 重试有助于客户端在请求失败时获得预期的结果，但也会消耗更多的服务器时间来获取所需的成功响应。当故障比率很低或者是临时性故障时，重试效果很好。当故障是由资源过载导致时，重试会导致情况进一步恶化。通过在客户端重试中添加指数回退和抖动，服务器可以从因资源过载导致的故障中恢复。抖动可避免请求同时出现造成峰值，指数回退可以减少因在正常请求负载中增添重试而导致的负载上升。最后，务必要配置最大重试次数或用时，以避免由于造成积压而导致亚稳态故障。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

控制与限制重试调用。在逐渐延长的间隔以后使用指数回退进行重试。引入抖动以使重试间隔随机化，并限制重试的最大次数。

默认情况下，一些 AWS SDK 实施重试和指数回退。在适用于您的工作负载的情况下，使用这些内置 AWS 实施。在调用幂等性服务时，以及重试可以提高客户端可用性时，在工作负载中实施类似的逻辑。根据您的使用场景确定超时以及何时停止重试。为重试使用场景构建和演练测试场景。

实施步骤

- 对于您的应用程序所依赖的服务，确定应用程序堆栈中最适合实施重试的层。
- 请注意，现有的 SDK 会针对您选择的语言，实施采用了指数回退和抖动方法的成熟重试策略，相比您自己编写重试实施，使用这些实施方法会更好。
- 确认 服务具有幂等性 ，然后再实施重试。实施重试后，请确保在生产环境中进行测试，并定期进行演练。
- 调用 AWS 服务 API 时，使用 AWS SDK 和 AWS CLI 并了解重试配置选项。确定默认值是否适用于您的使用场景，进行测试，并根据需要进行调整。

资源

相关最佳实践：

- REL04-BP04 使所有响应幂等

- REL05-BP02 限制请求
- REL05-BP04 快速失效机制和限制队列
- REL05-BP05 设置客户端超时
- REL11-BP01 监控工作负载的所有组件以检测故障

相关文档：

- AWS 中的错误重试和指数回退
- Amazon Builders' Library：为超时、重试和回退引入抖动
- 指数回退和抖动
- 使用幂等 API 确保重试安全

相关示例：

- Spring 重试
- Resilience4j 重试

相关视频：

- 重试、回退和抖动：AWS re:Invent 2019：介绍 Amazon Builders' Library（DOP328）

相关工具：

- AWS SDK 和工具：重试行为
- AWS Command Line Interface：AWS CLI 重试

REL05-BP04 快速失效机制和限制队列

当服务无法成功响应请求时，可采用快速失效机制。这样可释放与请求关联的资源，并允许该服务在资源不足的情况下进行恢复。快速失效机制一种成熟的软件设计模式，可用于在云端构建高度可靠的工作负载。队列也是一种成熟的企业集成模式，其能够实现平稳的负载，并在能够容忍异步处理的情况下，使得客户端能够释放资源。如果某个服务在正常条件下能够成功响应，但在请求速率过高时失败，请使用队列来缓冲请求。不过，不要允许出现较长的队列积压，否则可能导致处理已被客户端放弃的过时请求。

期望的结果： 当系统遇到资源争用、超时、异常或灰色故障等情况，导致无法实现服务等级目标时，快速失效机制策略可以加快系统恢复速度。如果系统必须承受流量峰值并能够适应异步处理，就可以使用队列来缓冲对后端服务的请求，让客户端可以快速释放请求，从而提高可靠性。在将请求缓冲到队列中时，系统将实施队列管理策略，来避免出现无法克服的积压。

常见反模式：

- 实施消息队列，但不配置死信队列（DLQ）或 DLQ 数量警报来检测系统出现故障的时间。
- 不测量队列中消息的时限（关于延迟的度量），来了解队列使用者何时落后或者出错并导致重试。
- 当业务不需要再存在时，处理积压的消息没有任何价值，但不从队列中清除这些消息。
- 当后进先出（LIFO）队列可以更好地满足客户端需求时，配置先进先出（FIFO）队列，例如，在不需要严格排序并且处理积压内容会延误所有新的和注重时效性的请求时，先进先出队列会导致所有客户端出现违反服务等级协议的情况。
- 向客户端公开内部队列，而不是公开那些管理工作摄入并将请求放入内部队列的 API。
- 将过多的工作请求类型合并到一个队列中，这会导致在一个队列中分配对多种请求类型的资源需求，进而会加剧积压情况。
- 在同一个队列中处理复杂请求和简单请求，但这些请求具有不同的监控、超时和资源分配需求。
- 不验证输入，也不使用断言在软件中实施快速失效机制，这些机制可将异常上报到更高级别的组件来轻松处理错误。
- 不从请求路由中移除出现故障的资源，尤其是在由于崩溃和重启、间歇性依赖项故障、容量减少或网络数据包丢失，导致同时出现成功和失败的灰色故障时。

建立此最佳实践的好处： 采用快速失效机制的系统更容易调试和修复，通常在将版本发布到生产环境之前，在编码和配置阶段就会暴露问题。采用有效排队策略的系统在面对流量高峰和间歇性系统故障的情况时，能够提供更出色的韧性和可靠性。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

快速失效机制策略能够通过编码形式构建到软件解决方案中，也能够在基础设施中配置。除了快速失效机制之外，还可通过队列这种简单而强大的架构技术，来解耦系统组件，实现平稳的负载。Amazon CloudWatch 提供监控故障和发出警报的功能。在确定系统出现故障时，可以调用缓解策略，包括从受损资源进行故障转移。当系统使用 Amazon SQS 和其他队列技术实施队列来实现平稳的负载时，须考虑如何管理队列积压以及消息使用故障。

实施步骤

- 在软件中实施编程式断言或特定指标，并将其用于明确发出关于系统问题的警报。Amazon CloudWatch 有助于根据应用程序日志模式和 SDK 工具创建指标和警报。
- 使用 CloudWatch 指标和警报从受损资源进行故障转移，这些受损资源会增加处理延迟或者在处理请求时反复失败。
- 要使用异步处理，您可以设计 API 来接受请求，并使用 Amazon SQS 将请求附加到内部队列，然后向生成消息的客户端发送成功消息，这样客户端就可以释放资源，并继续处理其他工作，同时后端队列使用者可以处理请求。
- 在每次从队列中删除消息时，通过将现在的时间戳与消息时间戳进行比较来生成 CloudWatch 指标，从而测量和监控队列处理延迟。
- 如果故障导致无法成功处理消息，或者有大量的流量高峰无法按照服务等级协议的要求处理，则将较旧或过多的流量转到溢出队列。这样便可以优先处理新的工作，在有可用容量时再处理较早的工作。这种技术与 LIFO 处理有相似之处，使得可以对所有新工作进行正常的系统处理。
- 使用死信或再驱动队列，将无法处理的消息从积压中移至另一个位置，供以后研究和解决
- 您可以重试，或者在允许的情况下，通过将现在的时间戳与消息时间戳进行比较，丢弃与发出请求的客户端不再相关的消息，以此来删除旧消息。

资源

相关最佳实践：

- REL04-BP02 实施松耦合的依赖关系
- REL05-BP02 限制请求
- REL05-BP03 控制与限制重试调用
- REL06-BP02 定义与计算指标（聚合）
- REL06-BP07 对系统中的请求进行端到端跟踪监控

相关文档：

- 避免无法克服的队列积压
- 快速失效机制
- 如何防止我的 Amazon SQS 队列中积压的消息不断增加？
- Elastic Load Balancing：可用区转移

- Amazon Route 53 应用程序恢复控制器：用于流量失效转移的路由控制

相关示例：

- 企业集成模式：死信通道

相关视频：

- AWS re:Invent 2022 – 运行高可用性多可用区应用程序

相关工具：

- Amazon SQS
- Amazon MQ
- AWS IoT Core
- Amazon CloudWatch

REL05-BP05 设置客户端超时

您应适当设置连接和请求的超时，对其进行系统性验证，不要依赖默认值，因为默认值并不了解具体的工作负载情况。

期望的结果： 客户端超时应考虑当完成请求需要超长时间时，与等待请求相关的客户端、服务器和工作负载成本。由于无法知晓任何超时的确切原因，因此客户端必须运用对服务的了解，预测可能的原因和相应的超时时间

根据配置的值，客户端连接超时。遇到超时后，客户端决定回退并重试，或者打开 断路器。这些模式可避免发出会加剧底层错误状况的请求。

常见反模式：

- 不了解系统超时或默认超时。
- 不了解正常的请求完成时间。
- 不了解完成请求需要超长时间的可能原因，也不了解与等待完成这些请求相关的客户端、服务器或工作负载性能成本。
- 不了解网络受损只要在达到超时后就可能会导致请求失败，也不了解未采用更短的超时时间而招致的客户端和工作负载性能成本。

- 未针对连接和请求来测试超时场景。
- 将超时设置得过高，这会导致等待时间过长并增加资源使用。
- 将超时设置得过低，会导致人为故障。
- 忽略处理远程调用超时错误的模式，例如断路器和重试。
- 不考虑监控服务调用错误率、延迟的服务等级目标和延迟异常值。这些指标能够提供关于过长或不合理超时的洞察信息

建立此最佳实践的好处： 配置了远程调用超时，且系统在设计上可以轻松处理超时，这样在远程调用响应异常缓慢时能够节省资源，且服务客户端可以轻松处理超时错误。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

针对所有服务依赖项调用以及一般情况下的所有跨流程调用，设置连接超时和请求超时。许多框架具有内置超时功能，但仍需谨慎，因为一些超时的默认值为无限值，或者高于您的服务目标可以接受的值。过高的值会降低超时的实用性，因为客户端等待超时发生时，系统会继续消耗资源。过低的值可能会重试请求过多次，因而导致后端流量增加以及延迟变长。在有些情况下，由于要对全部请求进行重试，从而可能导致完全中断。

在确定超时策略时，请考虑以下几点：

- 由于请求的内容、目标服务受损或网络分区故障，处理请求所需的时间可能比正常时间要长。
- 请求如果具有成本异常高的内容，就可能会消耗不必要的服务器和客户端资源。在这种情况下，让这些请求超时而不进行重试可以节省资源。服务还应利用限制和服务器端超时，来保护自身免受成本异常高的内容的侵害。
- 如果由于服务受损而导致请求用时超长，则可以使请求超时并进行重试。请求和重试的服务成本需要考虑在内，但如果原因是局部受损，则重试的成本可能不会太高，而且会减少客户端资源消耗。根据损害的性质，超时还可能会释放服务器资源。
- 如果由于网络未能传输请求或响应，导致请求完成用时很长，则可以使请求超时并进行重试。由于未能传输请求或响应，因此无论超时时间多长，结果都是失败。在这种情况下，超时不会释放服务器资源，但可以释放客户端资源并提高工作负载性能。

利用重试和断路器等成熟的设计模式，来轻松地处理超时并支持快速失效机制方法。AWS SDK 和 AWS CLI 可用于配置连接和请求超时，以及使用指数回退和抖动进行重试。AWS Lambda 函数支持配

置超时，通过 AWS Step Functions，您可以利用与 AWS 服务和 SDK 的预先构建集成，以低代码方式构建断路器。AWS App Mesh Envoy 提供超时和断路器功能。

实施步骤

- 配置远程服务调用的超时，并利用内置语言超时功能或开源超时库。
- 当您的工作负载使用 AWS SDK 进行调用时，请查看文档以了解具体语言的超时配置。
  - Python
  - PHP
  - .NET
  - Ruby
  - Java
  - Go
  - Node.js
  - C++
- 在工作负载中使用 AWS SDK 或 AWS CLI 命令时，可通过设置 AWS 配置默认值 ，为 `connectTimeoutInMillis` 和 `tlsNegotiationTimeoutInMillis`配置默认超时值。
- 应用 命令行选项 `cli-connect-timeout` 和 `cli-read-timeout` ，控制向 AWS 服务发出一次性 AWS CLI 命令。
- 监控远程服务调用的超时，并针对持续性错误设置警报，这样您就能够主动处理错误情况。
- 对调用错误率、延迟的服务等级目标和延迟异常值实施 CloudWatch Metrics 和 CloudWatch 异常检测 ，有助于深入了解如何管理过长或不合理的超时。
- 在 Lambda 函数上配置超时。
- 处理超时的时候，API Gateway 客户端必须实施自己的重试。对于下游集成，API Gateway 支持 50 毫秒到 29 秒的集成超时 ，而且在集成请求超时时不会重试。
- 实施 断路器 模式，以避免在超时时进行远程调用。打开断路器以避免失败调用，当调用响应正常时关闭断路器。
- 对于基于容器的工作负载，请查看 App Mesh Envoy 功能，以利用内置的超时和断路器。
- 使用 AWS Step Functions，以低代码方式为远程服务调用构建断路器，尤其是在调用 AWS 原生 SDK 和所支持的 Step Functions 集成时，以简化工作负载。

资源

相关最佳实践：

- REL05-BP03 控制与限制重试调用
- REL05-BP04 快速失效机制和限制队列
- REL06-BP07 对系统中的请求进行端到端跟踪监控

相关文档：

- AWS SDK：重试和超时
- Amazon Builders' Library：为超时、重试和回退引入抖动
- Amazon API Gateway 配额和重要注意事项
- AWS Command Line Interface：命令行选项
- AWS SDK for Java 2.x：配置 API 超时
- AWS Botocore 使用配置对象和配置引用
- AWS SDK for .NET：重试和超时
- AWS Lambda：配置 Lambda 函数选项

相关示例：

- 将断路器模式与 AWS Step Functions 和 Amazon DynamoDB 配合使用
- Martin Fowler：断路器

相关工具：

- AWS SDK
- AWS Lambda
- Amazon SQS
- AWS Step Functions
- AWS Command Line Interface

REL05-BP06 尽可能使系统为无状态

系统应该不需要状态，或者在不同的客户端请求之间卸载状态，磁盘上和内存中本地存储的数据不存在依赖关系。从而支持任意替换服务器，而且不会对可用性产生影响。

当用户或服务与应用程序进行交互，它们通常会执行一系列交互并构成一次会话。对于用户来说，会话是他们在使用应用程序时持续存在于请求之间的特殊数据。无状态应用程序是无需掌握之前交互而且不会存储会话信息的应用程序。

若采用无状态设计，则您可以使用无服务器计算服务，如 AWS Lambda 或 AWS Fargate。

除了服务器替换，无状态应用程序的另一项优点是，由于任何可用的计算资源（如 EC2 实例和 AWS Lambda 函数）都可以处理任何请求，因此它们可以进行横向扩展。

建立此最佳实践的好处：设计为无状态的系统更能适应横向扩展，从而可以根据流量和需求的波动来增加或删除容量。此类系统还固有故障恢复能力，为应用程序开发提供了灵活性和敏捷性。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

让应用程序无状态无状态应用程序支持横向扩展，并且可以承受单个节点故障。分析和了解在架构内维持状态的应用程序组件。这有助于您评测过渡到无状态设计的潜在影响。无状态架构将用户数据解耦并分流会话数据。这为独立扩展每个组件提供了灵活性，以满足不同的工作负载需求，并优化资源利用率。

实施步骤

- 确定并了解应用程序中的有状态组件。
- 将用户数据与核心应用程序逻辑分离开来进行管理，以此来解耦数据。
  - Amazon Cognito 可以使用身份池、用户池和 Amazon Cognito Sync 等功能，将用户数据与应用程序代码解耦。
  - 您可以使用 AWS Secrets Manager，通过将密钥存储在安全、集中的位置，来解耦用户数据。这意味着应用程序代码不需要存储密钥，这会令其更加安全。
  - 考虑使用 Amazon S3 来存储大型非结构化数据，例如图片和文档。您的应用程序可以在需要时检索这些数据，从而无需将其存储在内存中。
  - 使用 Amazon DynamoDB 存储用户个人资料等信息。您的应用程序可以近乎实时地查询这些数据。
- 将会话数据分流到数据库、缓存或外部文件。

  - 可用于分流会话数据的 AWS 服务示例包括 Amazon ElastiCache、Amazon DynamoDB、Amazon Elastic File System（Amazon EFS）和 Amazon MemoryDB for Redis。
- 在确定需要将哪些状态和用户数据保留在所选存储解决方案中之后，设计无状态架构。

资源

相关最佳实践：

- REL11-BP03 自动修复所有层

相关文档：

- Amazon Builders' Library：避免在分布式系统中回退
- Amazon Builders' Library：避免无法克服的队列积压
- Amazon Builders' Library：缓存挑战和策略
- AWS 上无状态 Web 层的最佳实践

REL05-BP07 实施紧急杠杆

紧急杠杆是可帮助您的工作负载减轻可用性影响的快速流程。

紧急杠杆的工作原理是使用已知且经过测试的机制，禁用、节流或更改组件或依赖项的行为。这可以缓解因需求意外增加导致资源耗尽而造成的工作负载损失，并减少工作负载中非关键组件故障的影响。

期望结果：通过实施紧急杠杆，您可以建立已知良好的流程，以保持工作负载中关键组件的可用性。在激活紧急杠杆期间，工作负载应适当降级，并继续执行其关键业务功能。有关适当降级的更多详细信息，请参阅 REL05-BP01 实施适当降级以将适用的硬依赖关系转换为软依赖关系。

常见的反面模式：

- 非关键依赖关系的故障会影响核心工作负载的可用性。
- 在非关键组件受损时，不测试或验证关键组件的行为。
- 没有为紧急杠杆的激活或停用定义明确的标准。

建立此最佳实践的好处：实施紧急杠杆可以为解决问题的人或程序提供既定流程，以应对意外的需求高峰或非关键依赖关系的故障，从而提高工作负载中关键组件的可用性。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

- 识别工作负载中的关键组件。
- 设计和构造工作负载中的关键组件，使其能够承受非关键组件的故障。
- 进行测试以验证关键组件在非关键组件出现故障期间的行为。
- 定义和监控相关指标或触发器，以启动紧急杠杆程序。
- 定义构成紧急杠杆的（手动或自动）程序。

实施步骤

- 识别工作负载中的关键业务组件。
  - 工作负载中的每个技术组件都应与相关业务职能相对应，并评定为关键组件或非关键组件。有关亚马逊的一些关键和非关键功能的示例，请参阅任何一天都可能成为 Prime Day：Amazon.com 搜索如何使用混沌工程每秒处理超过 8.4 万次请求。
  - 这既是技术决策又是业务决策，而且因组织和工作负载而异。
- 设计和构造工作负载中的关键组件，使其能够承受非关键组件的故障。
  - 在分析依赖关系期间，考虑所有潜在的故障模式，并验证您的紧急杠杆机制是否为下游组件提供了关键功能。
- 进行测试以验证关键组件在紧急杠杆激活期间的行为。
  - 避免双模态行为。有关更多详细信息，请参阅 REL11-BP05 使用静态稳定性来防止双模态行为。
- 定义、监控相关指标并发出警报，以启动紧急杠杆程序。
  - 根据工作负载，找到要监控的正确指标。例如，这些指标可以是延迟，或者是对依赖关系请求失败的次数。
- 定义构成紧急杠杆的手动或自动程序。
  - 这可能包括负载卸除、节流请求或实施适当降级等机制。

资源

相关最佳实践：

- REL05-BP01 实施适当降级以将适用的硬依赖关系转换为软依赖关系
- REL05-BP02 限制请求

- REL11-BP05 使用静态稳定性来防止双模态行为

相关文档：

- 自动执行安全、不需要人工介入的部署
- 任何一天都可能成为 Prime Day：Amazon.com 搜索如何使用混沌工程每秒处理超过 8.4 万次请求

相关视频：

- AWS re:Invent 2020：通过不可变性带来的可靠性、一致性和信心

变更管理

您必须提前为工作负载或其环境的更改做好准备，从而实现工作负载的可靠操作。此类更改包括，从外部施加到工作负载上的更改（如，需求高峰），以及内部更改（如功能部署和安全补丁）。

以下各节将介绍变更管理的最佳实践。

主题

监控工作负载资源

日志和指标是用于了解工作负载运行状况的强大工具。您可以配置工作负载以监控日志和指标，并在超出阈值或发生重大事件时发送通知。监控让您的工作负载可以发现超出低性能阈值和发生故障的情形，从而在响应中自动恢复。

监控对于确保满足可用性要求至关重要。监控需要有效检测故障。最糟糕的故障模式是“沉默”故障，即无法检测到功能已失效，除非间接执行。它会在您采取相关措施前影响到客户。在发生问题时收到提醒是您监控的主要目的之一。警报应该尽量与系统分离开来。如果由于服务中断而无法发出警报，那么服务中断的持续时间会更长。

AWS 在多个级别检测应用程序。我们记录每个请求、所有依赖项和流程内关键操作的延迟、错误率和可用性。也记录成功操作的指标。因此，我们能够在问题发生前发现问题。我们不会仅考虑平均延迟。我们会更审慎地关注延迟异常值，如第 99.9 和 99.99 百分位数。因为在 1000 或 10000 个请求中，即便有一个的速度过慢，体验也会非常糟糕。而且，虽然您的平均值可以接受，但每 100 个请求中有一个会导致极端延迟，那么当您的流量增加时，这最终就会成为问题。

AWS 的监控包含四个不同的阶段：

1. 生成—为工作负载监控全部组件
2. 聚合—定义与计算指标
3. 实时处理与警报—发送警报并使响应自动化
4. 存储与分析

最佳实践

- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP02 定义与计算指标（聚合）
- REL06-BP03 发送通知（实时处理和报警）
- REL06-BP04 自动响应（实时处理和告警）
- REL06-BP05 分析
- REL06-BP06 定期进行审核
- REL06-BP07 对系统中的请求进行端到端跟踪监控

REL06-BP01 为工作负载监控全部组件（生成）

使用 Amazon CloudWatch 或第三方工具监控工作负载组件。使用 AWS Health 控制面板监控 AWS 服务。

应监控您的工作负载的全部组件，包括前端、业务逻辑和存储层。定义关键指标，描述如何将其从日志中提取出来（如有必要），并且设置用于触发对应警报事件的阈值。确保这些指标与您工作负载的关键性能指标（KPI，Key Performance Indicator）相关，并使用指标和日志来确定服务性能下降的早期警告信号。例如，每分钟成功处理的订单数等与业务成果相关的指标，相比 CPU 利用率等技术指标，可以更快地指示工作负载问题。使用 AWS Health 控制面板提供 AWS 资源底层的 AWS 服务的性能和可用性的个性化视图。

云中监控创造新的机会。大多数云提供商都已经开发出可自定义的挂钩，可以提供分析洞察来帮助您监控工作负载的多个层。Amazon CloudWatch 等 AWS 服务应用统计和机器学习算法，集中分析系统与应用程序的指标，确定正常基准，并发现异常，同时最大程度地减少用户干预。异常检测算法考虑了指标的季节性和趋势变动。

AWS 提供了丰富的监控和日志信息以供使用，这些信息可用于定义特定于工作负载的指标、需求变化流程并且采用机器学习技术而无需机器学习专业知识。

此外还会监控您的所有外部端点，确保它们独立于基本实施。这种主动监控可通过合成事务（有时被称为 用户金丝雀，但不要与金丝雀部署相混淆）实现，它们会按照工作负载的客户端所执行的操作，定期执行许多常见任务。请确保这些任务的持续时间较短，并且在测试期间不要使您的工作流过载。Amazon CloudWatch Synthetics 使您能够 创建合成金丝雀 以便监控您的终端节点和 API。您还可以整合 Synthetic Canary 客户端节点和 AWS X-Ray 控制台，精确定位哪些 Synthetic Canary 遇到错误、故障，或对指定时段的速率进行限制的问题。

期望结果：

从工作负载的所有组件收集并使用关键指标，用于确保工作负载的可靠性和提供最佳用户体验。通过检测未能实现业务成果的工作负载，您可以快速发现灾难并从意外事件中恢复。

常见反模式：

- 仅监控连接到工作负载的外部接口。
- 未生成任何特定于工作负载的指标，并且只依靠工作负载所用的 AWS 服务提供给您的指标。
- 仅使用工作负载中的技术指标，不监控与工作负载所带来的非技术 KPI 相关的任何指标。
- 依靠生产流量和简单的运行状况检查来监控并评估工作负载状态。

建立此最佳实践的好处： 在工作负载的所有层级进行监控，方便您更快地预测并解决组成工作负载的组件中的问题。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

1. 启用日志记录功能（如适用）。 应从工作负载的所有组件获取监控数据。启用额外的日志记录，例如 S3 访问日志，并使工作负载记录特定于工作负载的数据。从 Amazon ECS、Amazon EKS、Amazon EC2、Elastic Load Balancing、AWS Auto Scaling 和 Amazon EMR 等服务收集 CPU、网络 I/O 和磁盘 I/O 平均值等指标。请参阅 发布 CloudWatch 指标的 AWS 服务 以了解将指标发布到 CloudWatch 的 AWS 服务列表。
2. 审查所有默认指标并探究任何数据收集欠缺。 每项服务都生成默认指标。通过收集默认指标，您可以更好地了解工作负载组件之间的依赖关系，以及组件的可靠性和性能如何影响工作负载。您还可以 使用 AWS CLI 或 API 创建自己的指标并发布到 CloudWatch。这将
3. 评估所有指标，以确定对于工作负载中的每个 AWS 服务，需要针对哪些指标发布警报。 您还可以选择对工作负载可靠性有重大影响的指标的子集。专注于关键指标和阈值让您可以精调 警报 的数量，并可帮助尽可能减少误报。
4. 定义警报，以及在触发警报之后工作负载的恢复流程。 通过定义警报，您可以快速通知、上报意外事件，按照必要的步骤从意外事件中恢复，并满足规定的恢复时间目标（RTO，Recovery Time Objective）。您可以使用 Amazon CloudWatch 告警 ，根据定义的阈值来调用自动工作流并启动恢复程序。
5. 探索使用合成事务来收集有关工作负载状态的相关数据。 合成监控遵循与客户相同的路线并执行相同的操作，这使得您可以持续验证客户体验，甚至在您的工作负载上没有任何客户流量时也可以。通过使用 合成事务，您可以先于客户发现问题。

资源

相关最佳实践：

- REL11-BP03 自动修复所有层

相关文档：

- 开始使用 AWS Health 控制面板 – 账户的运行状况
- 发布 CloudWatch 指标的 AWS 服务
- Network Load Balancer 的访问日志
- 应用程序负载均衡器的访问日志
- 访问 AWS Lambda 的 Amazon CloudWatch Logs
- Amazon S3 服务器访问日志记录
- 启用 Classic Load Balancer 的访问日志
- 将日志数据导出到 Amazon S3
- 在 Amazon EC2 实例上安装 CloudWatch 代理
- 发布自定义指标
- 使用 Amazon CloudWatch 控制面板
- 使用 Amazon CloudWatch 指标
- 使用金丝雀（Amazon CloudWatch Synthetics）
- 什么是 Amazon CloudWatch Logs？

用户指南：

- 创建跟踪记录
- 监控 Amazon EC2 Linux 实例的内存和磁盘指标
- 结合使用 CloudWatch Logs 与容器实例
- VPC 流日志
- 什么是 Amazon DevOps Guru？
- 什么是 AWS X-Ray？

相关博客：

- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试

相关示例和研讨会：

- AWS Well-Architected 实验室：卓越运营 – 依赖项监控
- Amazon Builders' Library：检测分布式系统的运营可见性
- 可观测性研讨会

REL06-BP02 定义与计算指标（聚合）

存储日志数据并在必要时应用筛选条件以计算指标，例如，特定日志事件的数量，或从日志事件时间戳计算得到的延迟。

Amazon CloudWatch 和 Amazon S3 充当主要聚合层和存储层。某些服务（如 AWS Auto Scaling 和 Elastic Load Balancing）针对整个集群或实例，默认情况下为 CPU 负载或平均请求延迟提供了一些默认指标。对于流式处理服务（如 VPC 流日志和 AWS CloudTrail），事件数据将被转发给 CloudWatch Logs，您需要定义和应用指标筛选条件，才能从事件数据中提取指标。这为您提供了时间序列数据，可被输入到您定义的触发提醒的 CloudWatch 警报。

未建立此最佳实践暴露的风险等级： 高

实施指导

- 定义与计算指标（聚合）。存储日志数据并在必要时应用筛选条件以计算指标，例如，特定日志事件的数量，或从日志事件时间戳计算得到的延迟
  - 指标筛选条件定义在将日志数据发送到 CloudWatch Logs 中时所查找的术语和模式。CloudWatch Logs 使用这些指标筛选条件将日志数据转换为 CloudWatch 数字指标，您可以对这些指标绘制图形或设置警报。
    - 搜索和筛选日志数据
  - 使用受信任第三方来聚合日志。
    - 遵循第三方的说明。大多数第三方产品可以与 CloudWatch 和 Amazon S3 集成。
  - 某些 AWS 服务可以直接向 Amazon S3 发布日志。如果您的主要需求是将日志存储在 Amazon S3 中，则可以让生成日志的服务轻松将其直接发送至 Amazon S3，无需设置额外的基础设施。
    - 将日志直接发送到 Amazon S3

资源

相关文档：

- Amazon CloudWatch Logs Insights 查询示例
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试
- 可观测性研讨会
- 搜索和筛选日志数据
- 将日志直接发送到 Amazon S3
- Amazon Builders' Library：检测分布式系统的运营可见性

REL06-BP03 发送通知（实时处理和报警）

当组织检测到潜在问题时，会向相应的人员和系统发送实时通知和警报，以便快速有效地处理这些问题。

期望的结果： 通过根据服务和应用程序指标配置相关警报，可对运维事件做出快速响应。当超出警报阈值时，相应的人员和系统会收到通知，以便解决潜在的问题。

常见反模式：

- 配置的警报阈值过高，导致无法发送重要通知。
- 配置的警报阈值过低，导致通知过多，而重要警报无法得到处理。
- 使用情况发生变化时不更新警报及其阈值。
- 对于最好通过自动操作来处理的警报，向人员发送通知而不是生成自动操作，从而导致发送的通知过多。

建立此最佳实践的好处： 通过向相应的人员和系统发送实时通知和警报，可以及早发现问题并快速处理运维方面的意外事件。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

工作负载应配备实时处理和报警功能，从而更及时地检测到可能影响应用程序可用性的问题，并充当自动响应的触发器。组织可以通过使用定义的指标创建警报来执行实时处理和报警，以便在发生重大事件或指标超过阈值时收到通知。

Amazon CloudWatch 让您可以基于静态阈值、异常检测和其他标准创建 指标 和复合警报（使用 CloudWatch 警报）。有关您可以使用 CloudWatch 配置的警报类型的详细信息，请参阅 CloudWatch 文档的警报部分。

您可以为团队自定义包含 AWS 资源指标和警报的视图 – 使用 CloudWatch 控制面板。通过 CloudWatch 控制台中的可自定义主页，您可以在单一视图中监控多个区域的资源。

警报可以执行一项或多项操作，例如，向 Amazon SNS 主题发送通知，执行 Amazon EC2 操作或 Amazon EC2 Auto Scaling 操作，或者 创建 OpsItem 或 事件 （在 AWS Systems Manager 中）。

Amazon CloudWatch 使用 Amazon SNS 在警报状态发生变化时发送通知，提供从发布者（生产者）到订阅用户（消费者）的信息传递。有关设置 Amazon SNS 通知的详细信息，请参阅 配置 Amazon SNS。

CloudWatch 会在以下情况下发送 EventBridge 事件 ：每当创建、更新、删除 CloudWatch 警报或者其状态发生变化时。您可以结合使用 EventBridge 与这些事件来创建执行操作的规则，例如，每当警报状态发生变化时通知您，或者使用 Systems Manager 自动化功能自动触发账户中的事件。

何时应使用 EventBridge？何时应使用 Amazon SNS？

EventBridge 和 Amazon SNS 都可用于开发事件驱动型应用程序，您可以根据自己的具体需求进行选择。

如果您想构建一个能对来自您自己的应用程序、SaaS 应用程序和 AWS 服务的事件做出反应的应用程序，建议使用 Amazon EventBridge。EventBridge 是唯一一项直接与第三方 SaaS 合作伙伴集成的基于事件的服务。EventBridge 还可以自动从 200 多个 AWS 服务中摄取事件，而无需开发人员在自己的账户中创建任何资源。

EventBridge 使用已定义的基于 JSON 的事件结构，有助于您创建应用于整个事件主体的规则，以便选择要转发到 目标的事件。EventBridge 目前支持将 20 多种 AWS 服务作为目标，包括 AWS Lambda、Amazon SQS、Amazon SNS、Amazon Kinesis Data Streams和 Amazon Data Firehose。

对于需要高扇出（数千或数百万个端点）的应用程序，建议使用 Amazon SNS。我们常见的一种模式是，客户将 Amazon SNS 用作规则的目标，来筛选所需的事件并扇出到多个端点。

消息是非结构化的，可以是任何格式。Amazon SNS 支持将消息转发到六种目标，包括 Lambda、Amazon SQS、HTTP/S 端点、SMS、移动推送和电子邮件。 Amazon SNS 典型延迟不超过 30 毫秒。许多 AWS 服务配置为发送 Amazon SNS 消息（超过 30 个服务，包括 Amazon EC2、Amazon S3和 Amazon RDS）。

实施步骤

1. 使用 Amazon CloudWatch 警报来创建警报。
    a. 指标警报可监控单个 CloudWatch 指标或依赖于 CloudWatch 指标的表达式。这种警报会根据在若干时间间隔内，指标或表达式的值与阈值的比较结果，启动一项或多项操作。这些操作可能包

括：向 Amazon SNS 主题发送通知，执行 Amazon EC2 操作或 Amazon EC2 Auto Scaling 操作，或者 创建 OpsItem 或 事件 （在 AWS Systems Manager 中）。

b. 复合警报由一个规则表达式组成，该规则表达式考虑了您创建的其他警报的警报条件。只有满足所有规则条件，复合警报才会进入警报状态。复合警报的规则表达式中指定的警报可以包括指标警报和其他复合警报。复合警报可以在状态发生变化时发送 Amazon SNS 通知，并且可以在进入警报状态时创建 Systems Manager OpsItem 或 事件 ，但无法执行 Amazon EC2 或 Auto Scaling 操作。

2. 设置 Amazon SNS 通知。创建 CloudWatch 警报时，可以包括 Amazon SNS 主题，以便在警报状态发生变化时发送通知。
3. 在 EventBridge 中创建规则 以便与指定的 CloudWatch 警报匹配。每条规则都支持多个目标，包括 Lambda 函数。例如，您可以定义一个警报，该警报在可用磁盘空间不足时启动，它会通过 EventBridge 规则触发 Lambda 函数来清理空间。有关 EventBridge 目标的详细信息，请参阅 EventBridge 目标。

资源

相关的 Well-Architected 最佳实践：

- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP02 定义与计算指标（聚合）
- REL12-BP01 使用行动手册调查故障

相关文档：

- Amazon CloudWatch
- CloudWatch Logs Insights
- 使用 Amazon CloudWatch 警报
- 使用 Amazon CloudWatch 控制面板
- 使用 Amazon CloudWatch 指标
- 设置 Amazon SNS 通知
- CloudWatch 异常检测
- CloudWatch Logs 数据保护
- Amazon EventBridge

- Amazon Simple Notification Service

相关视频：

- re:Invent 2022 可观测性视频
- AWS re:Invent 2022 – Amazon 的可观测性最佳实践

相关示例：

- One Observability Workshop
- Amazon EventBridge 通过 Amazon CloudWatch 警报对 AWS Lambda 进行反馈控制

REL06-BP04 自动响应（实时处理和告警）

检测到事件后，利用自动化功能执行操作；例如，更换故障组件。

实施警报的自动实时处理，以便系统可以快速采取纠正措施，并在触发警报时尝试防止故障或服务降级。警报的自动响应可能包括更换故障组件，调整计算容量，将流量重定向到运行状况良好的主机、可用区或其他区域，以及通知操作员。

期望结果：识别实时警报，并设置警报的自动处理，以便调用适当的措施来维护服务级别目标和服务级别协议（SLA）。自动处理的范围可以是单个组件的自我修复活动，也可以是全站点的失效转移。

常见的反面模式：

- 没有明确的关键实时警报的清单或目录。
- 关键警报没有自动响应（例如，当计算资源即将耗尽时自动进行扩展）。
- 警报响应操作相互矛盾。
- 操作员在收到警报通知时没有任何标准操作程序（SOP）可以遵循。
- 不监控配置更改，因为未检测到的配置更改可能会导致工作负载停机。
- 没有撤消意外配置更改的策略。

建立此最佳实践的好处：自动警报处理可以提高系统的韧性。系统会自动采取纠正措施，从而减少手动操作，而手动操作往往是容易出错的人工干预。工作负载的运行符合可用性目标，并减少服务中断。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

为了有效地管理警报并自动进行响应，请根据警报的严重程度和影响对警报进行分类，记录响应程序，并在对任务进行评级之前制定好响应计划。

确定需要特定操作的任务（通常在运行手册中详细说明），并检查所有运行手册和行动手册，以确定哪些任务可以自动执行。如果操作可以定义，它们通常可以自动化。如果操作无法自动化，请在 SOP 中记录手动步骤并对操作员进行培训。不断挑战手动流程，寻找自动化机会，以便制定和维护自动响应警报的计划。

实施步骤

1. 创建警报清单：要获取所有警报的列表，您可以通过结合使用 AWS CLI和 Amazon CloudWatch 命令 `describe-alarms`。根据您设置的警报数量，您可能必须使用分页来检索每次调用的警报子集，或者也可以使用 AWS SDK 通过 API 调用来获取警报。
2. 记录所有警报操作：更新关于所有警报及其操作的运行手册，无论它们是手动还是自动。AWS Systems Manager 提供了预定义的运行手册。有关运行手册的更多信息，请参阅使用运行手册。有关如何查看运行手册内容的详细信息，请参阅查看运行手册内容。
3. 设置和管理警报操作：对于任何需要操作的警报，请使用 CloudWatch SDK 指定自动操作。例如，您可以通过创建和启用警报操作或禁用警报操作，根据 CloudWatch 警报自动更改 Amazon EC2 实例的状态。

   还可以使用 Amazon EventBridge 自动响应系统事件，例如应用程序可用性问题或资源变化。您可以创建规则来指明您对哪些事件感兴趣，以及当事件与规则匹配时要采取的操作。可以自动启动的操作包括调用 AWS Lambda 函数、调用 Amazon EC2 Run Command、将事件中继到 Amazon Kinesis Data Streams，以及查看使用 EventBridge 实现 Amazon EC2 自动化。
4. 标准操作程序（SOP）：根据您的应用程序组件，AWS Resilience Hub 会推荐多个 SOP 模板。您可以使用这些 SOP 来记录在出现警报时操作员应遵循的所有流程。您还可以根据 Resilience Hub 的建议构建 SOP，其中您需要一个具有相关韧性策略的 Resilience Hub 应用程序，以及针对该应用程序的历史韧性评估。针对您的 SOP 的建议由韧性评估生成。

   Resilience Hub 与 Systems Manager 结合，通过提供许多可用作 SOP 基础的 SSM 文档，自动执行 SOP 的步骤。例如，Resilience Hub 可以基于现有 SSM 自动化文档，推荐用于添加磁盘空间的 SOP。
5. 使用 Amazon DevOps Guru 执行自动化操作：可以使用 Amazon DevOps Guru 自动监控应用程序资源的异常行为并提供针对性的建议，以缩短识别问题和进行修复所需的时间。借助 DevOps Guru，您可以近乎实时地监控来自多个来源的运营数据流，包括 Amazon CloudWatch

指标、AWS Config、AWS CloudFormation 和 AWS X-Ray。您还可以使用 DevOps Guru 在 OpsCenter 中自动创建 OpsItems，并将事件发送到 EventBridge 以实现更多自动化操作。

资源

相关最佳实践：

- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP02 定义与计算指标（聚合）
- REL06-BP03 发送通知（实时处理和报警）
- REL08-BP01 对部署等标准活动使用运行手册

相关文档：

- AWS Systems Manager 自动化
- 创建通过 AWS 资源中的事件触发的 EventBridge 规则
- 可观测性研讨会
- Amazon Builders' Library：检测分布式系统的运营可见性
- 什么是 Amazon DevOps Guru？
- 使用自动化文档（行动手册）

相关视频：

- AWS re:Invent 2022 – Amazon 的可观测性最佳实践
- AWS re: Invent 2020：使用 AWS Systems Manager 实现全面自动化
- AWS Resilience Hub 简介
- 为 Amazon DevOps Guru 通知创建自定义工单系统
- 使用 Amazon DevOps Guru 启用多账户洞察聚合

相关示例：

- 可靠性研讨会
- Amazon CloudWatch 和 Systems Manager 研讨会

REL06-BP05 分析

收集日志文件和指标历史，并对其进行分析以获得更广泛的趋势和工作负载见解。

Amazon CloudWatch Logs Insights 支持 简单但强大的查询语言， 您可以用它分析日志数据。Amazon CloudWatch Logs 还支持订阅，允许数据无缝流动到 Amazon S3（您可以在其中使用此类数据）或 Amazon Athena 以便对数据进行查询。它还支持查询多种格式。请参阅 支持的 SerDes 和数据格式（参见 Amazon Athena 用户指南）。针对大型日志文件集的分析，您可以运行 Amazon EMR 集群以执行 PB 级分析。

AWS 合作伙伴和第三方提供了许多用于聚合、处理、存储和分析的工具。这些工具包括 New Relic、Splunk、Loggly、Logstash、CloudHealth 和 Nagios。但是，系统和应用程序日志之外的生成对于每个云提供商，甚至每个服务来说都是独一无二的。

监控过程中常常被忽视的部分是数据管理。您需要确定数据监控的保留要求，然后相应地应用生命周期策略。Amazon S3 支持 S3 存储桶级别的生命周期管理。此生命周期管理可以通过不同的方式应用到存储桶中的不同路径。您可以在生命周期临近结束时，将数据转移到 Amazon S3 Glacier 进行长期存储，然后在保留期结束后让它们过期。S3 智能分层存储类旨在通过将数据自动移动到最具成本效益的访问层，而不会对性能或运营开销产生影响，从而实现优化成本的目的。

未建立此最佳实践暴露的风险等级： 中

实施指导

- 借助 CloudWatch Logs Insights，您可对 Amazon CloudWatch Logs 中的日志数据进行交互搜索和分析。
  - 使用 CloudWatch Logs Insights 分析日志数据
  - Amazon CloudWatch Logs Insights 查询示例
- 使用 Amazon CloudWatch Logs 将日志发送到 Amazon S3，您可以在此处使用这些日志或者使用 Amazon Athena 来查询数据。
  - 如何使用 Athena 分析我的 Amazon S3 服务器访问日志？
    - 为服务器访问日志存储桶创建 S3 生命周期策略。配置生命周期策略以定期删除日志文件。这样做可以减少 Athena 针对每次查询分析的数据量。
      - 如何为 S3 存储桶创建生命周期策略？

资源

相关文档：

- Amazon CloudWatch Logs Insights 查询示例
- 使用 CloudWatch Logs Insights 分析日志数据
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试
- 如何为 S3 存储桶创建生命周期策略？
- 如何使用 Athena 分析我的 Amazon S3 服务器访问日志？
- 可观测性研讨会
- Amazon Builders' Library：检测分布式系统的运营可见性

REL06-BP06 定期进行审核

经常审核工作负载监控的实施情况，并根据重大事件和变更加以更新。

关键业务指标可促进有效监控。确保随着业务优先事项的变化在您的工作负载中对这些指标进行调整。

审计监控有助于确保您了解应用程序何时达到其可用性目标。根本原因分析需要具备在出现故障时发现具体情况的能力。AWS 提供的服务让您能够在意外事件发生期间跟踪服务的状态：

- Amazon CloudWatch Logs： 您可以将日志存储在此服务中并检查日志内容。
- Amazon CloudWatch Logs Insights：是一项完全托管式服务，让您可以在数秒内分析大量日志。它为您提供快速、交互式的查询和可视化。
- AWS Config： 您可以查看在不同的时间点使用了哪些 AWS 基础设施。
- AWS CloudTrail： 您可以查看哪些委托人在什么时候调用了哪些 AWS API。

AWS 每周召开一次会议， 以审查运营性能 并在团队之间分享经验。因为 AWS 有很多团队，我们设置了 The Wheel 以随机挑选一个工作负载进行审查。定期开展运营性能审查和知识共享，有助于您增强帮助运营团队提高绩效的能力。

常见反模式：

- 仅收集默认指标。
- 设置监控策略后不再过问。
- 部署重大更改后不讨论监控问题。

建立此最佳实践的好处： 定期审核监控可主动预测潜在问题，而不是当预测问题真实发生后被动应对通知。

未建立此最佳实践暴露的风险等级： 中

实施指导

- 为工作负载创建多个控制面板。您必须具有顶级控制面板，其中包含关键业务指标，以及已确定与使用情况发生变化时工作负载的预期运行状况最相关的技术指标。您还应该具有可以检查各种应用程序层和依赖项的控制面板。
  - 使用 Amazon CloudWatch 控制面板
- 计划和执行工作负载控制面板常规检查。执行控制面板常规检查。您可能对检查深度具有不同的安排。
  - 检查指标中的趋势。对比指标值与历史值，了解是否有趋势表明需要调查某些情况。这种情况的示例包括：延迟增加、主要业务功能减少以及故障响应增加。
  - 检查指标中的离群值/异常值。平均值或中值会掩盖离群值和异常值。查看时间范围内的最高值和最低值，调查出现这些极值的原因。当您继续消除这些原因时，降低对极值的定义可以使您继续提高工作负载性能的一致性。
  - 查找清晰的行为变化。指标数量或方向的立即更改可能表示应用程序出现更改，或者出现了您需要添加额外指标进行跟踪外部因素。

资源

相关文档：

- Amazon CloudWatch Logs Insights 查询示例
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试
- 可观测性研讨会
- Amazon Builders' Library：检测分布式系统的运营可见性
- 使用 Amazon CloudWatch 控制面板

REL06-BP07 对系统中的请求进行端到端跟踪监控

跟踪各个服务组件的请求处理情况，这样产品团队便能够更轻松地分析和调试问题并提高性能。

期望的结果： 针对所有组件全面跟踪工作负载，实现轻松调试，进而通过简化发现错误根本原因的过程，缩短错误的 平均解决时间 （MTTR）和延迟。采用端到端的跟踪方式，有助于更快地发现受影响的组件，并详细深入地了解造成错误或延迟的根本原因。

常见反模式：

- 只针对部分组件而不是全部组件进行跟踪。例如，如果不跟踪 AWS Lambda，团队可能无法清楚地了解高峰工作负载中冷启动所造成的延迟。
- Synthetics Canary 或真实用户监控（RUM）未配置跟踪功能。没有 Canary 或 RUM，跟踪分析中会忽略客户端交互遥测数据，这样得出的性能概况就不够完整。
- 混合工作负载包括云原生跟踪工具和第三方跟踪工具，但尚未采取措施来选择并完全集成单个跟踪解决方案。根据所选跟踪解决方案，应使用云原生跟踪 SDK 来检测非云原生组件，或者应将第三方工具配置为摄取云原生跟踪遥测数据。

建立此最佳实践的好处： 当开发团队收到问题提醒时，能够查看系统组件交互情况的全貌，包括各个组件在日志记录、性能和故障方面的相关性。由于跟踪有助于直观且轻松地找出根本原因，因此调查根本原因所花费的时间得以减少。在解决问题时，团队如果能详细了解组件的交互情况，就可以更快地做出更好的决策。分析系统跟踪数据有助于改进多种决策，例如何时调用灾难恢复（DR）失效转移，或者在何处实施自我修复策略最合适等，最终势必能够提高客户对服务的满意度。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

团队在运行分布式应用程序时，能够借助跟踪工具来建立关联标识符、收集请求跟踪数据，以及构建互联组件的服务地图。请求跟踪中应该涵盖所有应用程序组件，包括服务客户端、中间件网关和事件总线、计算组件以及存储（包括键/值存储和数据库）。在端到端跟踪配置中纳入 Synthetics Canary 和真实用户监控，来衡量远程客户端交互情况和延迟，这样您就能够根据服务等级协议和目标准确地评估系统性能。

您可以使用 AWS X-Ray 和 Amazon CloudWatch 应用程序监控 检测服务，全面了解应用程序的请求处理情况。X-Ray 可收集应用程序遥测数据，让您能够跨有效负载、函数、跟踪、服务、API 对数据进行可视化和筛选，并且能够通过无代码或低代码方式，为系统组件开启此功能。CloudWatch 应用程序监控包括 ServiceLens，可将您的跟踪数据与指标、日志和警报整合在一起。CloudWatch 应用程序监控还包括用于监控端点和 API 的 Synthetics，以及用于检测 Web 应用程序客户端的真实用户监控。

实施步骤

- 对所有受支持的原生服务使用 AWS X-Ray，例如 Amazon S3、AWS Lambda 和 Amazon API Gateway。这些 AWS 服务可使用基础设施即代码、AWS SDK 或 AWS Management Console，通过切换配置来启用 X-Ray。
- 检测应用程序 适用于 OpenTelemetry 的 AWS Distro 以及 X-Ray 或第三方数据收集代理。

- 请查看 AWS X-Ray 开发人员指南 ，了解实施所用的具体编程语言。这些文档部分详细介绍了如何检测 HTTP 请求、SQL 查询和其他特定于应用程序编程语言的进程。
- 使用 X-Ray 来跟踪 Amazon CloudWatch Synthetics Canary 和 Amazon CloudWatch RUM ，以便分析从最终用户客户端到下游 AWS 基础设施的请求路径。
- 根据资源运行状况和 Canary 遥测数据来配置 CloudWatch 指标和警报，这样团队就能够快速收到问题提醒，然后使用 ServiceLens 深入探究跟踪数据和服务地图。
- 如果您使用第三方工具作为主要跟踪解决方案，则将 X-Ray 与下列第三方跟踪工具进行集成：Datadog、New Relic或 Dynatrace 。

资源

相关最佳实践：

- REL06-BP01 为工作负载监控全部组件（生成）
- REL11-BP01 监控工作负载的所有组件以检测故障

相关文档：

- 什么是 AWS X-Ray？
- Amazon CloudWatch：应用程序监控
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试
- Amazon Builders' Library：检测分布式系统的运营可见性
- 将 AWS X-Ray 与其他 AWS 服务集成
- 适用于 OpenTelemetry 的 AWS Distro 以及 AWS X-Ray
- Amazon CloudWatch：使用 Synthetics 监控
- Amazon CloudWatch：使用 CloudWatch RUM
- 设置 Amazon CloudWatch Synthetics Canary 和 Amazon CloudWatch 警报
- 可用性及其他内容：了解和提高 AWS 上分布式系统的韧性

相关示例：

- 可观测性研讨会

相关视频：

- AWS re:Invent 2022 – 如何跨多个账户监控应用程序
- 如何监控 AWS 应用程序

相关工具：

- AWS X-Ray
- Amazon CloudWatch
- Amazon Route 53

设计工作负载以适应需求的变化

可扩展 工作负载 具有自动添加或移除资源的弹性，因此确保在任何时间点都能准确满足当前的需求。

最佳实践

- REL07-BP01 在获取或扩展资源时利用自动化
- REL07-BP02 在检测到对工作负载的破坏时获取资源
- REL07-BP03 当检测到某个工作负载需要更多资源时，就会获取资源
- REL07-BP04 对工作负载进行负载测试

REL07-BP01 在获取或扩展资源时利用自动化

在替换被损坏的资源或扩展您的工作负载时，通过采用托管 AWS 服务（如 Amazon S3 和 AWS Auto Scaling）对流程进行自动处理。您还可以使用第三方工具和 AWS 开发工具包自动扩展。

托管 AWS 服务包括 Amazon S3、Amazon CloudFront、AWS Auto Scaling、AWS Lambda、Amazon DynamoDB、AWS Fargate 和 Amazon Route 53。

AWS Auto Scaling 让您可以检测与替换被破坏的实例。它还可以帮助您为资源制定扩展计划，包括 Amazon EC2 实例和 Spot 队列、 Amazon ECS 任务、 Amazon DynamoDB 表和索引，以及 Amazon Aurora 副本。

在扩展 EC2 实例时，请确保您使用多个可用区（最好至少三个）并增加或减少容量以保持这些可用区之间的平衡。ECS 任务或 Kubernetes 容器组（pod）（使用 Amazon Elastic Kubernetes Service 时）也应分布在多个可用区中。

如果使用 AWS Lambda，实例会自动扩展。每次收到关于您的函数的事件通知时，AWS Lambda 会快速找到其计算队列中的可用容量，然后运行您的代码至分配的并发值。您需要确保在特定的 Lambda 上，以及在您的Service Quotas中配置必要的并发值。

Amazon S3 会自动扩展以处理较高的请求速率。例如，您的应用程序可以在存储桶中为每个前缀每秒至少发送 3500 个 PUT/COPY/POST/DELETE 或 5500 个 GET/HEAD 请求。存储桶中的前缀数量没有限制。您可以通过并行化读取提高您的读取或写入性能。例如，如果在 Amazon S3 存储桶中创建 10 个前缀以便对读取进行并行化，您可以将读取性能扩展至每秒 55000 个读取请求。

配置和使用 Amazon CloudFront 或受信任的内容分发网络（CDN，Content Delivery Network）。CDN 可以缩短最终用户的响应时间，并从缓存中为请求提供内容，从而减少扩展工作负载的请求。

常见反模式：

- 实施 Auto Scaling 组进行自动修复，但无法实施弹性。
- 使用 Auto Scaling 响应流量激增。
- 部署高状态应用程序，消除了部署弹性选项。

建立此最佳实践的好处： 自动化可以避免部署和淘汰资源时的潜在手动错误。自动化可以避免由于缓慢响应部署或淘汰需求而导致的服务超支和拒绝服务风险。

未建立此最佳实践暴露的风险等级： 高

实施指导

- 配置和使用 AWS Auto Scaling。它会监控您的应用程序，并自动调整容量来维持稳定、可预测的性能，并且成本最低。使用 AWS Auto Scaling，您可以跨多个服务为多个资源轻松设置应用程序扩展。
  - 什么是 AWS Auto Scaling？
    - 在您的 Amazon EC2 实例和竞价型实例集、Amazon ECS 任务、Amazon DynamoDB 表和索引、Amazon Aurora 副本以及 AWS Marketplace 设备上配置自动扩展（如果适用）。
      - 使用 DynamoDB Auto Scaling 自动管理吞吐能力
        - 使用服务 API 操作来指定警报、扩展策略、预热时间和冷却时间。
- 使用 Elastic Load Balancing。负载均衡器可以按路径或网络连接分配负载。
  - 什么是 Elastic Load Balancing？
    - Application Load Balancers 可以按路径分配负载。

    - 什么是 Application Load Balancer？
      - 配置 Application Load Balancer，根据域名下的路径将流量分配给不同的工作负载。
      - Application Load Balancers 可以与 AWS Auto Scaling 集成来分配负载，以便管理需求。
        - 将负载均衡器与自动扩缩组配合使用
    - 网络负载均衡器可以按连接分配负载。
      - 什么是网络负载均衡器？
        - 配置网络负载均衡器，以便使用 TCP 将流量分配给不同的工作负载，或者为工作负载指定一组恒定的 IP 地址。
        - 网络负载均衡器可以与 AWS Auto Scaling 集成来分配负载，以便管理需求。
- 使用高度可用的 DNS 提供商。使用 DNS 名称，用户可以输入名称而不是 IP 地址来访问您的工作负载，并将该信息分发到指定的范围内，通常面向全局范围内工作负载的所有用户。
  - 使用 Amazon Route 53 或可信 DNS 提供商。
    - 什么是 Amazon Route 53？
  - 使用 Route 53 管理 CloudFront 分配和负载均衡器。
    - 确定要管理的域和子域。
    - 使用 ALIAS 或 CNAME 记录来创建适当的记录集。
      - 使用记录
- 使用 AWS 全球网络可优化用户与应用程序之间的路径。AWS Global Accelerator 持续监控应用程序端点的运行状况，可在 30 秒内将流量重定向到运行状况良好的端点。
  - AWS Global Accelerator 是一项可帮助本地或全球用户提高应用程序可用性和性能的服务。它提供的静态 IP 地址可用作从单个或多个 AWS 区域 区域（例如 Application Load Balancers、网络负载均衡器或 Amazon EC2 实例）访问应用程序端点的固定入口点。
    - 什么是 AWS Global Accelerator？
- 配置和使用 Amazon CloudFront 或受信任的内容分发网络（CDN，Content Delivery Network）。内容分发网络可以缩短最终用户的响应时间，还可以对可能导致工作负载进行不必要扩展的内容请求做出响应。
  - 什么是 Amazon CloudFront？
    - 针对您的工作负载配置 Amazon CloudFront 分配，或者使用第三方 CDN。
      - 您可以通过在端点安全组或访问策略中使用 CloudFront 的 IP 范围，将对工作负载的访问限制为只能从 CloudFront 访问。

资源

相关文档：

- APN 合作伙伴：可以帮您制定自动计算解决方案的合作伙伴
- AWS Auto Scaling：扩展计划的工作原理
- AWS Marketplace：可以与 Auto Scaling 一起使用的产品
- 使用 DynamoDB Auto Scaling 自动管理吞吐能力
- 将负载均衡器与自动扩缩组配合使用
- 什么是 AWS Global Accelerator？
- 什么是 Amazon EC2 Auto Scaling？
- 什么是 AWS Auto Scaling？
- 什么是 Amazon CloudFront？
- 什么是 Amazon Route 53？
- 什么是 Elastic Load Balancing？
- 什么是网络负载均衡器？
- 什么是 Application Load Balancer？
- 使用记录

REL07-BP02 在检测到对工作负载的破坏时获取资源

如果可用性受到影响，在必要时被动扩展资源，从而还原工作负载的可用性。

首先，您必须配置运行状况检查和关于此类检查的标准，表示在什么时候可用性会因缺少资源而受到影响。然后，通知适当的人员手动扩展资源，或启动自动化以对其进行自动扩展。

可以为您的工作负载手动调整规模（例如，通过 AWS Management Console 或 AWS CLI 更改 Auto Scaling 组中 EC2 实例的数量，或者修改 DynamoDB 表的吞吐量来实现）。但是，应尽可能使用自动化（请参阅获取或扩展资源时使用自动化）。

期望结果：在检测到故障或客户体验下降时，启动扩展活动（自动或手动）以恢复可用性。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

对工作负载中的所有组件实施可观测性和监控，以监控客户体验并检测故障。定义扩展所需资源的手动或自动程序。有关更多信息，请参阅 REL11-BP01 监控工作负载的所有组件以检测故障。

实施步骤

- 定义扩展所需资源的手动或自动程序。
  - 扩展程序取决于工作负载中不同组件的设计方式。
  - 扩展程序还因所使用的底层技术而异。
    - 使用 AWS Auto Scaling 的组件可以使用扩展计划来配置一组用于扩展资源的指令。如果使用 AWS CloudFormation 或为 AWS 资源添加标签，您可以根据应用程序为不同的资源集设置扩展计划。Auto Scaling 提供了针对每个资源自定义的扩展策略建议。创建扩展计划后，Auto Scaling 结合了动态扩展和预测式扩缩方法来支持扩展策略。有关更多详细信息，请参阅扩展计划的工作原理。
    - Amazon EC2 Auto Scaling 会验证您是否拥有适量的 Amazon EC2 实例，可处理您的应用程序负载。您可创建 EC2 实例的集合，称为 Auto Scaling 组。您可以指定每个 Auto Scaling 组中的最小和最大实例数，Amazon EC2 Auto Scaling 会确保您的组永远不会低于或超过这些限制。有关更多详细信息，请参阅什么是 Amazon EC2 Auto Scaling？
    - Amazon DynamoDB 自动扩缩使用 Application Auto Scaling 服务，代表您动态调整预置的吞吐能力，以响应实际的流量模式。这将使表或全局二级索引提高预置读取和写入容量，从而在不节流的情况下应对流量突增。有关更多详细信息，请参阅使用 DynamoDB 自动扩缩自动管理吞吐容量。

资源

相关最佳实践：

- REL07-BP01 在获取或扩展资源时利用自动化
- REL11-BP01 监控工作负载的所有组件以检测故障

相关文档：

- AWS Auto Scaling：扩展计划的工作原理
- 使用 DynamoDB 自动扩缩自动管理吞吐容量
- 什么是 Amazon EC2 Auto Scaling？

REL07-BP03 当检测到某个工作负载需要更多资源时，就会获取资源

主动扩展资源以满足需求并避免影响可用性。

很多 AWS 服务会自动扩展以满足需求。如果使用 Amazon EC2 实例或 Amazon ECS 集群，您可以根据与您的工作负载的需求对应的使用指标，配置它们会在何时自动扩展。针对 Amazon EC2，平均 CPU 利用率、负载均衡器请求数量，或网络带宽可被用于扩展（或缩减）EC2 实例。而对于 Amazon ECS，可使用平均 CPU 利用率、负载均衡器请求数量和内存利用率横向扩展（或横向缩减）ECS 任务。在 AWS 上使用 Target Auto Scaling，Autoscaler 将扮演“家用恒温器”的角色，增加或减少资源以保持您所指定的目标值（例如，70% CPU 利用率）。

AWS Auto Scaling 还可以执行 Predictive Auto Scaling，该操作利用机器学习来分析每个资源的历史工作负载，并且定期预测未来两天的负载。

利特尔法则可帮助计算您需要多少计算实例（EC2 实例、并发 Lambda 函数，等等）。

$L = \lambda W$

L = 实例数量（或系统中的平均并发值）

λ = 收到请求的平均速率（请求数量/秒）

W = 每个请求在系统中所花的平均时间（秒）

例如，假设每秒请求数为 100，若每个请求所需的处理时间为 0.5 秒，您将需要 50 个实例才能满足需求。

未建立此最佳实践暴露的风险等级： 中

实施指导

- 当检测到某个工作负载需要更多资源时，就会获取资源。主动扩展资源以满足需求并避免影响可用性。
  - 计算处理给定请求速率需要多少计算资源（计算并发）。
    - 讲述与利特尔法则有关的故事
  - 当您具有历史使用模式时，请为 Amazon EC2 Auto Scaling 设置计划扩展。
    - Amazon EC2 Auto Scaling 的计划扩缩
  - 使用 AWS 预测式扩缩。
    - 由机器学习提供支持的 EC2 预测式扩缩

资源

相关文档：

- AWS Auto Scaling：扩展计划的工作原理
- AWS Marketplace：可以与 Auto Scaling 一起使用的产品
- 使用 DynamoDB Auto Scaling 自动管理吞吐能力
- 由机器学习提供支持的 EC2 预测式扩缩
- Amazon EC2 Auto Scaling 的计划扩缩
- 讲述与利特尔法则有关的故事
- 什么是 Amazon EC2 Auto Scaling？

REL07-BP04 对工作负载进行负载测试

采用负载测试方法来衡量扩展活动能否满足工作负载要求。

持续开展负载测试，这一点很重要。负载测试用于发现工作负载的断点并测试工作负载的性能。利用 AWS，您可以轻松设置能够模拟生产工作负载规模的临时测试环境。在云中，您可以根据需要创建一套生产规模等级的测试环境，完成测试，然后停用资源。由于测试环境只需在运行时付费，您模拟真实环境的成本仅为本地测试成本的一小部分。

生产中的负载测试还应该被视为实际试用活动的一部分，因为在客户使用量降低的那几个小时内，在场的所有员工都忙于解读结果与处理任何出现的问题，生产系统承受着很大的压力。

常见反模式：

- 对与您的生产采用不同配置的部署执行负载测试。
- 仅对单个工作负载分段（而非整个工作负载）执行负载测试。
- 使用请求子集，而不是具有代表性的实际请求集执行负载测试。
- 对超出预期负载的较小安全系数执行负载测试。

建立此最佳实践的好处： 您知道架构中哪些组件会在负载下失败，而且能够确定要监控哪些可指示您即将达到该负载的指标，从而及时解决问题，防止故障影响。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

- 执行负载测试，确定工作负载的哪些方面表明您必须添加或移除容量。负载测试应具有您在生产中接收的流量类似的代表性流量。增加负载，同时监视所有已检测指标，以便确定哪种指标指示何时必须添加或移除资源。
  - AWS 上的分布式负载测试：模拟数千个连接的用户
    - 确定请求组合。您可能拥有不同的请求组合，因此应当在确定流量组合时查看不同的时间范围。
    - 实施负载驱动程序。您可以使用自定义代码、开源或商用软件来实施负载驱动程序。
    - 最初使用小容量进行负载测试。通过将负载降低到较小容量（可能小到一个实例或容器），可能会有立竿见影的效果。
    - 针对更大的容量进行负载测试。分布式负载的效果会有所不同，因此您必须对尽量接近生产环境的目标进行测试。

资源

相关文档：

- AWS 上的分布式负载测试：模拟数千个连接的用户
- 负载测试应用程序

相关视频：

- AWS 2023 ANZ 峰会：通过 AWS 分布式负载测试充满信心地提升速度

实施变更

部署新功能和确保工作负载及运行环境运行已知，而且经过适当修补的软件需要对变更进行控制。如果此类更改不受控制，您将难以预测这些更改的影响，或难以处理由它们引发的问题。

最佳实践

- REL08-BP01 对部署等标准活动使用运行手册
- REL08-BP02 将功能测试作为部署的一部分进行集成
- REL08-BP03 将韧性测试作为部署的一部分进行集成
- REL08-BP04 使用不可变基础设施进行部署
- REL08-BP05 使用自动化功能部署更改

REL08-BP01 对部署等标准活动使用运行手册

运行手册是用来实现特定结果的预定义程序。使用运行手册执行标准活动，无论这些活动是手动还是自动执行。其中的示例包括部署工作负载，修补工作负载，或修改 DNS。

例如，实施流程以 确保部署期间安全回滚.确保您可以为客户进行部署回滚而不会出现中断，这是保证服务可靠的关键。

针对运行手册程序，从一个有效的手动流程开始，用代码进行实施，并在适当的情况下触发其自动运行。

即使是高度自动化的复杂工作负载，运行手册同样适用于 运行实际试用 或用于满足严格的报告和审计要求。

请注意，行动手册可用于对特定事件做出响应，运行手册则用来达成特定的结果。通常，运行手册适用于例行活动，而行动手册则被用于对非例行事件做出响应。

常见反模式：

- 对生产中的配置执行计划外更改。
- 跳过计划中的步骤以加快部署速度，导致部署失败。
- 在未测试反向更改的情况下做出更改。

建立此最佳实践的好处： 有效更改计划有助于您成功执行更改，因为您知道所有受影响的系统。在测试环境中验证更改能够增强您的信心。

未建立此最佳实践暴露的风险等级： 高

实施指导

- 通过在运行手册中记录程序，实现对为人熟知的事件的一致且及时的响应。
  - AWS Well-Architected Framework：概念：运行手册
- 使用基础设施即代码的原则定义您的基础设施。通过使用 AWS CloudFormation（或受信任的第三方）来定义您的基础设施，您可以使用版本控制软件对更改实施版本控制并进行跟踪。
  - 使用 AWS CloudFormation（或受信任的第三方提供商）定义您的基础设施。
    - 什么是 AWS CloudFormation？
  - 使用良好的软件设计原则创建单个解耦模板。
    - 确定实施的权限、模板和责任方。

    - 使用 AWS Identity and Access Management 控制访问权限
  - 使用源代码控制（例如 AWS CodeCommit 或受信任的第三方工具）进行版本控制。
    - 什么是 AWS CodeCommit？

资源

相关文档：

- AWS 合作伙伴：可以帮助您创建自动化部署解决方案的合作伙伴
- AWS Marketplace：可用于自动实施部署的产品
- AWS Well-Architected Framework：概念：运行手册
- 什么是 AWS CloudFormation？
- 什么是 AWS CodeCommit？

相关示例：

- 使用行动手册和运行手册自动完成操作

REL08-BP02 将功能测试作为部署的一部分进行集成

功能测试作为自动化部署的一部分运行。若未满足成功条件，则相关管道会中止或回滚。这些测试在预生产环境中运行，该环境会在管道中的生产开始前被暂存。在理想情况下，此操作是部署管道的一部分。

期望结果：您使用自动化功能来执行功能测试，相关的测试数据减少了测试用时和费用，并提高了测试结果的准确性。您可以将功能测试集成到部署流程中，这有助于您实现发布管道的自动化，从而快速可靠地更新应用程序和基础设施。

常见反面模式：

- 您在部署管道之外手动执行测试。
- 您跳过自动化流程中的测试步骤，采用手动应急工作流。
- 您急于求成，而不遵循既定的测试计划和流程。

建立此最佳实践的好处：功能测试可验证系统能否按照指定要求运行。使用该最佳实践，可持续验证用户界面、API、数据库和源代码等组件是否按预期正常运行。当您检查系统的这些组件时，功能测试会

验证每个功能是否按预期运行，这既可以确保达成用户的期望，也保护了软件的完整性。将功能测试融入到常规部署中，并使用自动化功能来部署所有变更，从而降低引入人为错误的可能性。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

将功能测试作为部署的一部分进行集成。功能测试作为自动化部署的一部分运行。若未满足成功条件，则相关管道会中止或回滚。AWS CodePipeline 提供了连续的交付管道用于自动化测试，这使得测试人员可以实现整个测试和部署流程的自动化。与 AWS CodeBuild 和 AWS CodeDeploy 等 AWS 服务集成后，可自动执行软件开发生命周期的构建、测试和部署阶段。

实施步骤

- 配置管道：使用 AWS CodePipeline 控制台或 AWS Command Line Interface（CLI）设置源代码，以及构建、测试和部署阶段。
  - 定义源代码：使用 AWS CodePipeline，您可以自动从 GitHub、AWS CodeCommit 或 Bitbucket 等版本控制系统检索源代码，这样可以确保测试中使用的是最新代码。
  - 自动构建和测试：AWS CodeBuild 可以自动构建和测试您的代码并生成测试报告。该服务支持 JUnit、NUnit 和 TestNG 等流行的测试框架。
  - 部署代码：构建和测试代码后，AWS CodeDeploy 可以将其部署到您的测试环境，包括 Amazon EC2 实例、AWS Lambda 函数或本地服务器。
  - 监控管道：AWS CodePipeline 可以跟踪管道的进度和每个阶段的状态。您还可以使用质量检查，根据测试执行状态来阻止管道。此外，您可以接收有关任何管道阶段故障或管道完成的通知。

资源

相关文档：

- Use AWS CodePipeline with AWS CodeBuild to test code and run builds
- Logging and monitoring in AWS CodeBuild
- Indicators for functional testing

REL08-BP03 将韧性测试作为部署的一部分进行集成

集成韧性测试功能，通过在系统中特意引入故障来衡量系统在遇到破坏性情景时的功能。韧性测试不同于通常集成在部署周期中的单元和功能测试，因为它们侧重于识别系统中的意外故障。虽然在预生产环

境中集成韧性测试是安全的，但您要设定一个目标，将这些测试作为演练日活动的一部分在生产环境中实施。

期望结果：韧性测试有助于建立信心，确信系统能够承受生产环境中的性能降级。通过实验来找出可能导致故障的薄弱环节，这可以帮助您改进系统，从而自动有效地减少故障和性能降级的情况。

常见反面模式：

- 部署流程中缺乏可观测性和监控能力
- 依靠人工来解决系统故障
- 糟糕的质量分析机制
- 只看到系统中的已知问题，缺少通过实验来发现未知问题的手段
- 识别故障，但没有解决
- 没有关于调查发现和运行手册的文档

建立此最佳实践的好处：在部署中集成韧性测试有助于识别系统中的未知问题，以防这些问题被忽视，从而导致生产停机。识别系统中的这些未知问题可以协助您记录调查发现，将测试集成到 CI/CD 流程中，并制定运行手册，从而通过高效、可重复的机制简化缓解措施。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

在系统部署中，可以集成的最常见的韧性测试形式是灾难恢复和混沌工程。

- 对于任何重大部署，都要包括对灾难恢复计划和标准操作程序（SOP，Standard Operating Procedure）的更新。
- 将可靠性测试集成到您的自动部署管道中。AWS Resilience Hub 等服务可以集成到您的 CI/CD 管道中，用于构建持续韧性评测功能，从而在每次部署中自动进行评测。
- 在 AWS Resilience Hub 中定义您的应用程序。韧性评测会生成代码片段，帮助您以 AWS Systems Manager 文档的形式为应用程序创建恢复程序，并提供推荐的 Amazon CloudWatch 监控和警报列表。
- 更新灾难恢复计划和标准操作程序后，完成灾难恢复测试以验证它们是否有效。灾难恢复测试可帮助您确定，在事件发生后是否可以恢复系统并恢复正常运行。您可以模拟各种灾难恢复策略，确定计划是否足以满足您的正常运行时间要求。常见的灾难恢复策略包括备份和恢复、指示灯、冷备用、温备用、热备用和双活模式，这些方法成本和复杂性各不相同。在进行灾难恢复测试之前，建议您定

义恢复时间目标（RTO）和恢复点目标（RPO），这样可以简化模拟策略的选择。AWS 提供 AWS Elastic Disaster Recovery 等灾难恢复工具，用于帮助您开始进行规划和测试。

- 混沌工程实验会在系统中引入中断，例如网络中断和服务故障。通过模拟受控的故障，您可以发现系统的漏洞，同时控制注入故障的影响。就像其他策略一样，请在非生产环境中，使用 AWS Fault Injection Service 等服务运行受控故障模拟，以便在生产环境中进行部署之前，树立起对系统韧性的信心。

资源

相关文档：

- Experiment with failure using resilience testing to build recovery preparedness
- Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline
- Disaster recovery (DR) architecture on AWS, part 1: Strategies for recovery in the cloud
- Verify the resilience of your workloads using Chaos Engineering
- Principles of Chaos Engineering
- Chaos Engineering Workshop

相关视频：

- AWS re:Invent 2020: Testing Resilience using Chaos Engineering
- Improve Application Resilience with AWS Fault Injection Service
- Prepare & Protect Your Applications From Disruption With AWS Resilience Hub

REL08-BP04 使用不可变基础设施进行部署

不可变基础设施模式要求在生产工作负载上不会出现就地更新、安全补丁或配置更改。需要更改时，会在新的基础设施上构建架构，并将其部署到生产环境中。

遵循不可变基础设施部署策略，以提高工作负载部署的可靠性、一致性和可重复性。

期望结果：使用不可变基础设施，不允许为了运行工作负载中的基础设施资源而就地修改。相反，当需要更改时，会将一组包含所有必要更改的新基础设施资源与现有资源并行部署。会自动验证此部署，如果成功，流量将逐渐转移到新的资源集。

此部署策略适用于软件更新、安全补丁、基础设施更改、配置更新和应用程序更新等。

常见的反面模式：

- 对正在运行的基础设施资源实施就地更改。

建立此最佳实践的好处：

- 提高跨环境的一致性：由于不同环境的基础设施资源没有差异，因此提高了一致性并简化了测试。
- 减少配置偏差：通过使用已知且受版本控制的配置替换基础设施资源，可以将基础设施设置为已知、经过测试和可信的状态，从而避免配置偏差。
- 可靠的原子部署：部署要么成功完成，要么没有任何变化，从而提高部署过程的一致性和可靠性。
- 简化部署：由于无需支持升级，部署得到简化。升级即意味着新的部署。
- 采用快速回滚和恢复流程的更安全部署：由于之前运行的版本未发生更改，因此部署变得更安全。您可以在检测到错误时进行回滚。
- 增强安全状况：通过不允许更改基础设施，可以禁用远程访问机制（例如 SSH）。这样做可以减少攻击向量，改善您的组织的安全状况。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

自动化

在定义不可变基础设施部署策略时，建议尽可能使用自动化，来提高可重复性并最大限度地减少出现人为错误的可能性。有关更多详细信息，请参阅 REL08-BP05 使用自动化功能部署更改和自动执行安全、不需要人工介入的部署。

使用基础设施即代码（IaC），基础设施预置、编排和部署步骤以编程、描述性和声明性方式定义，并存储在源代码管理系统中。利用基础设施即代码可以更轻松地自动化基础设施部署，并有助于实现基础设施的不可变性。

部署模式

当需要更改工作负载时，不可变基础设施部署策略要求部署一组新的基础设施资源，包括所有必要的更改。这组新资源必须遵循可最大限度地减少对用户影响的推出模式，这一点非常重要。此部署有两种主要策略：

金丝雀部署：将您的少量客户引导到新版本的做法，它通常在单个服务实例（金丝雀）上运行。然后，您可以深入检查生成的任何行为更改或错误。如果遇到了严重问题，您可以将 Canary 中的流量删除，

并将用户发回到以前的版本。如果部署成功，您可以继续以期望的速度进行部署，同时监控更改以便发现错误，直到所有部署完成。AWS CodeDeploy 的部署配置可以配置为允许金丝雀部署。

蓝绿部署：与金丝雀部署类似，只是会并行部署一整套应用程序。您可以在两个堆栈（蓝和绿）之间轮流部署。同样，您可以将流量发送到新版本中，如果发现部署中存在问题，可以对其进行故障恢复，然后送回旧版本中。通常来说，所有流量会被一次性切换，但您也可以通过 Amazon Route 53 的加权 DNS 路由功能向每个版本发送部分流量，以加快采用新版本的速度。AWS CodeDeploy 和 AWS Elastic Beanstalk 的部署配置可以配置为允许蓝绿部署。

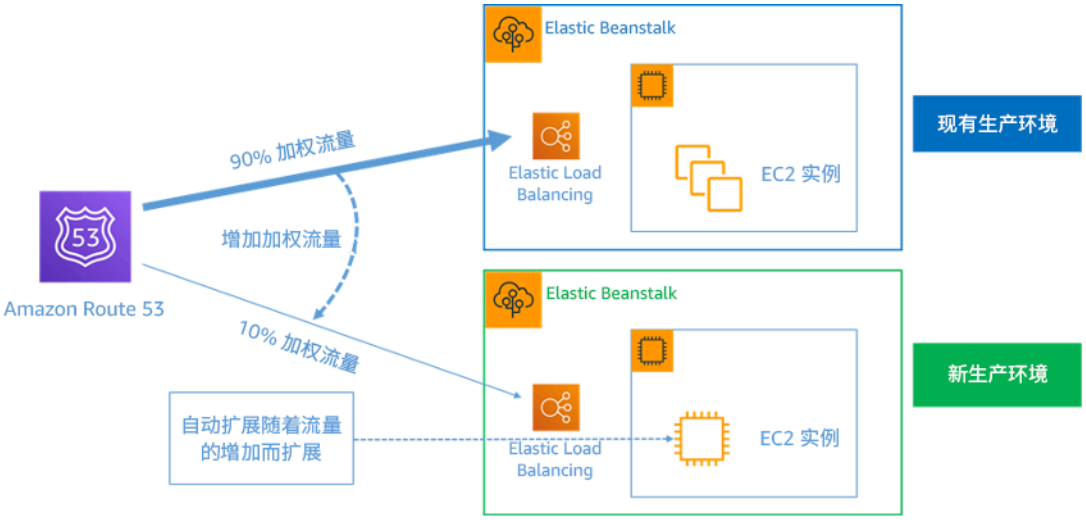


图 8：使用 AWS Elastic Beanstalk 和 Amazon Route 53 进行蓝绿部署

偏差检测

偏差是指导致基础设施资源的状态或配置不同于预期的任何更改。任何类型非受管配置更改都与不可变基础设施的概念背道而驰，因此应加以检测和修复，以便成功实施不可变基础设施。

实施步骤

- 禁止就地修改正在运行的基础设施资源。
  - 您可以使用 AWS Identity and Access Management（IAM）来指定谁或什么可以访问 AWS 中的服务和资源，集中管理精细权限，并分析访问权限以细化 AWS 中的权限。
- 自动部署基础设施资源，以提高可重复性并最大限度地减少出现人为错误的可能性。
  - 正如《AWS 上的 DevOps 简介》白皮书中所述，自动化是 AWS 服务的基石，在所有服务、功能和产品中都受到内部支持。
  - 预先制作您的亚马逊机器映像（AMI）可以加快启动它们的时间。EC2 Image Builder 是一项完全托管式 AWS 服务，可协助您自动创建、维护、验证、共享和部署自定义、安全且最新的 Linux 或 Windows 自定义 AMI。

- 一些支持自动化的服务包括：
  - AWS Elastic Beanstalk 这项服务可用于在 Apache、NGINX、Passenger 和 IIS 等熟悉的服务器上快速部署和扩展使用 Java、.NET、PHP、Node.js、Python、Ruby、Go 和 Docker 开发的 Web 应用程序。
  - AWS Proton 让平台团队能够连接和协调开发团队进行基础设施预置、代码部署、监控和更新所需的所有不同工具。AWS Proton 可实现自动化基础设施即代码预置，以及无服务器和基于容器的应用程序的部署。
- 利用基础设施即代码可以轻松实现基础设施部署的自动化，并有助于实现基础设施的不可变性。AWS 提供的服务可用于以编程、描述和声明的方式创建、部署和维护基础设施。
  - AWS CloudFormation 让开发人员能够以有序和可预测的方式创建 AWS 资源。资源使用 JSON 或 YAML 格式写入文本文件。模板需要特定的语法和结构，具体取决于创建和管理的资源类型。您可以使用任何代码编辑器（例如 AWS Cloud9）以 JSON 或 YAML 格式编写资源，将其签入版本控制系统，然后 CloudFormation 会以安全、可重复的方式构建指定的服务。
  - AWS Serverless Application Model（AWS SAM）是一个开源框架，可用于在 AWS 上构建无服务器应用程序。AWS SAM 与其他 AWS 服务集成，是 AWS CloudFormation 的扩展。
  - AWS Cloud Development Kit (AWS CDK) 是一个开源软件开发框架，用于使用熟悉的编程语言对您的云应用程序资源进行建模和预置。您可以使用 AWS CDK 通过 TypeScript、Python、Java 和 .NET 对应用程序基础设施进行建模。AWS CDK 在后台使用 AWS CloudFormation，以安全、可重复的方式提供资源。
  - AWS Cloud Control API 引入了一组通用的创建、读取、更新、删除和列出（CRUDL）API，以协助开发人员以简单一致的方式管理其云基础设施。Cloud Control API 通用 API 允许开发人员统一管理 AWS 和第三方服务的生命周期。
- 实施能够最大限度减少对用户的影响的部署模式。
  - 金丝雀部署：
    - 设置 API Gateway 金丝雀版本部署
    - 使用 AWS App Mesh 为 Amazon ECS 的金丝雀部署创建管道
  - 蓝绿部署：《AWS 上的蓝绿部署》白皮书描述了实施蓝绿部署策略的示例技术。
- 检测配置或状态偏差。有关更多详细信息，请参阅检测堆栈和资源的非受管配置更改。

资源

相关最佳实践：

- REL08-BP05 使用自动化功能部署更改

相关文档：

- 自动执行安全、不需要人工介入的部署
- 利用 AWS CloudFormation 在 Nubank 创建不可变的基础设施
- 基础设施即代码
- 实现警报以自动检测 AWS CloudFormation 堆栈中的偏差

相关视频：

- AWS re:Invent 2020：通过不可变性带来的可靠性、一致性和信心

REL08-BP05 使用自动化功能部署更改

自动部署与修补以消除负面影响。

对许多组织来说，对生产系统进行变更是风险最大的工作之一。除了软件解决的业务问题外，我们认为部署也是亟待解决的首要问题。如今，这意味着根据实际情况在操作中使用自动化，包括测试和部署更改、添加或删除容量以及迁移数据。

期望结果：您通过全面的预生产测试、自动回滚和分阶段生产部署，将自动化部署安全性融入发布流程中。这种自动化尽可能地减少了部署失败对生产造成的潜在影响，开发人员不再需要主动关注部署到生产的情况。

常见反面模式：

- 您手动执行更改。
- 您跳过自动化流程中的步骤，采用手动应急工作流。
- 您急于求成，而不遵循既定的计划和流程。
- 您在不预留烘焙时间的情况下，快速执行了后续部署。

建立此最佳实践的好处：当您使用自动化功能来部署所有更改时，可以消除引入人为错误的可能性，并实现了在更改生产环境之前进行测试的能力。在生产推送之前执行此流程，以便验证您的计划是否能完成。此外，自动回滚到发布流程可以识别生产问题，并将您的工作负载恢复到以前的正常工作运行状态。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

实现部署管道的自动化。借助部署管道，您可以调用自动化测试和异常检测，并且能够在生产部署前的某个步骤停止管道，或自动回滚更改。其中必不可少的是采用持续集成和持续交付/部署（CI/CD）文化，这样一来，提交或代码更改会经过各种自动化阶段，完成构建和测试，并最终部署至生产环境中。

虽然传统观点认为，您应该让人来处理循环中最困难的操作程序，但出于相同的原因，我们建议您将最困难的程序自动化。

实施步骤

您可以按照以下步骤实现自动化部署，从而消除手动操作：

- 设置代码存储库来安全地存储您的代码：使用 AWS CodeCommit，创建基于 Git 的安全存储库。
- 配置持续集成服务来编译源代码、运行测试和创建部署构件：要为此目的设置构建项目，请参阅 Getting started with AWS CodeBuild using the console。
- 设置部署服务，以自动执行应用程序部署并处理复杂的应用程序更新，而无需依赖容易出错的人工部署过程：AWS CodeDeploy 可自动将软件部署到各种计算服务，例如 Amazon EC2、AWS Fargate、AWS Lambda 和本地服务器。要配置这些步骤，请参阅 Getting started with CodeDeploy。
- 设置持续交付服务，实现发布管道的自动化，从而带来更快、更可靠的应用程序和基础设施更新：请考虑使用 AWS CodePipeline 来帮助您实现发布管道的自动化。有关更多详细信息，请参阅 CodePipeline tutorials。

资源

相关最佳实践：

- OPS05-BP04 使用构建和部署管理系统
- OPS05-BP10 完全自动化集成和部署
- OPS06-BP02 测试部署
- OPS06-BP04 自动测试和回滚

相关文档：

- Continuous Delivery of Nested AWS CloudFormation Stacks Using AWS CodePipeline

- Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline
- AWS 合作伙伴：可以帮助您创建自动化部署解决方案的合作伙伴
- AWS Marketplace：可用于自动化部署的产品
- Automate chat messages with webhooks
- Amazon Builders' Library：确保部署期间安全回滚
- Amazon Builders' Library：采用持续交付，加速交付进度
- What Is AWS CodePipeline?
- What Is CodeDeploy?
- AWS Systems Manager Patch Manager
- What is Amazon SES?
- What is Amazon Simple Notification Service?

相关视频：

- AWS Summit 2019: CI/CD on AWS

故障管理

> 如果故障在所难免，那么一切操作会在一段时间后全部失败：从路由器到硬盘，从操作系统到内存单元 TCP 数据包损坏，从暂时性错误到永久性故障。这是注定的，不管您使用的是最高质量的硬件或最低成本的组件 - Werner Vogels，首席技术官 - Amazon.com

低级别的硬件组件故障是本地数据中心每天都要处理的问题。不过，在云中，您可以避免大多数的此类故障。例如，Amazon EBS 卷被置于特定的可用区内，它们会被自动复制，以避免单个组件出现故障。所有 EBS 卷都被设计具有 99.999% 的可用性。Amazon S3 对象会被存储在至少三个可用区内，在指定的一年时间内为其提供 99.999999999% 的持久性。无论选择哪家云提供商，您都有可能遭遇故障，因而对您的工作负载造成影响。因此，如果要使您的工作负载具有可靠性，您必须采取措施以实施弹性。

应用此处讨论的最佳实践的先决条件为，您必须确保负责设计实施并运行您的工作负载的人员认识到要做到这一点所需的业务目标和可靠性目标。这些人员必须了解这些可靠性要求，并且接受过相关的培训。

以下各节将介绍管理故障以预防对您的工作负载产生影响的最佳实践：

主题

- 备份数据
- 使用故障隔离，以保护您的工作负载
- 将工作负载设计为能够承受组件故障的影响
- 测试可靠性
- 灾难恢复 (DR) 计划

备份数据

备份数据、应用程序和配置，以满足恢复时间目标 (RTO) 和恢复点目标 (RPO) 的要求。

最佳实践

- REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据
- REL09-BP02 保护并加密备份

- REL09-BP03 自动执行数据备份
- REL09-BP04 定期执行数据恢复以验证备份完整性和流程

REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据

了解并使用工作负载所用的数据服务和资源的备份功能。大多数服务提供了备份工作负载数据的功能。

期望结果：数据来源已确定，并根据重要性进行了分类。然后，根据 RPO 为数据恢复建立了策略。此策略涉及到备份这些数据来源，或者能够从其他来源复制数据。在出现数据丢失的情况下，所实施的策略可以在定义的 RPO 和 RTO 内实现数据的恢复或复制。

云成熟度阶段：基础

常见反模式：

- 不了解工作负载的所有数据来源及其重要性。
- 没有对关键数据来源进行备份。
- 仅对部分数据来源进行备份，但没有考虑重要性标准。
- 没有定义 RPO，或者备份频率无法满足 RPO。
- 没有评估备份是否必需或者是否可以从其他来源复制数据。

建立此最佳实践的好处：确定需要备份的位置并实施某种机制来创建备份，或者具备从外部来源复制数据的能力，这样可以提高在停机期间还原和恢复数据的能力。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

所有 AWS 数据存储均提供备份功能。Amazon RDS 和 Amazon DynamoDB 等服务还额外地支持可实现时间点故障恢复（PITR）的自动备份，这使您可以将备份恢复到距当前时间不超过五分钟的任意时间点。许多 AWS 服务提供了将备份复制到其他 AWS 区域 的功能。AWS Backup 工具向您提供了在不同 AWS 服务中集中实现自动化数据保护的能力。AWS Elastic Disaster Recovery 使您可以从本地、跨可用区或跨区域复制完整的服务器工作负载并保持连续数据保护，恢复点目标（RPO）以秒为单位。

Amazon S3 可用作自行管理数据来源和 AWS 托管数据来源的备份目标。Amazon EBS、Amazon RDS 和 Amazon DynamoDB 等 AWS 服务具有可用于创建备份的内置功能。此外，也可使用第三方备份软件。

可以使用 AWS Storage Gateway 或 AWS DataSync 将本地数据备份到 AWS Cloud。Amazon S3 存储桶可用于在 AWS 中存储此数据。Amazon S3 提供多个存储层（例如 Amazon S3 Glacier 或 S3 Glacier Deep Archive），可用于降低数据存储的成本。

您可以从其他来源复制数据，以此来满足数据恢复需求。例如，Amazon ElastiCache 副本节点或 Amazon RDS 只读副本可用于在主来源丢失时复制数据。如果像这样的来源可用于满足恢复点目标（RPO）和恢复时间目标（RTO）要求，您可能不需要备份。在另一个例子中，如果使用 Amazon EMR，只要可以将数据从 Amazon S3 复制到 Amazon EMR 中，则可能不需要备份 HDFS 数据存储。

在选择备份策略时，请考虑恢复数据所用的时间。恢复数据所需的时间取决于备份的类型（在采用备份策略时）或数据复制机制的复杂性。此时间应该符合工作负载的 RTO。

实施步骤

1. 确定工作负载的所有数据来源。数据可以存储在多种资源中，例如数据库、卷、文件系统、日志记录系统和对象存储。请参阅资源部分，查找有关存储数据的不同 AWS服务的相关文档，以及这些服务提供的备份功能。
2. 根据重要性对数据来源进行分类。对于工作负载，不同数据集具有不同的重要程度，因此对弹性具有不同的要求。例如，一些数据可能会非常重要，要求接近于零的 RPO，而另一些数据则不那么重要，可以承受较高的 RPO 和某种程度的数据丢失。与此类似，不同数据集也可能会有不同的 RTO 要求。
3. 使用 AWS 或第三方服务来创建数据的备份。AWS Backup 是一项托管服务，支持在 AWS 上创建各种数据源的备份。AWS Elastic Disaster Recovery 处理到 AWS 区域 的自动亚秒级数据复制。大多数 AWS 服务还具有原生的创建备份功能。AWS Marketplace 有许多解决方案同样提供了这些功能。请参阅下面所列的资源，了解如何从不同 AWS 服务创建数据备份的信息。
4. 对于没有备份的数据，请建立数据复制机制。您可能会出于各种原因，不对可从其他来源复制的数据进行备份。您可能会遇到一种情况，在需要时从来源复制数据的成本相比创建备份更低，因为可能会有与存储备份相关的成本。另一个例子是从备份进行还原的时间比从来源复制数据用时更长，使得备份不符合 RTO 要求。在此类情况下请做出权衡，并建立明确定义的流程，确定在需要进行恢复时如何从这些来源复制数据。例如，如果您从 Amazon S3 将数据加载到数据仓库（如 Amazon Redshift）或 MapReduce 集群（如 Amazon EMR），以便对此类数据进行分析，这就是可从其他来源复制数据的例子。只要此类分析的结果被存储在某位置，或者可重现，您不会因为数据仓库或 MapReduce 集群故障而遭遇数据丢失的情况。其他可从数据源复制数据的例子包括，缓存（如 Amazon ElastiCache）或 RDS 只读副本。
5. 建立备份数据的频率。创建数据来源的备份是一个定期执行的流程，其频率取决于 RPO。

实施计划的工作量级别：适中

资源

相关最佳实践：

REL13-BP01 定义停机和数据丢失的恢复目标

REL13-BP02 使用定义的恢复策略来实现恢复目标

相关文档：

- 什么是 AWS Backup？
- 什么是 AWS DataSync？
- 什么是卷网关？
- AWS 合作伙伴：可以帮助进行备份的合作伙伴
- AWS Marketplace：可以用于备份的产品
- Amazon EBS 快照
- 备份 Amazon EFS
- 备份 Amazon FSx for Windows File Server
- ElastiCache for Redis 备份和还原
- 在 Neptune 中创建数据库集群快照
- 创建数据库快照
- 创建按计划触发的 EventBridge 规则
- 使用 Amazon S3 进行跨区域复制
- EFS 到 EFS AWS Backup
- 将日志数据导出到 Amazon S3
- 对象生命周期管理
- DynamoDB 的按需备份和还原
- DynamoDB 的时间点恢复
- 使用 Amazon OpenSearch Service 索引快照
- 什么是 AWS Elastic Disaster Recovery？

相关视频：

- AWS re:Invent 2021 – 使用 AWS 进行备份、灾难恢复和勒索软件防护

- AWS Backup 演示：跨账户和跨区域备份
- AWS re:Invent 2019：深入了解 AWS Backup，主讲：Rackspace(STG341)

相关示例：

- Well-Architected 实验室 - 为 Amazon S3 实施双向跨区域复制（CRR）
- Well-Architected 实验室 - 测试数据的备份与还原
- Well-Architected 实验室 - 面向分析工作负载的备份和还原（具备失效自动恢复功能）
- Well-Architected 实验室 - 灾难恢复 - 备份与还原

REL09-BP02 保护并加密备份

使用身份验证和授权来控制并检测对备份的访问。使用加密功能，防止并检测备份的数据完整性是否受到损坏。

常见反模式：

- 对备份和还原自动化的访问权限与对数据的访问权限相同。
- 未加密您的备份。

建立此最佳实践的好处：保护备份安全可防止篡改数据，而加密数据可防止数据意外暴露时对其访问。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 AWS Identity and Access Management（IAM）等身份验证和授权服务，控制并检测对备份的访问。使用加密功能，防止并检测备份的数据完整性是否受到损坏。

Amazon S3 支持多种对您的静态数据进行加密的方式。借助服务器端加密功能，Amazon S3 以未加密数据的形式接受您的对象，然后在存储此类数据时进行加密。若采用客户端加密，您的工作负载应用程序需要负责在将其发送到 Amazon S3 之前加密数据。这两种方式都让您可以使用 AWS Key Management Service（AWS KMS）创建并存储数据密钥，或者您也可以提供自己的密钥并自行对其负责。使用 AWS KMS，您可以通过 IAM 设置策略，决定谁可以以及谁不可以访问您的数据密钥与解密数据。

针对 Amazon RDS，如果您已选择对数据库进行加密，那么您的备份也会被加密。DynamoDB 备份则始终被加密。使用 AWS Elastic Disaster Recovery 时，加密所有传输中数据和静态数据。借助 Elastic

Disaster Recovery，可以使用默认 Amazon EBS 加密 Volume Encryption Key 或自定义客户管理的密钥来加密静态数据。

实施步骤

1. 对每个数据存储使用加密。如果源数据已加密，则备份也将被加密。
   - 在 Amazon RDS 中使用加密。当您创建 RDS 实例时，可以使用 AWS Key Management Service 配置静态加密。
   - 在 Amazon EBS 卷上使用加密。您可以配置默认加密或在创建卷时指定唯一密钥。
   - 使用所需的 Amazon DynamoDB 加密。DynamoDB 会加密所有静态数据。您可以使用 AWS 拥有的 AWS KMS 密钥或者 AWS 托管 KMS 密钥，指定存储在您账户中的密钥。
   - 加密 Amazon EFS 中存储的数据。在创建文件系统时配置加密。
   - 在源和目标区域中配置加密。您可以使用 KMS 中存储的密钥配置 Amazon S3 中的静态加密，但这些密钥是特定于区域的。您在配置复制时可以指定目标密钥。
   - 选择是为 Elastic Disaster Recovery 使用默认还是自定义 Amazon EBS 加密。使用此选项会加密暂存区域子网磁盘和复制磁盘上的已复制静态数据。
2. 实施用于访问您的备份的最低权限。请遵循最佳实践，根据安全最佳实践来限制对备份、快照和副本的访问。

资源

相关文档：

- AWS Marketplace：可以用于备份的产品
- Amazon EBS 加密
- Amazon S3：利用加密保护数据
- CRR 附加配置：复制通过存储在 AWS KMS 中的加密密钥、使用服务器端加密（SSE）创建的对象
- DynamoDB 静态加密
- 加密 Amazon RDS 资源
- 在 Amazon EFS 中加密数据和元数据
- AWS 中的备份的加密
- 管理加密的表
- 安全性支柱 – AWS Well-Architected Framework
- 什么是 AWS Elastic Disaster Recovery？

相关示例：

- Well-Architected 实验室 - 为 Amazon S3 实施双向跨区域复制（CRR）

REL09-BP03 自动执行数据备份

将备份配置为根据遵循恢复点目标（RPO）的定期计划自动备份，或者在数据集发生更改时自动备份。具有低数据丢失需求的关键数据资产需要频繁地自动备份，而可以接受某些丢失的较不重要数据的备份频率可以更低。

期望结果：按照确定的节奏创建数据来源备份的自动流程。

常见反模式：

- 手动执行备份。
- 使用具有备份功能的资源，但不包括自动化中的备份。

建立此最佳实践的好处：自动化备份可以确保按照 RPO 执行备份，并在未备份时发出警报。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

AWS Backup 可用于创建各种 AWS 数据来源的自动数据备份。Amazon RDS 实例可以按照五分钟的频率进行几乎连续的备份，Amazon S3 对象可以按照十五分钟的频率进行几乎连续的备份，提供可恢复到备份历史记录中的特定时间点的时间点故障恢复（PITR）功能。对于其他 AWS 数据来源（如 Amazon EBS 卷、Amazon DynamoDB 表或 Amazon FSx 文件系统），AWS Backup 最快可以按每小时的频率运行自动备份。这些服务还提供了原生备份功能。以下 AWS 服务提供了具备时间点故障恢复的自动备份功能：Amazon DynamoDB、Amazon RDS 和 Amazon Keyspaces（Apache Cassandra 兼容）- 这些备份可以恢复到备份历史记录中的特定时间点。大部分其他 AWS 数据存储服务提供了计划定期备份的功能，频率最快为每小时一次。

Amazon RDS 和 Amazon DynamoDB 提供支持时间点恢复的持续备份。一旦启用，Amazon S3 版本控制即是自动的。Amazon Data Lifecycle Manager 可用于自动拍摄、复制和删除 Amazon EBS 快照。它还可以自动创建、复制、弃用和取消注册 Amazon EBS 支持的亚马逊云机器镜像（AMI）及其底层 Amazon EBS 快照。

AWS Elastic Disaster Recovery 提供从源环境（本地或 AWS）到目标恢复区域的持续、块级复制。服务会自动创建和管理时间点 Amazon EBS 快照。

针对您的备份自动化和历史的集中式视图，AWS Backup 提供完全托管的基于策略的备份解决方案。它会使用 AWS Storage Gateway 将云端和本地的多项 AWS 服务的数据备份集中在一起并自动处理。

除了版本控制，Amazon S3 还具有复制功能。整个 S3 存储桶都可自动复制到相同或不同 AWS 区域中的其他存储桶。

实施步骤

1. 确定当前在手动备份的数据来源。有关更多详细信息，请参阅 REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据.
2. 确定工作负载的 RPO。有关更多详细信息，请参阅 REL13-BP01 定义停机和数据丢失的恢复目标.
3. 使用自动化备份解决方案或托管服务。AWS Backup 是一项完全托管式服务，可让您在云端和本地对不同 AWS 服务中的数据保护实现集中化和自动化。使用 AWS Backup 中的备份计划，创建规则来定义要备份的资源，以及创建这些备份的频率。此频率应遵循在第 2 步中确定的 RPO。有关如何使用 AWS Backup 创建自动备份的动手实践指导，请参阅测试数据的备份和恢复。用于存储数据的大多数 AWS 服务提供了原生备份功能。例如，可以利用 RDS 来实现支持时间点故障恢复（PITR）的自动备份。
4. 对于自动备份解决方案或托管服务不支持的数据来源（如本地数据来源或消息队列），请考虑使用受信任的第三方解决方案来创建自动备份。或者，您可以使用 AWS CLI 或开发工具包创建自动化过程来完成此操作。您可以使用 AWS Lambda 函数或 AWS Step Functions 来定义创建数据备份中涉及的逻辑，并使用 Amazon EventBridge 按照基于 RPO 确定的频率来执行它。

实施计划的工作量级别：低

资源

相关文档：

- AWS 合作伙伴：可以帮助进行备份的合作伙伴
- AWS Marketplace：可以用于备份的产品
- 创建按计划触发的 EventBridge 规则
- 什么是 AWS Backup？
- 什么是 AWS Step Functions？
- 什么是 AWS Elastic Disaster Recovery？

相关视频：

- AWS re:Invent 2019：深入了解 AWS Backup，主讲：Rackspace (STG341)

相关示例：

- Well-Architected 实验室 - 测试数据的备份与还原

REL09-BP04 定期执行数据恢复以验证备份完整性和流程

通过执行恢复测试，验证您的备份流程实施是否满足恢复时间目标（RTO）和恢复点目标（RPO）要求。

期望结果：使用明确定义的机制定期从备份恢复数据，确认可以按照为工作负载确定的恢复时间目标（RTO）来恢复数据。验证从备份进行还原可以得到包含原始数据的资源，而不会造成数据损坏或无法访问数据，并且数据丢失在恢复点目标（RPO）之内。

常见反模式：

- 还原备份，但未查询或检索任何数据以确认还原操作可用。
- 假定备份存在。
- 假定系统的备份完全正常运行，并且可从中恢复数据。
- 假定从备份还原或恢复数据的时间满足工作负载的 RTO。
- 假定备份中包含的数据符合工作负载的 RPO
- 需要时进行还原，没有使用运行手册或者没有按照确定的自动程序执行。

建立此最佳实践的好处：测试备份的恢复过程可以确认在需要时能够将数据还原，不必担心数据可能丢失或损坏，可以按照工作负载要求的 RTO 还原和恢复，并且任何数据丢失都符合工作负载的 RPO。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

测试备份和还原功能可树立信心，确信能够在出现中断时执行这些操作。定期将备份还原到新的位置，并运行测试以验证数据的完整性。应该执行一些常见的测试，以核实所有数据是否均可用、未损坏、可访问且任何数据丢失都符合工作负载的 RPO。此类测试还可以帮助确定恢复机制是否足够快以满足工作负载的 RTO 要求。

使用 AWS，您可以构建一个测试环境，还原您的备份以评估 RTO 和 RPO 功能，并且对数据的内容和完整性执行测试。

此外，Amazon RDS 和 Amazon DynamoDB 还允许时间点恢复（PITR）。您可以使用持续备份将您的数据集还原到其在指定日期与时间所处的状态。

数据是否可用、没有损坏、是否可以访问并且任意数据丢失都符合工作负载的 RPO。此类测试还可以帮助确定恢复机制是否足够快以满足工作负载的 RTO 要求。

AWS Elastic Disaster Recovery 提供 Amazon EBS 卷的持续时间点恢复快照。复制源服务器时，根据配置的策略记录一段时间内的时间点状态。Elastic Disaster Recovery 可以启动实例用于测试和演练，而不重定向流量，从而帮助您验证这些快照的完整性。

实施步骤

1. 确定当前备份的数据来源以及存储这些备份的位置。有关实施指导，请参阅 REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据.
2. 为每个数据来源建立数据验证标准。不同类型的数据具有不同的属性，这可能需要不同的验证机制。在您确信将此数据用于生产之前，请考虑可以如何验证此数据。一些验证数据的常见方法包括使用数据和备份属性，例如数据类型、格式、校验和、大小，或者将这些属性与自定义的验证逻辑结合使用。例如，可以将所恢复资源的校验和值，与创建备份时数据来源的校验和值进行比较。
3. 建立 RTO 和 RPO，以根据数据重要性来还原数据。有关实施指导，请参阅 REL13-BP01 定义停机和数据丢失的恢复目标.
4. 评估恢复能力。检查备份和还原策略，了解它是否可以满足您的 RTO 和 RPO，并根据需要调整策略。使用 AWS 韧性监测中心，您可以对工作负载运行评估。该评估根据弹性策略评估您的应用程序配置，并报告是否能够满足 RTO 和 RPO 目标。
5. 使用当前为生产环境中数据还原所确立的流程执行测试还原。这些流程依赖于对原始数据来源进行备份的方法，备份本身的格式和存储位置，或者数据是否从其他来源复制。例如，如果您使用 AWS Backup 等托管服务，则此过程可能就是简单地将备份还原到新的资源。如果您使用 AWS Elastic Disaster Recovery，则可以启动恢复演练。
6. 根据您之前为数据验证确立的标准，从还原后的资源验证数据恢复。还原和恢复的数据是否包含备份时的最新记录或项目？ 此数据是否在工作负载的 RPO 之内？
7. 测量还原和恢复所需的时间，并与确立的 RTO 进行比较。此流程是否符合工作负载的 RTO？ 例如，比较还原过程开始时的时间戳以及恢复验证完成时的时间戳，以计算此过程的用时。所有 AWS API 调用均有时间戳，此信息在 AWS CloudTrail 中提供。虽然此信息可以提供还原过程何时开始的详细信息，但验证完成时的结束时间戳应该由您的验证逻辑来记录。如果使用自动化过程，则 Amazon DynamoDB 等服务可用于存储此信息。此外，许多 AWS 服务提供了事件历史记录，其中可提供发生特定操作时的时间戳信息。在 AWS Backup 中，备份和还原操作称为作业，这些作业在其元数据中包含时间戳信息，可用于测量还原和恢复所需的时间。

8. 如果数据验证失败，或者如果还原和恢复所需的时间超过了为工作负载设定的 RTO，则通知利益攸关方。在实施自动化以完成此操作时（例如在本实验中），可以使用 Amazon Simple Notification Service（Amazon SNS）等服务将推送通知（例如电子邮件或短信）发送给利益攸关方。这些消息还可以发布到消息传递应用程序，例如 Amazon Chime、Slack 或 Microsoft Teams，或用于使用 AWS Systems Manager OpsCenter 来创建 OpsItems 等任务。
9. 自动执行此流程以便定期运行。例如，AWS Lambda 等服务或 AWS Step Functions 中的状态机可用于自动完成还原和恢复流程，Amazon EventBridge 可用于定期触发此自动工作流，如以下架构图中所示。了解如何使用 AWS Backup 自动完成数据恢复验证。此外，这个 Well-Architected 实验室提供动手实践体验，可用于练习针对此处的多个步骤实现自动化的方法。

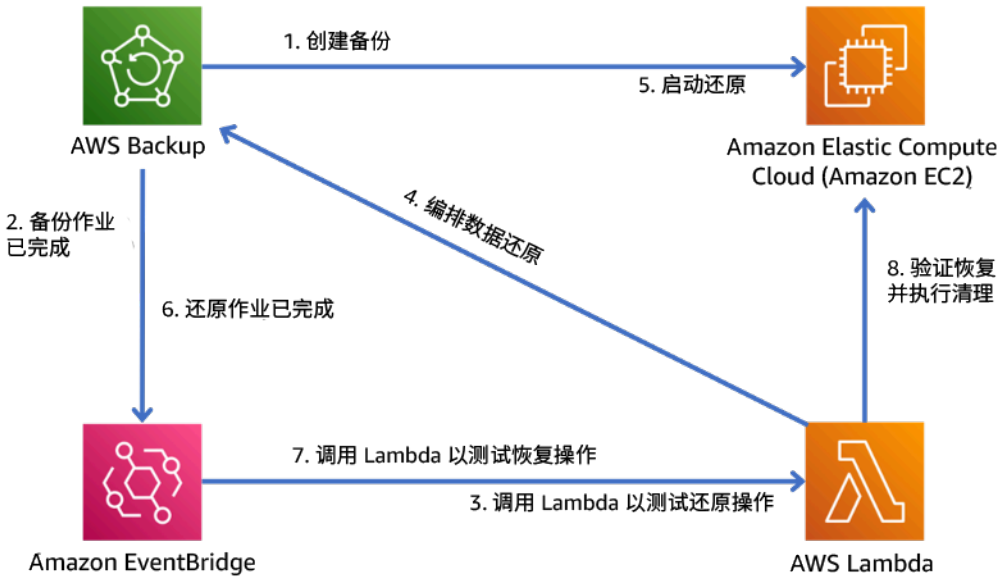


图 9.自动化的备份和还原流程

实施计划的工作量级别：中到高，具体取决于验证条件的复杂性。

资源

相关文档：

- 使用 AWS Backup 自动完成数据恢复验证
- AWS 合作伙伴：可以帮助进行备份的合作伙伴
- AWS Marketplace：可以用于备份的产品
- 创建按计划触发的 EventBridge 规则
- DynamoDB 的按需备份和还原

- 什么是 AWS Backup？
- 什么是 AWS Step Functions？
- 什么是 AWS Elastic Disaster Recovery
- AWS Elastic Disaster Recovery

相关示例：

- Well-Architected 实验室：测试数据的备份与还原

使用故障隔离，以保护您的工作负载

故障隔离边界可将一个工作负载内的故障影响限制于有限数量的组件。边界以外的组件不会受到故障的影响。使用多个故障隔离边界，您可以限制作用于您的工作负载的影响。

最佳实践

- REL10-BP01 将工作负载部署到多个位置
- REL10-BP02 为您的多位置部署选择合适的位置
- REL10-BP03 组件的自动恢复受限于单个位置
- REL10-BP04 采用隔板架构来限制影响范围

REL10-BP01 将工作负载部署到多个位置

将工作负载数据和资源分布到多个可用区，或在必要时分布到多个 AWS 区域。可通过选择不同位置满足各种需求。

在 AWS，服务设计的其中一个基本原则是避免底层物理基础设施中存在单点故障。这促使我们构建使用多个可用区并能灵活应对单个可用区故障的软件和系统。同样，系统也被构建可灵活应对单个计算节点、单个存储卷或单个数据库实例故障。构建依赖冗余组件的系统时，务必要确保组件独立运行；如果是 AWS 区域，组件应自主运行。只有实现了这一点，采用冗余组件的理论可用性计算的优点才能发挥作用。

可用区（AZ，Availability Zone）

AWS 区域由在设计上彼此相互独立的多个可用区组成。每个可用区之间都间隔相当的物理距离，以避免因环境公害（如火灾、洪水和龙卷风）导致的相互关联的故障情况。每个可用区还拥有独立的物理基础设施：专用的公用电源连接、独立备份电源、独立机械服务以及可用区内外的独立网络连接。此设计

将任意这些系统的故障限制在受影响的那一个 AZ 中。尽管可用区在地理位置上相互分离，但它们位于相同的区域中，从而实现高吞吐量、低延迟的联网。整个 AWS 区域（跨多个可用区，由多个物理上独立的设计中心组成）可以视为工作负载的单个逻辑部署目标，包括同步复制数据（例如，在两个数据库之间）的能力。这样一来，您便能在主动/主动或主动/备用配置中使用可用区。

可用区是独立的，因此当工作负载采用了使用多个可用区的架构时，可以提高工作负载的可用性。一些 AWS 服务（包括 Amazon EC2 实例数据面板）作为严格的区级别服务部署，与其所在的可用区共存亡。其他 AZ 中的 Amazon EC2 实例不受影响，可以继续正常工作。与此类似，如果某个可用区中的故障导致 Amazon Aurora 数据库失败，则不受影响的 AZ 中的只读副本 Aurora 实例可以自动提升为主实例。另一方面，区域性 AWS 服务（例如 Amazon DynamoDB）在内部以主动/主动配置的形式使用多个可用区，以实现为该服务设定的可用性设计目标，而且无需您配置 AZ 置放。

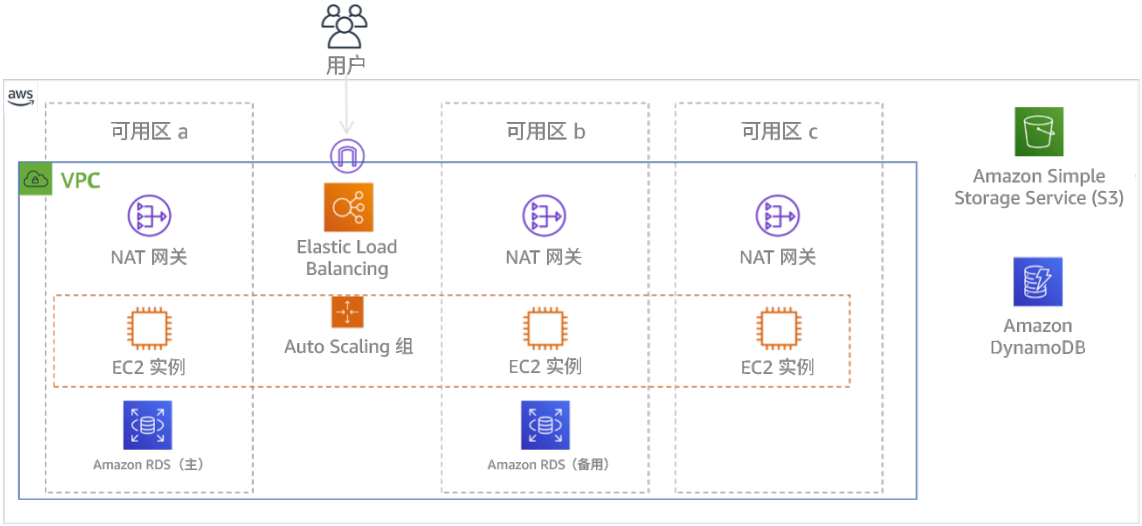


图 9：跨三个可用区部署多层架构。请注意，Amazon S3 和 Amazon DynamoDB 始终会自动部署到多个可用区。而 ELB 也会被部署到所有三个区。

虽然 AWS 控制面板通常提供在整个区域（多个可用区）内管理资源的功能，但某些控制面板（包括 Amazon EC2 和 Amazon EBS）能够将结果筛选到单个可用区。完成筛选后，请求仅在指定可用区中进行处理，从而降低其他可用区的中断风险。此 AWS CLI 示例演示仅从 us-east-2c 可用区中获取 Amazon EC2 实例信息：

```
AWS ec2 describe-instances --filters Name=availability-zone,Values=us-east-2c
```

AWS Local Zones

AWS Local Zones 在其各自的 AWS 区域内的作用与可用区相似，它们可被选择作为区级别 AWS 资源（如子网和 EC2 实例）的置放位置。特别之处在于，它们并不位于相关的 AWS 区域内，而是靠近

目前还未设置 AWS 区域的人口密集的工业和 IT 中心。但是，您还是可以享有高带宽，并且能够在本地区域的本地工作负载与在 AWS 区域内运行的工作负载之间进行安全连接。您应该利用 AWS Local Zones 将工作负载尽量部署在接近用户的地方，以满足低延迟的要求。

Amazon 全球边缘网络

Amazon 全球边缘网络由全球各大城市的边缘站点组成。Amazon CloudFront 使用此网络以较低的延迟向最终用户分发内容。您可以通过 AWS Global Accelerator 在这些边缘站点创建您的工作负载端点，以便在靠近您的用户的 AWS 全球网络进行接入。利用 Amazon API Gateway，使用 CloudFront 分配的边缘优化 API 端点可以通过最近的边缘站点方便客户端访问。

AWS 区域

AWS 区域采用自主设计，因此通过多区域方法，您可以将服务的专用副本部署到每个区域。

多区域方法对于 灾难恢复 策略很常见，用于在偶发的大规模事件中满足恢复目标。请参阅 灾难恢复 (DR) 计划 以了解有关这些策略的更多信息。不过，这里我们的重点是 可用性，旨在达成长期使用中的平均正常运行时间目标。对于高可用性目标，通常可以将多区域架构设计为主动/主动模式，各个服务副本（在其各自的区域中）处于活动状态（为请求提供服务）。

> **推荐**
>
> 大多数工作负载的可用性目标都可通过在单个 AWS 区域内采用多 AZ 策略来实现。只有当工作负载具有极高的可用性要求或者其他业务目标时，才考虑多区域架构，在这些情况下需要使用多区域架构。

AWS 提供了跨区域运行服务的功能。例如，AWS 使用 Amazon Simple Storage Service（Amazon S3）复制、Amazon RDS 只读副本（包括 Aurora 只读副本）和 Amazon DynamoDB 全局表，提供了连续异步数据复制功能。通过连续复制，您的数据版本可近乎实时地供各个活动区域使用。

使用 AWS CloudFormation，您可以跨 AWS 账户和 AWS 区域定义基础设施并一致地进行部署。AWS CloudFormation StackSets 扩展了此功能，允许您通过单个操作，跨多个账户和区域创建、更新或删除 AWS CloudFormation 堆栈。对于 Amazon EC2 实例部署，亚马逊云机器镜像（AMI，Amazon Machine Image）可用于提供诸如硬件配置和已安装软件等信息。您可以实施 Amazon EC2 Image Builder 管道来创建所需的 AMI，并将这些 AMI 复制到您的活动区域。这可以确保这些 Golden AMI 具有您需要部署的所有内容，并可在各个新区域中扩展您的工作负载。

对于路由流量，Amazon Route 53 和 AWS Global Accelerator 均可定义策略来确定哪些用户转向哪个活动的区域端点。使用 Global Accelerator，您可以设置流量转盘，控制导向各个应用程序端点的流量

的百分比。Route 53 支持这种百分比方法，还有多个其他策略可用，包括基于地理位置距离和延迟的策略。Global Accelerator 自动利用 AWS 边缘服务器广泛的网络，尽可能快地将流量载入到 AWS 主干网，从而得到较低的请求延迟。

所有这些功能在执行时，都保留了各个区域的自主性。这种方法有极少的例外，包括我们提供全球边缘交付的服务（例如 Amazon CloudFront 和 Amazon Route 53），以及 AWS Identity and Access Management（IAM）服务的控制面板。大多数服务都完全在单个区域中运行。

本地数据中心

对于在本地数据中心运行的工作负载来说，尽可能打造混合体验。AWS Direct Connect 提供从您的本地到 AWS 的专用网络连接，使您可以同时在两者中运行。

另一个选项是，通过 AWS Outposts 在本地运行 AWS 基础设施和服务。AWS Outposts 是一项完全托管式服务，可将 AWS 基础设施、AWS 服务、API 和工具延伸到您的数据中心。在您的设计中心会安装与 AWS Cloud 中使用的相同硬件基础设施。然后，AWS Outposts 会连接到最近的 AWS 区域。您可以使用 AWS Outposts 支持您的低延迟工作负载，或满足本地数据处理要求。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

- 使用多个可用区和 AWS 区域。将工作负载数据和资源分布到多个可用区，或在必要时分布到多个 AWS 区域。可通过选择不同位置满足各种需求。
  - 区域性服务本质上是跨多个可用区部署的。
    - 这包括 Amazon S3、Amazon DynamoDB 和 AWS Lambda（未连接到 VPC 时）
  - 将容器、实例和基于功能的工作负载部署到多个可用区中。使用包括缓存在内的多可用区数据存储。使用 EC2 Auto Scaling、ECS 任务置放、AWS Lambda 函数配置（在 VPC 中运行时）和 ElastiCache 集群的功能。
    - 部署 Auto Scaling 组时，请使用单独可用区中的子网。
      - 示例：跨多个可用区分布实例
      - Amazon ECS 任务置放策略
      - 配置 AWS Lambda 函数以访问 Amazon VPC 中的资源
      - 选择区域和可用区
    - 部署 Auto Scaling 组时，请使用单独可用区中的子网。
      - 示例：跨多个可用区分布实例
    - 使用 ECS 任务置放参数，并指定数据库子网组。

    - Amazon ECS 任务置放策略
  - 配置要在 VPC 中运行的函数时，请使用多个可用区中的子网。
    - 配置 AWS Lambda 函数以访问 Amazon VPC 中的资源
  - 将多个可用区与 ElastiCache 集群配合使用。
    - 选择区域和可用区
- 如果您的工作负载必须部署到多个区域，请选择一个多区域策略。大多数可靠性需求可通过在单个 AWS 区域中使用多可用区策略来满足。可在必要时使用多区域策略来满足您的业务需求。
  - AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）
    - 备份到另一个 AWS 区域可以让您更加确信，数据在需要时可用。
    - 有些工作负载具有法规要求，需要使用多区域策略。
- 评估您工作负载的 AWS Outposts。如果您的工作负载需要到本地部署数据中心的较低延迟，或具有本地数据处理要求，然后使用 AWS Outposts 在本地运行 AWS 基础设施和服务
  - 什么是 AWS Outposts？
- 确定 AWS Local Zones 是否可以帮助您为用户提供服务。如果您有低延迟要求，请查看 AWS Local Zones 是否距离您的用户较近。如果是，则使用它将工作负载部署到离这些用户较近的位置。
  - AWS Local Zones 常见问题

资源

相关文档：

- AWS 全球基础设施
- AWS Local Zones 常见问题
- Amazon ECS 任务置放策略
- 选择区域和可用区
- 示例：跨多个可用区分布实例
- 全局表：使用 DynamoDB 的多区域复制
- 使用 Amazon Aurora 全局数据库
- 使用 AWS 服务创建多区域应用程序博客系列
- 什么是 AWS Outposts？

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）
- AWS re:Invent 2019：AWS 全球网络基础设施的创新与运营（NET339）

REL10-BP02 为您的多位置部署选择合适的位置

期望结果

要实现高可用性，请始终（在可能时）将您的工作负载组件部署到多个可用区（AZ，Availability Zone），如图 10 中所示。对于具有极高弹性要求的工作负载，请谨慎评估用于多区域架构的选项。

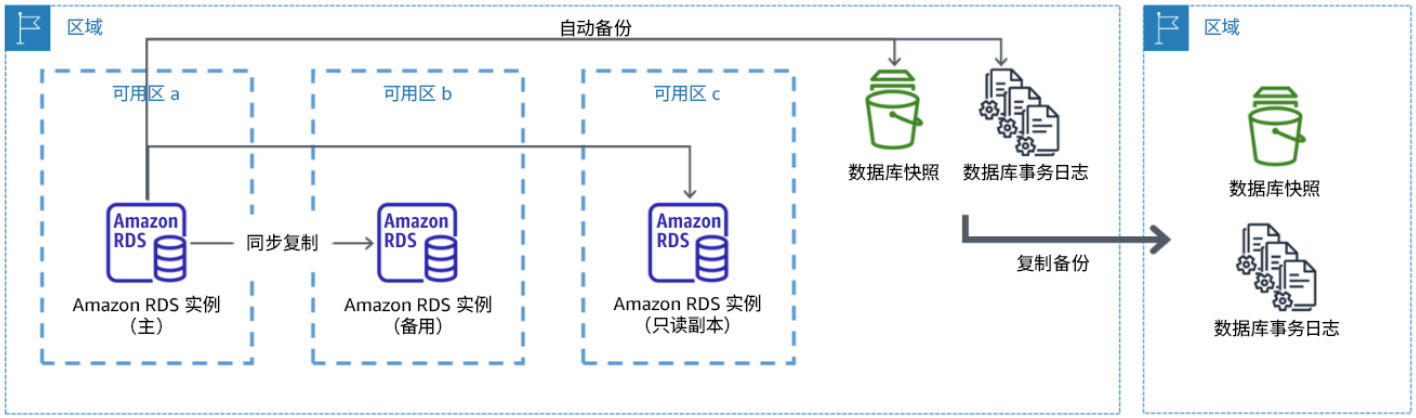


图 10：一个弹性多 AZ 数据库部署，该部署备份到另一个 AWS 区域

常见反模式

- 在多 AZ 架构可以满足需求时选择设计多区域架构。
- 当应用程序部件之间的弹性和多位置需求不同时，没有考虑它们之间的依赖关系。

建立此最佳实践的好处

要实现弹性，您应使用构建防御层的方法。其中一层使用多 AZ，通过构建高度可用的架构，防护较小规模的、更常见的中断。另一个防御层用于防御很少发生的事件，例如大范围的自然灾害和区域级别的中断。这个第二层涉及到设计应用程序的架构来跨越多个 AWS 区域。

- 99.5% 的可用性与 99.99% 的可用性相比，每个月的正常运行时间之差超过 3.5 小时。采用多 AZ 的工作负载的可用性，预期只能达到“四个九”。
- 通过在多个 AZ 中运行工作负载，您可以隔离电力、冷却和网络中的故障，以及火灾和洪水之类的大多数自然灾害。

- 为工作负载实施多区域策略，有助于防御影响到某个国家/地区中较大地理面积的大范围自然灾害，或者区域范围的技术故障。请注意，实施多区域架构会有很高的复杂性，对于大部分工作负载通常来说都是不必要的。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

对于一个可用区的中断或部分丢失而导致的灾难事件，在单个 AWS 区域内的多个可用区中实施高可用工作负载，有助于防范自然灾难和技术灾难。每个 AWS 区域由多个可用区组成，各个可用区之间实现了故障隔离并且间隔相当的物理距离。不过，对于可能造成间隔相当距离的多个可用区组件丢失风险的灾难事件，您应该实施灾难恢复选项，以防范整个区域的自然灾难和技术故障。对于需要极高弹性的工作负载（关键基础设施、与生命健康相关的应用程序、财务系统基础设施等），需要使用多区域策略。

实施步骤

1. 评估您的工作负载并确定需要使用多 AZ 方法（单个 AWS 区域）还是多区域方法才能够满足弹性需求。实施多区域架构来满足这些需求会引入额外的复杂性，因此请谨慎考虑您的使用场景及其需求。弹性需求几乎总是可以使用单个 AWS 区域来满足。在确定您是否需要使用多区域时，请考虑以下可能的需求：
   a. 灾难恢复（DR，Disaster Recovery）：对于一个可用区的中断或部分丢失而导致的灾难事件，在单个 AWS 区域内的多个可用区中实施高可用工作负载，有助于防范自然灾难和技术灾难。对于可能造成间隔相当距离的多个可用区组件丢失风险的灾难事件，您应该实施跨多个区域的灾难恢复，以防范整个区域的自然灾难和技术故障。
   b. 高可用性（HA，High Availability）：多区域架构（在每个区域中使用多个 AZ）可用于实现四个 9 以上（> 99.99%）的可用性。
   c. 堆栈本地化：面向全球受众部署工作负载时，您可以将本地化的堆栈部署在不同的 AWS 区域中，以便服务于这些区域中的受众。本地化可以包括语言、货币和所存储数据的类型。
   d. 靠近用户： 面向全球受众部署工作负载时，您可以通过在靠近最终用户所在位置的 AWS 区域中部署堆栈，从而减少延迟。
   e. 数据驻留：一些工作负载面临着数据驻留要求，来自特定用户的数据必须保留在特定国家/地区的边界内。根据相关的法规，您可以选择将整个堆栈或者仅仅将数据部署到这些边界内的 AWS 区域中。
2. 以下是 AWS 服务提供的一些多 AZ 功能的示例：

a. 为了使用 EC2 或 ECS 保护工作负载，请在计算资源前端部署 Elastic Load Balancer。然后，Elastic Load Balancing 提供解决方案来检测未正常运行的区中的实例，并将流量路由到正常运行的区中。

   i. 开始使用 Application Load Balancers

   ii. 开始使用网络负载均衡器

b. 当 EC2 实例运行不支持负载均衡的现成商用软件时，您可以通过实施多 AZ 灾难恢复方法来实现某种形式的容错能力。

   i. the section called “REL13-BP02 使用定义的恢复策略来实现恢复目标”

c. 对于 Amazon ECS 任务，将您的服务均匀地部署在三个 AZ 上以实现可用性与成本的平衡。

   i. Amazon ECS 可用性最佳实践 | 容器

d. 对于非 Aurora Amazon RDS，您可以选择多 AZ 作为配置选项。在主数据库实例出现故障时，Amazon RDS 会自动提升备用数据库，用于接收其他可用区中的流量。还可以创建多区域只读副本来改进弹性。

   i. Amazon RDS 多可用区部署

   ii. 在不同 AWS 区域中创建只读副本

3. 以下是 AWS 服务提供的一些多区域功能的示例：

a. 对于 Amazon S3 工作负载，当服务自动提供了多 AZ 可用性时，如果需要多区域部署，请考虑多区域接入点。

   i. Amazon S3 中的多区域接入点

b. 对于 DynamoDB 表，此时服务自动提供了多 AZ 可用性，您可以轻松地将现有表转换为全局表来利用多区域的优势。

   i. 将单区域 Amazon DynamoDB 表转换为全局表

c. 如果您的工作负载采用 Application Load Balancers 或网络负载均衡器作为前端，请将流量引导到包含正常运行端点的多个区域，从而使用 AWS Global Accelerator 来改进应用程序的可用性。

   i. AWS Global Accelerator 中标准加速器的端点 – AWS Global Accelerator（amazon.com）

d. 对于利用 AWS EventBridge 的应用程序而言，请考虑使用跨区域总线来将事件转发到您选择的其他区域。

   i. 在 AWS 区域之间发送和接收 Amazon EventBridge 事件

e. 对于 Amazon Aurora 数据库，请考虑使用跨多个 AWS 区域的 Aurora 全局数据库。可以对现有集群进行修改来添加新的区域。

   i. 开始使用 Amazon Aurora 全局数据库

f. 如果您的工作负载包括 AWS Key Management Service（AWS KMS）加密密钥，请考虑多区域密钥是否适合您的应用程序。

   i. AWS KMS 中的多区域密钥

g. 对于其他 AWS 服务功能，请参阅此博客系列中的以下内容： 使用 AWS 服务创建多区域应用程序系列

实施计划的工作量级别： 中到高

资源

相关文档：

- 使用 AWS 服务创建多区域应用程序系列
- AWS 上的灾难恢复（DR，Disaster Recovery）架构，第 IV 部分：多站点主动/主动
- AWS 全球基础设施
- AWS Local Zones 常见问题
- AWS 上的灾难恢复（DR，Disaster Recovery）架构，第 I 部分：云中的恢复策略
- 云中的灾难恢复不相同
- 全局表：使用 DynamoDB 的多区域复制

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）
- Auth0：多区域高可用性架构，可扩展至每月 15亿+ 次登录，并具有自动故障转移功能

相关示例：

- AWS 上的灾难恢复（DR，Disaster Recovery）架构，第 I 部分：云中的恢复策略
- DTCC 实现了本地部署无法企及的弹性
- Expedia Group 使用具有专有 DNS 服务的多区域、多可用区架构来增加应用程序的弹性
- Uber：用于多区域 Kafka 的灾难恢复
- Netflix：实现多区域弹性的主动-主动架构
- 我们如何为 Atlassian Cloud 构建数据驻留
- Intuit TurboTax 跨两个区域运行

REL10-BP03 组件的自动恢复受限于单个位置

如果工作负载的组件只能在单个可用区或本地部署数据中心内运行，您必须利用相关功能在定义的恢复目标内彻底重建工作负载。

在未建立这种最佳实践的情况下暴露的风险等级：中等

实施指导

如果由于技术限制无法使用将工作负载部署到多个位置的最佳实践，您必须实施其他的弹性路径。在这种情况下，您必须让重建必要基础设施、重新部署应用程序和重建必要数据的操作实现自动化。

例如，Amazon EMR 会为相同可用区内的特定集群启动全部节点，因为在相同区内运行集群可以改善作业流的性能，提高数据访问速率。如果这是工作负载弹性所需的必要组件，则您必须设法重新部署集群及其数据。同样对于 Amazon EMR，您还应该通过除多可用区以外的方式对冗余进行预置。您可以预置多个节点。使用 EMR 文件系统（EMRFS），EMR 中的数据可以存储在 Amazon S3 中，进而可以实现跨多个可用区或 AWS 区域复制。

同样，对于 Amazon Redshift 来说，它默认会在您选择的 AWS 区域内随机选择可用区，然后对其中的集群进行预置。相同区内的全部集群节点都会被预置。

对于部署到本地数据中心基于服务器的有状态工作负载，您可以使用 AWS Elastic Disaster Recovery 来保护 AWS 中的工作负载。如果您已在 AWS 中托管，则可以使用 Elastic Disaster Recovery 将工作负载保护到备用可用区或区域。Elastic Disaster Recovery 使用持续块级复制将数据复制到轻量级暂存区域，以便为本地和基于云的应用程序提供快速、可靠的恢复。

实施步骤

1. 实施自我修复。尽可能使用弹性伸缩部署实例或容器。如果不能使用弹性伸缩，则使用 EC2 实例的自动恢复功能，或者基于 Amazon EC2 或 ECS 容器生命周期事件实施自我修复自动化。
   - 将 Amazon EC2 Auto Scaling 组用于对单个实例 IP 地址、私有 IP 地址、弹性 IP 地址和实例元数据没有要求的实例和容器工作负载。
     - 启动模板用户数据可以用于实现自动化，从而让大多数工作负载可以自我修复。
   - 将 Amazon EC2 实例的自动恢复功能用于需要单个实例 IP 地址、私有 IP 地址、弹性 IP 地址和实例元数据的工作负载。
     - 自动恢复功能会在检测到实例故障时，向 SNS 主题发送恢复状态提醒。
   - 在无法使用弹性伸缩或 EC2 恢复的情况下，使用 Amazon EC2 实例生命周期事件或 Amazon ECS 事件实现自动化的自我修复。

    - 使用这些事件调用自动化，该自动化将根据您需要的流程逻辑来修复组件。
- 使用 AWS Elastic Disaster Recovery 保护仅限于单个位置的有状态工作负载。

资源

相关文档：

- Amazon ECS 事件
- Amazon EC2 Auto Scaling 生命周期挂钩
- 恢复您的实例。
- 服务弹性伸缩
- 什么是 Amazon EC2 Auto Scaling？
- AWS Elastic Disaster Recovery

REL10-BP04 采用隔板架构来限制影响范围

实施隔板架构（也称为基于 cell 的架构），将工作负载中故障的影响限制到有限数量的组件上。

期望结果：基于 cell 的架构使用工作负载的多个独立实例，每个实例称为一个 cell。每个 cell 是独立的，不与其他 cell 分享状态，并处理整个工作负载请求的一个子集。这样减少了故障（例如，错误的软件更新）对单个 cell 及其正在处理的请求的潜在影响。如果工作负载使用 10 个 cell 来提供 100 个请求，则在发生故障时，总请求中有 90% 不受故障的影响。

常见反模式：

- 让 cell 无限制地发展。
- 同时将代码更新或部署应用到所有 cell。
- 在不同 cell 之间分享状态或组件（路由器层除外）。
- 向路由器层添加复杂的业务或路由逻辑。
- 没有尽量减少 cell 间的相互作用。

建立此最佳实践的好处：借助基于 cell 的架构，许多常见类型的故障控制在 cell 本身，从而实现了额外的故障隔离。在出现难以控制的故障类型（例如不成功的代码部署，或者是出错或触发特定故障模式的请求，也称为“毒药”请求）时，这些故障边界可以提供弹性。

实施指导

在船舶上，隔板确保船体裂口控制在船体的一个区段内。在复杂的系统中，通常会复制此模式以实现故障隔离。故障隔离边界可将一个工作负载内的故障影响限制于有限数量的组件。边界以外的组件不会受到故障的影响。使用多个故障隔离边界，您可以限制作用于您的工作负载的影响。在 AWS 中，客户可以使用多个可用区和区域来实现故障隔离，但故障隔离的概念可以也可以扩展到工作负载的架构。

整个工作负载是分区的 cell，按分区键进行划分。这个键需要与服务的粒度，或者与使用最小的跨 cell 交互来细分服务工作负载的自然方式保持一致。分区键的示例包括客户 ID、资源 ID 或可以在大多数 API 调用中轻松访问的任何其他参数。cell 路由层根据分区键将请求分发到单个 cell，并向客户端提供单个端点。

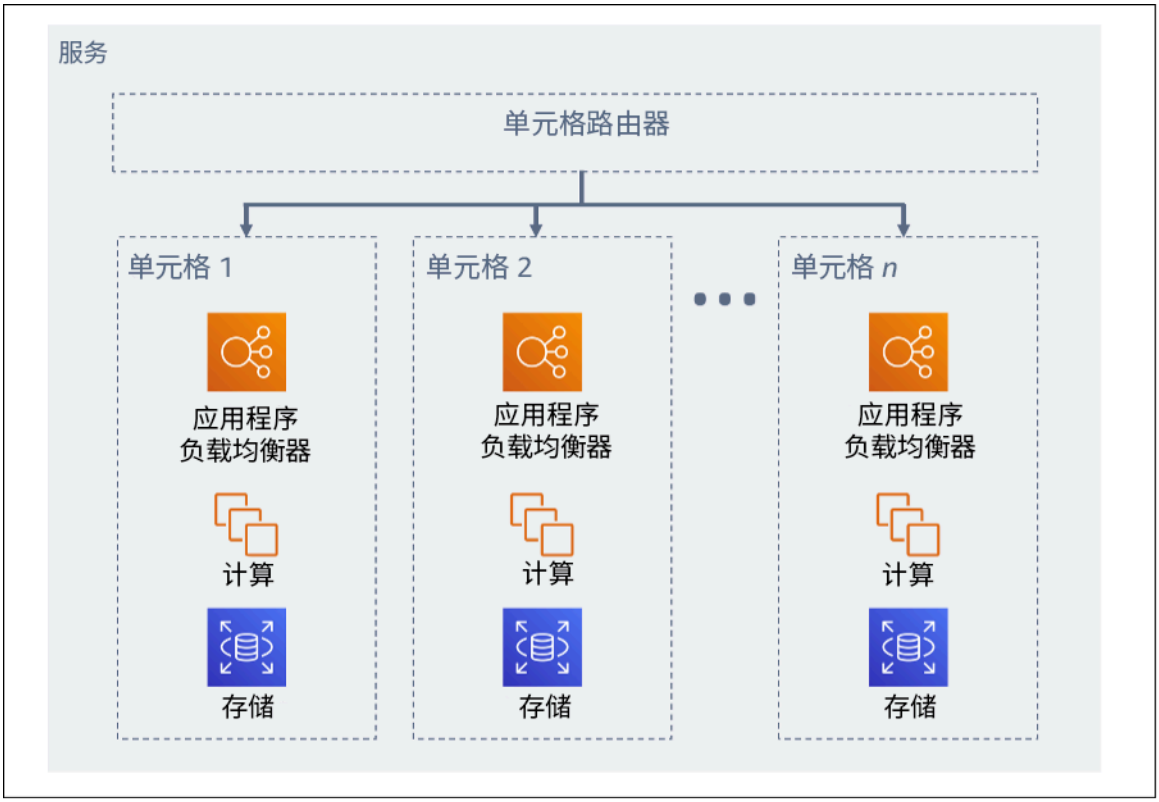


图 11：基于 Cell 的架构

实施步骤

在设计基于 cell 的架构时，需要考虑几个设计注意事项：

1. 分区键：选择分区键时应特别考虑。
   - 它应与服务的粒度，或者与使用最小的跨 cell 交互来细分服务工作负载的自然方式保持一致。示例包括 `## ID` 或 `## ID`。

- 在所有请求中都必须提供分区键，要么直接提供，要么通过很容易由其他参数确定地推断出来的方式提供。

2. 持久 cell 映射：上游服务在其资源的生命周期内应只与单个 cell 交互。
   - 根据工作负载，可能需要使用 cell 迁移策略将数据从一个 cell 迁移到另一个 cell。可能需要进行 cell 迁移的一种情况是：工作负载中的一个特定用户或资源变得太大，要求它具有专用的 cell。
   - Cell 之间不应该共享状态或组件。
   - 因此，应该避免或尽量减少跨 cell 的交互，因为这些交互会在 cell 之间产生依赖关系，从而削弱故障隔离的改进。
3. 路由器层：路由器层是 cell 之间的一个共享组件，因此无法遵循与 cell 相同的分隔策略。
   - 建议路由器层使用分区映射算法以一种计算效率高的方式将请求分发到各个 cell，例如结合加密哈希函数和模块化算法，将分区键映射到 cell。
   - 为避免产生多 cell 影响，路由层必须尽可能保持简单和可横向扩展，这就需要在此层中避免出现复杂的业务逻辑。这还有一个额外的好处，即始终可以轻松地理解其预期行为，从而实现彻底的可测试性。正如 Colm MacCárthaigh 在可靠性、持续工作和安然无忧中所说，简单的设计和持续工作模式可产生可靠的系统并降低抗脆弱性。
4. Cell 大小：Cell 应具有最大大小，不得发展到超过这个大小。
   - 应通过执行彻底的测试来确定最大大小，直至达到临界点并建立安全的运营边际。有关如何实施测试实践的更多详细信息，请参阅 REL07-BP04 对工作负载进行负载测试
   - 总体工作负载增长时应增加额外的 cell，使得工作负载能够随着需求的增加而扩展。
5. 多可用区或多区域策略：应该利用多层弹性来防止出现不同的故障域。
   - 要实现弹性，您应使用构建防御层的方法。其中一层使用多可用区，通过构建高度可用的架构，防止出现较小规模、更常见的中断。另一个防御层用于防御很少发生的事件，例如大范围的自然灾害和区域级别的中断。这个第二层涉及到设计应用程序的架构来跨越多个 AWS 区域。为工作负载实施多区域策略，有助于防御影响到某个国家/地区中较大地理面积的大范围自然灾害，或者区域范围的技术故障。请注意，实施多区域架构会有很高的复杂性，对于大部分工作负载通常来说都是不必要的。有关更多详细信息，请参阅 REL10-BP02 为您的多位置部署选择合适的位置。
6. 代码部署：交错的代码部署策略应该优于同时将代码更改部署到所有 cell。
   - 这将有助于最大限度地减少由于部署不当或人为错误而导致多个 cell 可能出现故障。有关更多详细信息，请参阅自动执行安全、不需要人工介入的部署。

在未建立这种最佳实践的情况下暴露的风险等级：高

资源

相关最佳实践：

- REL07-BP04 对工作负载进行负载测试
- REL10-BP02 为您的多位置部署选择合适的位置

相关文档：

- 可靠性、持续工作和安然无忧
- AWS 和分隔
- 采用随机分区进行工作负载隔离
- 自动执行安全、不需要人工介入的部署

相关视频：

- AWS re:Invent 2018：闭环系统和开放思维：如何掌控不同规模的系统
- AWS re:Invent 2018：AWS 如何将故障的影响范围最小化（ARC338）
- 随机分片：AWS re:Invent 2019：Amazon Builders' Library 简介（DOP328）
- 2021 AWS 峰会（澳大利亚和新西兰）- 故障在所难免：针对弹性而设计

相关示例：

- Well-Architected 实验室 - 使用随机分片进行故障隔离

将工作负载设计为能够承受组件故障的影响

在设计具有高可用性和较短平均恢复时间（MTTR）要求的工作负载时必须考虑到弹性。

最佳实践

- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP02 失效转移到运行状况良好的资源
- REL11-BP03 自动修复所有层
- REL11-BP04 恢复期间依赖于数据平面而不是控制平面

- REL11-BP05 使用静态稳定性来防止双模态行为
- REL11-BP06 当事件影响可用性时发出通知
- REL11-BP07 构造您的产品以满足可用性目标和正常运行时间服务等级协议（SLA）

REL11-BP01 监控工作负载的所有组件以检测故障

持续监控您的工作负载的运行状况，以便您和您的自动化系统立即发现任何故障或性能下降情况。监控基于商业价值的关键性能指标（KPI）。

所有恢复和修复机制必须从快速检测问题的能力入手。首先，应该检测技术故障并加以解决。不过，可用性基于您的工作负载创造商业价值的能力，因此衡量它的关键性能指标（KPI）需要成为您的检测和补救策略的一部分。

期望的结果： 独立监控工作负载的重要组成部分，以检测故障发生的时间和位置，并发出警报。

常见反模式：

- 由于未配置警报，因此在发生中断时不会进行通知。
- 虽然存在警报，但只有在达到阈值时才会发出警报，导致没有足够的响应时间。
- 收集指标的频率不够高，无法满足恢复时间目标（RTO）。
- 工作负载中只有面向客户的接口才会被主动监控。
- 只收集技术指标，不收集业务功能指标。
- 没有衡量工作负载用户体验的指标。
- 创建的监控太多。

建立此最佳实践的好处： 如果您在所有层面都设置了适当的监控，则可以通过减少检测时间来缩短恢复时间。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

确定将接受审查以决定是否监控的所有工作负载。确定工作负载中所有需要监控的组成部分后，就需要确定监控间隔。根据检测故障所花费的时间，监控间隔将直接影响启动恢复的速度。平均检测时间（MTTD）是从故障发生到修复操作开始之间的时间。服务清单应广泛而完整。

监控必须覆盖应用程序堆栈的所有层，包括应用、平台、基础设施和网络。

您的监控策略应考虑以下因素的影响： 灰色故障。有关灰色故障的更多详细信息，请参阅《Advanced Multi-AZ Resilience Patterns》白皮书中的 Gray failures 。

实施步骤

- 您的监控间隔取决于您必须以多快的速度恢复。您的恢复时间取决于恢复所需的时间，因此您在确定收集频率时，必须考虑此时间和恢复时间目标（RTO）。
- 为组件和托管服务配置详细监控。
  - 确定 详细监控对于 EC2 实例 和 Auto Scaling 来说是否有必要。详细监控以 1 分钟为间隔提供指标，默认监控以 5 分钟为间隔提供指标。
  - 确定 增强监控 对于 RDS 来说是否有必要。增强监控使用 RDS 实例上的代理，来获取关于不同进程或线程的有用信息。
  - 确定以下各项的关键无服务器组件的监控要求： Lambda、 API Gateway、 Amazon EKS、 Amazon ECS，以及所有类型的 负载均衡器。
  - 确定以下各项的存储组件的监控要求： Amazon S3、 Amazon FSx、 Amazon EFS和 Amazon EBS。
- 创建 自定义指标 来衡量业务关键绩效指标（KPI）。工作负载会实现关键业务功能，这些功能应用作 KPI 来协助在发生间接问题时予以识别。
- 使用用户金丝雀来监控用户的故障体验。 可运行和模拟客户行为的综合事务测试 （又称为“金丝雀测试”，但不要和金丝雀部署相混淆）是最重要的测试流程之一。从不同的远程位置针对您的工作负载端点持续地运行此类测试。
- 创建 自定义指标 来跟踪用户体验。如果您可以衡量客户体验，就可以确定发生了客户体验下降。
- 设置警报 ，以在检测到工作负载的任何部分未正常运行时发出警报，并指示什么时候自动扩展资源。警报可以直观地显示在控制面板上，通过 Amazon SNS 或电子邮件发送，并与 Auto Scaling 结合使用来扩展或缩减工作负载资源。
- 创建 控制面板 以可视化形式呈现指标。可以使用控制面板直观地查看趋势、离群值和表示其他潜在问题的指标，或者提供您可能需要进行调查的问题的指示。
- 为您的服务创建 分布式跟踪监控 。使用分布式监控，您可以了解应用程序及其底层服务的运行情况，以便确定和诊断性能问题及错误的根本原因。
- 在单独的区域和账户中创建监控系统（使用 CloudWatch 或 X-Ray）控制面板和数据收集。
- 为 Amazon Health Aware 集成监控功能，以便监控可能出现性能下降的 AWS 资源。对于关键业务工作负载，此解决方案可提供对 AWS 服务的主动实时警报的访问。

资源

相关最佳实践：

- 可用性定义
- REL11-BP06 当事件影响可用性时发出通知

相关文档：

- Amazon CloudWatch Synthetics 使您能够创建用户金丝雀
- 为您的实例启用或禁用详细监控
- 增强监控
- Monitoring Your Auto Scaling Groups and Instances Using Amazon CloudWatch
- 发布自定义指标
- 使用 Amazon CloudWatch 警报
- 使用 CloudWatch 控制面板
- 跨区域跨账户使用 CloudWatch 控制面板
- 跨区域跨账户使用 X-Ray 跟踪
- 了解可用性
- 实施 Amazon Health Aware（AHA）

相关视频：

- Mitigating gray failures

相关示例：

- Well-Architected 实验室：第 300 级：实施运行状况检查和管理依赖项以提高可靠性
- 可观测性研讨会：探索 X-Ray

相关工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP02 失效转移到运行状况良好的资源

如果资源发生故障，运行状况良好的资源应继续为请求提供服务。对于位置受损（如可用区或 AWS 区域），确保您拥有适当的系统，以失效转移到未受损位置内运行状况良好的资源。

设计服务时，应在资源、可用区或区域之间分配负载。因此，可以通过将流量转移到运行状况良好的剩余资源，缓解单个资源的故障或损坏。考虑在出现故障时如何发现服务并将流量路由到相应服务。

在设计服务时要考虑故障恢复。在 AWS，我们设计服务时会尽量缩短从故障恢复的时间并降低对数据的影响。我们的服务主要使用的数据存储，只有在数据持久存储在一个区域中的多个副本之后，才会确认请求。它们被构建为使用基于单元格的隔离，并使用可用区提供的故障隔离功能。我们在自己的运营过程中广泛使用自动化。我们还将替换和重新启动功能优化为可从中断快速恢复。

允许失效转移的模式和设计因各项 AWS 平台服务而异。许多 AWS 原生托管服务本质上跨多个可用区（如 Lambda 或 API Gateway）。其他 AWS 服务（如 EC2 和 EKS）需要特定的最佳实践设计，来支持跨可用区的资源或数据存储的失效转移。

应设置监控功能，以检查失效转移资源是否正常，跟踪资源失效转移的进度，并监控业务流程的恢复情况。

期望的结果： 系统能够自动使用新资源从降级中恢复，或手动恢复。

常见反模式：

- 规划和设计阶段未考虑如何应对失败。
- 未确立 RTO 和 RPO。
- 监控不足，无法检测出故障的资源。
- 适当隔离故障域。
- 不考虑多区域失效转移。
- 在决定是否进行失效转移时，故障检测过于敏感或过于激进。
- 未测试或验证失效转移设计。
- 执行自动修复，但不通知需要进行该修复。
- 缺少缓冲期，无法避免太快进行失效自动恢复。

建立此最佳实践的好处： 您可以构建更具韧性的系统，在遇到故障时，通过正常降级和快速恢复，保持可靠性。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

AWS 服务（例如 Elastic Load Balancing 和 Amazon EC2 Auto Scaling）有助于跨资源和可用区分配负载。因此，可以通过将流量转移到运行状况良好的剩余资源，缓解单个资源（例如 EC2 实例）的故障或可用区的损坏。

对于多区域工作负载，设计就比较复杂。例如，跨区域只读副本允许您将数据部署到多个 AWS 区域。但是，仍需要进行失效转移才能将只读副本提升为主副本，然后将流量指向新的终端节点。Amazon Route 53、Route 53、Route 53 ARC、CloudFront 和 AWS Global Accelerator 可以协助跨 AWS 区域路由流量。

Amazon S3、Lambda、API Gateway、Amazon SQS、Amazon SNS、Amazon SES、Amazon Pinpoint、Amazon ECR、AWS Certificate Manager、EventBridge 或 Amazon DynamoDB 等 AWS 服务将通过 AWS 自动部署到多个可用区。如果出现故障，这些 AWS 服务会自动将流量路由到状态正常的位置。数据在多个可用区中进行冗余存储，并保持可用。

对于 Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon EKS 或 Amazon ECS，多可用区是一个配置选项。如果启动了失效转移，AWS 可以将流量引导到运行正常的实例。此失效转移操作可由 AWS 执行，或根据客户要求执行

对于 Amazon EC2 实例、Amazon Redshift、Amazon ECS 任务或 Amazon EKS Pod，您要选择部署到哪些可用区。对于某些设计，Elastic Load Balancing 会提供解决方案以检测运行不正常的可用区内的实例，并将流量路由至运行正常的可用区。Elastic Load Balancing 还可以将流量路由至本地数据中心内的组件。

对于多区域流量失效转移，重新路由可以利用 Amazon Route 53、Route 53 ARC、AWS Global Accelerator、Route 53 Private DNS for VPCs 或 CloudFront 来提供一种方法，以定义互联网域并分配路由策略（包括运行状况检查），从而将流量路由到运行正常的区域。AWS Global Accelerator 提供静态 IP 地址，这些地址充当应用程序的固定入口点，然后使用 AWS 全球网络而不是互联网来路由到您选择的 AWS 区域中的端点，以获得更高的性能和可靠性。

实施步骤

- 为所有相应的应用程序和服务创建失效转移设计。隔离每个架构组件，为每个组件创建符合 RTO 和 RPO 的失效转移设计。
- 使用失效转移计划所需的所有服务配置底层环境（例如开发或测试）。使用基础设施即代码（IaC）部署解决方案，以确保可重复性。
- 配置恢复站点（例如第二个区域）以实施和测试失效转移设计。如有必要，可以临时配置测试资源以限制额外成本。

- 确定哪些失效转移计划由 AWS 自动执行，哪些可以通过 DevOps 流程自动执行，哪些可能需要手动执行。记录并测量每项服务的 RTO 和 RPO。
- 创建失效转移行动手册，包括对每个资源、应用程序和服务进行失效转移的所有步骤。
- 创建失效自动恢复行动手册，包括对每个资源、应用程序和服务进行失效自动恢复的所有步骤（包含时机）
- 制定计划以启动和演练行动手册。使用模拟和混沌测试来测试行动手册步骤和自动化功能。
- 对于位置受损（如可用区或 AWS 区域），确保您拥有适当的系统，以失效转移到未受损位置内运行状况良好的资源。在测试失效转移之前，请检查配额、自动扩展级别和正在运行的资源。

资源

相关的 Well-Architected 最佳实践：

- REL13 - 制定灾难恢复计划
- REL10 - 使用故障隔离来保护您的工作负载

相关文档：

- 设置 RTO 和 RPO 目标
- 使用应用程序负载均衡器设置 Route 53 ARC
- 使用 Route 53 加权路由进行失效转移
- 使用 Route 53 ARC 进行灾难恢复
- 带自动扩缩功能的 EC2
- EC2 部署 - 多可用区
- ECS 部署 - 多可用区
- 使用 Route 53 ARC 切换流量
- 使用 Application Load Balancer 和失效转移的 Lambda
- ACM 复制和失效转移
- Parameter Store 复制和失效转移
- ECR 跨区域复制和失效转移
- Secrets Manager 跨区域复制配置
- 为 EFS 和失效转移启用跨区域复制

- EFS 跨区域复制和失效转移
- 网络失效转移
- 使用 MRAP 的 S3 端点失效转移
- 为 S3 创建跨区域复制
- 使用 Route 53 ARC 对区域 API Gateway 进行失效转移
- 使用多区域全球加速器进行失效转移
- 使用 DRS 进行失效转移
- 使用 Amazon Route 53 创建灾难恢复机制

相关示例：

- Disaster Recovery on AWS
- Elastic Disaster Recovery on AWS

REL11-BP03 自动修复所有层

在检测到故障时，使用自动化功能执行修复操作。降级可能会通过内部服务机制自动修复，也可能需要通过补救措施重启或移除资源。

对于自我管理型应用程序和跨区域修复，可以从 现有最佳实践中获取恢复设计和自动修复流程。

重启或移除资源是修复故障的重要方法。最佳实践是尽可能使服务为无状态。这可以防止重启资源时数据丢失或可用性受损。在云中，作为重启的一部分，您可以（而且在一般情况下也应该）替换完整的资源（例如计算实例或无服务器函数）。重启本身是从故障恢复的简单而可靠的方法。工作负载中会发生很多不同类型的故障。故障可能发生在硬件、软件、通信和操作上。

重启或重试也适用于网络请求。向网络超时以及依赖项返回错误的依赖性故障应用相同的恢复方法。这两个事件对系统具有类似的影响，可应用类似的采用指数回退和抖动的有限重试策略，而不是尝试将各个事件当作特例进行处理。重启功能是面向恢复的计算和高可用性集群架构的特色恢复机制。

期望的结果： 执行自动操作以修复检测到的故障。

常见反模式：

- 预置资源，而不进行自动扩缩。
- 在实例或容器中单独部署应用程序。

- 在不使用自动恢复的情况下，部署无法部署到多个位置的应用程序。
- 手动修复弹性伸缩和自动恢复无法修复的应用程序。
- 缺乏对数据库进行失效转移的自动化功能。
- 缺乏将流量重新路由到新端点的自动化方法。
- 缺乏存储复制功能。

建立此最佳实践的好处： 自动修复可以缩短平均恢复时间，并提高可用性。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

Amazon EKS 或其他 Kubernetes 服务的设计应包括最小和最大副本集或有状态集，以及最小集群和节点组大小。这些机制提供了最低数量的持续可用处理资源，同时使用 Kubernetes 控制平面自动修复任何故障。

使用计算集群通过负载均衡器访问的设计模式应利用 Auto Scaling 组。Elastic Load Balancing（ELB）自动在一个或多个可用区（AZ）中的多个目标和虚拟设备之间分配传入的应用程序流量。

对于不使用负载平衡的基于集群计算的设计，其规模至少应能承受一个节点的中断。这将允许服务在恢复新节点时，使自身以可能降低的容量维持运行状态。示例服务有 Mongo、DynamoDB Accelerator、Amazon Redshift、Amazon EMR、Cassandra、Kafka、MSK-EC2、Couchbase、ELK 和 Amazon OpenSearch Service。其中许多服务都可以设计额外的自动修复功能。某些集群技术必须在节点丢失时生成警报，从而触发自动或手动工作流程来重新创建新节点。可以使用 AWS Systems Manager 来自动执行此工作流程，以快速修复问题。

Amazon EventBridge 可用于监控和筛选事件，例如 CloudWatch 警报或其他 AWS 服务中的状态更改。然后，它可以根据事件信息调用 AWS Lambda、Systems Manager Automation 或其他目标，对您的工作负载运行自定义修复逻辑。可以将 Amazon EC2 Auto Scaling 配置为检查 EC2 实例的运行状况。若实例处于正在运行以外的任何状态，或系统状态受损，Amazon EC2 Auto Scaling 会认为实例的运行不正常，并且启动替换实例。针对大规模替换（例如整个可用区丢失），静态稳定性更适合用来实现高可用性。

实施步骤

- 使用 Auto Scaling 组在工作负载中部署层。 Auto Scaling 可以对无状态应用程序执行自我修复，以及添加或移除容量。

- 对于前面提到的计算实例，可使用 负载均衡 ，然后选择合适的负载均衡器类型。
- 考虑 Amazon RDS 修复。对于备用实例，请配置 自动失效转移 到备用实例。针对 Amazon RDS 只读副本，需要自动化工作流程，使只读副本成为主副本。
- 在部署了应用程序的 EC2 实例上实施自动恢复 ，这些应用程序无法部署到多个位置，但在故障后允许重新启动。当无法将应用程序部署到多个位置时，可以使用自动恢复来替换发生故障的硬件并重新启动实例。将保留实例元数据和关联的 IP 地址，以及 EBS 卷 和 Amazon Elastic File System 或 适用于 Lustre 的文件系统 和 适用于 Windows 的文件系统的挂载点。使用 AWS OpsWorks时，您可以在层级别配置 EC2 实例的自动修复。
- 当您无法使用自动扩展或自动恢复时，或者自动恢复失败时，使用 AWS Step Functions 和 AWS Lambda 实施自动恢复。当您无法使用自动扩展，并且无法使用自动恢复或自动恢复失败时，可以使用 AWS Step Functions 和 AWS Lambda 进行自动修复。
- Amazon EventBridge 可用于监控和筛选事件，例如 CloudWatch 警报 或其他 AWS 服务状态的变化。根据事件信息，它可以调用 AWS Lambda（或其他目标），在您的工作负载上运行自定义修复逻辑。

资源

相关最佳实践：

- 可用性定义
- REL11-BP01 监控工作负载的所有组件以检测故障

相关文档：

- AWS Auto Scaling 的工作原理
- Amazon EC2 Automatic Recovery
- Amazon Elastic Block Store（Amazon EBS）
- Amazon Elastic File System（Amazon EFS）
- 什么是 Amazon FSx for Lustre？
- 什么是 Amazon FSx for Windows File Server？
- AWS OpsWorks：使用自动修复来更换失败的实例
- 什么是 AWS Step Functions？
- 什么是 AWS Lambda？
- 什么是 Amazon EventBridge？

- 使用 Amazon CloudWatch 警报
- Amazon RDS 失效转移
- SSM - Systems Manager 自动化
- 弹性架构最佳实践

相关视频：

- Automatically Provision and Scale OpenSearch Service
- Amazon RDS Failover Automatically

相关示例：

- Auto Scaling 研讨会
- Amazon RDS 失效转移研讨会

相关工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP04 恢复期间依赖于数据平面而不是控制平面

控制平面提供用于创建、读取和描述、更新、删除和列出（CRUDL）资源的管理 API，而数据平面则处理日常服务流量。在对可能影响弹性的事件实施恢复或缓解响应时，着眼于使用最少数量的控制平面操作，来实现对服务的恢复、重新扩展、恢复、修复或失效转移。在这些降级事件期间，数据平面操作应凌驾于任何活动之上。

例如，以下是所有控制平面操作：启动新的计算实例、创建数据块存储和描述队列服务。启动计算实例时，控制平面必须执行多项任务，例如查找具有容量的物理主机、分配网络接口、准备本地数据块存储卷、生成凭证以及添加安全规则。控制平面往往是复杂的编排。

期望的结果： 当资源进入受损状态时，系统能够通过将流量从受损资源转移到正常运行资源，来自动或手动恢复。

常见反模式：

- 依赖于通过更改 DNS 记录来重新路由流量。

- 由于预置资源不足，依赖控制平面扩展操作来替换受损组件。
- 依靠广泛、多服务、多 API 控制平面操作来修复任何类别的受损情况。

建立此最佳实践的好处： 提高自动修复的成功率可以缩短平均恢复时间，并提高工作负载的可用性。

未建立这种最佳实践的情况下暴露的风险等级： 中。对于某些类型的服务降级，控制平面会受到影响。依赖于大量使用控制平面进行修复可能会增加恢复时间（RTO）和平均恢复时间（MTTR）。

实施指导

要限制数据平面操作，请评估每项服务，以了解恢复服务所需的操作。

利用 Amazon Route 53 Application Recovery Controller 来转移 DNS 流量。这些功能持续监控您的应用程序从故障中恢复的能力，并使您能够跨多个 AWS 区域、可用区和本地部署控制您的应用程序恢复。

Route 53 路由策略使用控制平面，因此不要依赖控制平面进行恢复。Route 53 数据平面会回复 DNS 查询，并执行和评估运行状况检查。它们分布在全球各地，专为 100% 可用性的服务等级协议（SLA）而设计。

您用于创建、更新和删除 Route 53 资源的 Route 53 管理 API 和控制台是在控制平面上运行的，而这些控制平面设计用于优先考虑您在管理 DNS 时所需的强一致性和持久性。为了实现这一点，控制平面位于单个区域“美国东部（弗吉尼亚州北部）”中。虽然这两个系统都非常可靠，但控制平面不包含在 SLA 中。在极少数情况下，数据平面的弹性设计允许它保持可用性，而控制平面做不到。对于灾难恢复和失效转移机制，使用数据平面功能可提供尽可能高的可靠性。

对于 Amazon EC2，使用静态稳定性设计来限制控制平面操作。控制平面操作包括单独扩展资源，或使用 Auto Scaling 群组（ASG）来扩展资源。要获得最高级别的弹性，请在集群中配置足够的容量用于失效转移。如果必须限制此容量阈值，请在整个端到端系统上设置限制，以安全地限制到达有限资源集的总流量。

对于诸如 Amazon DynamoDB、Amazon API Gateway、负载均衡器和 AWS Lambda 无服务器之类的服务，使用这些服务时可以利用数据平面。但是，创建新函数、负载均衡器、API 网关或 DynamoDB 表是一项控制平面操作，应在降级之前完成，以便为事件和演练失效转移操作做准备。对于 Amazon RDS，数据平面操作允许访问数据。

有关数据平面、控制平面以及 AWS 如何构建服务以满足高可用性目标的更多信息，请参阅 使用可用区的静态稳定性。

了解哪些操作位于数据平面，哪些位于控制平面。

实施步骤

对于降级事件后需要恢复的每个工作负载，请评估失效转移运行手册、高可用性设计、自动修复设计或 HA 资源恢复计划。确定可能被视为控制平面操作的每个操作。

考虑将控制平面操作更改为数据平面操作：

- Auto Scaling（控制平面）与预扩展 Amazon EC2 资源（数据平面）的比较
- 迁移到 Lambda 及其扩展方法（数据平面）或 Amazon EC2 和 ASG（控制平面）
- 评估任何使用 Kubernetes 的设计以及控制平面操作的性质。在 Kubernetes 中，添加 Pod 是一项数据平面操作。操作应仅限于添加 Pod 而不是添加节点。使用 过度预置的节点 是限制控制平面操作的首选方法

考虑允许数据平面操作影响相同修复措施的其他方法。

- Route 53 记录更改（控制平面）或 Route 53 ARC（数据平面）
- Route 53 运行状况检查以获取更多自动更新

如果服务是任务关键型，可考虑使用辅助区域中的某些服务，以便在不受影响的区域内进行更多控制平面和数据平面操作。

- 主区域中的 Amazon EC2 Auto Scaling 或 Amazon EKS 与辅助区域中的 Amazon EC2 Auto Scaling 或 Amazon EKS 的比较，以及将流量路由到辅助区域（控制平面操作）
- 在辅助区域中创建只读副本或在主区域中尝试相同的操作（控制平面操作）

资源

相关最佳实践：

- 可用性定义
- REL11-BP01 监控工作负载的所有组件以检测故障

相关文档：

- APN 合作伙伴：可以帮助您实现容错自动化的合作伙伴
- AWS Marketplace：可以支持容错的产品
- Amazon Builders' Library：通过控制较小的服务来避免分布式系统中出现过载

- Amazon DynamoDB API（控制平面和数据平面）
- AWS Lambda 执行（拆分为控制平面和数据平面）
- AWS Elemental MediaStore 数据平面
- 使用 Amazon Route 53 应用程序恢复控制器构建高弹性应用程序，第 1 部分：单区域堆栈
- 使用 Amazon Route 53 应用程序恢复控制器构建高弹性应用程序，第 2 部分：多区域堆栈
- 使用 Amazon Route 53 创建灾难恢复机制
- 什么是 Route 53 应用程序恢复控制器？
- Kubernetes 控制平面和数据平面

相关视频：

- Back to Basics - Using Static Stability
- Building resilient multi-site workloads using AWS global services

相关示例：

- Amazon Route 53 应用程序恢复控制器简介
- Amazon Builders' Library：通过控制较小的服务来避免分布式系统中出现过载
- 使用 Amazon Route 53 应用程序恢复控制器构建高弹性应用程序，第 1 部分：单区域堆栈
- 使用 Amazon Route 53 应用程序恢复控制器构建高弹性应用程序，第 2 部分：多区域堆栈
- 使用可用区的静态稳定性

相关工具：

- Amazon CloudWatch
- AWS X-Ray

REL11-BP05 使用静态稳定性来防止双模态行为

工作负载应具有静态稳定性，并且仅在单一正常模式下运行。双模态行为是指工作负载在正常模式和故障模式下表现出不同的行为。

例如，您可能会尝试在不同的可用区中启动新实例，以便从可用区故障中恢复。在故障模式下，这可能会导致双模态响应。您应该构建静态稳定的工作负载，并且仅在一个模式下运行。在此示例中，这些实

例应在出现故障之前，已经在第二个可用区中预置。这种静态稳定性设计可确保工作负载仅在单一模式下运行。

期望的结果： 在正常模式和故障模式下，工作负载不会表现出双模态行为。

常见反模式：

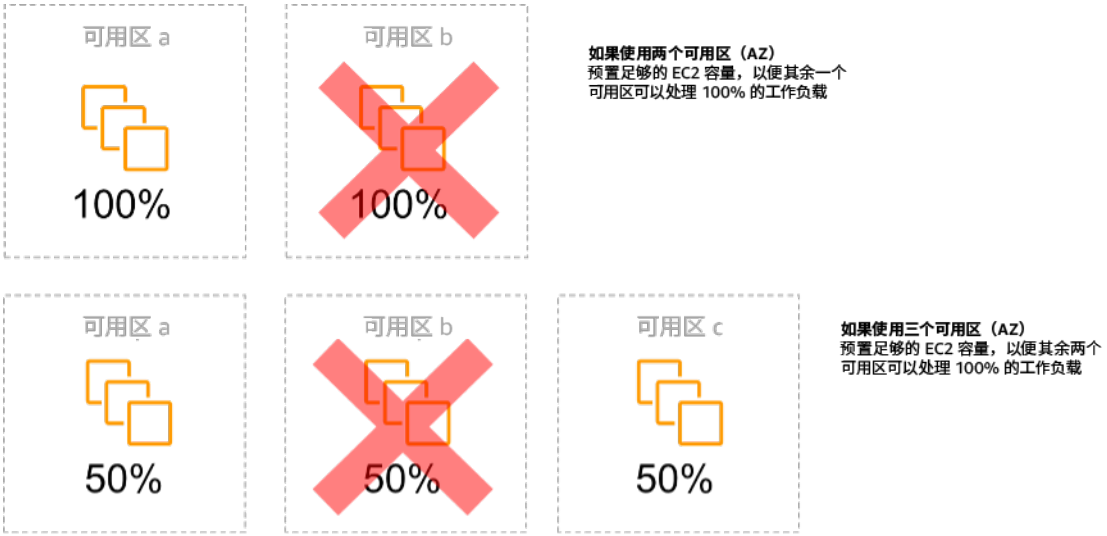
- 假设无论故障范围如何，始终可以预置资源。
- 尝试在故障期间动态获取资源。
- 在出现故障之前，没有跨可用区或跨区域预置足够的资源。
- 仅为计算资源考虑了静态稳定设计。

建立此最佳实践的好处： 采用静态稳定设计运行的工作负载，在正常情况和故障事件期间能够得到可预测的结果。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

当工作负载在正常模式和故障模式下展现出不同的行为时，这就是双模态行为（例如，在可用区发生故障时依赖于启动新的实例）。双模态行为的一个例子是，稳定 Amazon EC2 设计在每个可用区中预置了足够的实例，用于处理在移除了一个可用区时的工作负载。Elastic Load Balancing 或 Amazon Route 53 运行状况将执行检查，并将负载从受损实例上移出。在流量转移以后，使用 AWS Auto Scaling 异步替换故障可用区的实例，并在运行正常的可用区内启动。适用于计算部署（如 EC2 实例或容器）的静态稳定性将提供最高水平的可靠性。



跨可用区的 EC2 实例的静态稳定性

您必须就这一模型的成本以及在所有情况下保持工作负载弹性的业务价值进行权衡。预置较少的计算容量并依赖于在出现故障时启动新实例，这种方法的成本较低，但是对于大规模故障（例如可用区或区域受损），这种方法效果较差，因为它既依赖于运营层面，也依赖未受影响的可用区或区域中是否有足够的可用资源。

您的解决方案还需要在工作负载的可靠性和成本需求之间做出取舍。静态稳定性架构适用于各种架构，包括跨多个可用区的计算实例、数据库只读副本设计、Kubernetes（Amazon EKS）集群设计和多区域失效转移架构。

您还可以在每个可用区中使用更多资源，从而实施具有更好的静态稳定性的设计。通过添加更多可用区，您可以减少实现静态稳定性所需的额外计算容量。

双模态行为的一个例子是，网络超时可能会导致系统尝试刷新整个系统的配置状态。这会向另一个组件添加意外负载，进而可能导致该组件出现故障，引发其他意外后果。此负面的反馈环路会影响您的工作负载的可用性。相反，您可以构建静态稳定的系统，并且仅在一个模式下运行。静态稳定的设计会持续工作，并且始终定期刷新配置状态。当调用失败时，工作负载将使用先前缓存的值并启动警报。

双模态行为的另一个示例是允许客户端在故障发生时绕过您的工作负载缓存。这看起来似乎是可以满足客户端需求的解决方案，但它会明显改变您的工作负载的需求，而且很有可能导致故障。

评估关键工作负载，以确定哪些工作负载需要这种弹性设计。对于被视为关键的工作负载，必须审核每个应用程序组件。需要静态稳定性评估的服务类型示例包括：

- 计算：Amazon EC2、EKS-EC2、ECS-EC2、EMR-EC2
- 数据库：Amazon Redshift、Amazon RDS、Amazon Aurora
- 存储：Amazon S3（单区）、Amazon EFS（挂载）、Amazon FSx（挂载）
- 负载均衡器： 在某些设计下

实施步骤

- 构建静态稳定的系统，并且仅在一个模式下运行。在这种情况下，应在每个可用区或区域中预置足够的实例，以便在移除了一个可用区或区域时处理工作负载容量。您可以使用多种服务来路由到正常运行的资源，例如：
  - 跨区域 DNS 路由
  - MRAP Amazon S3 多区域路由
  - AWS Global Accelerator
  - Amazon Route 53 Application Recovery Controller

- 配置 数据库只读副本 ，来应对单个主实例或只读副本丢失的情况。如果流量由只读副本提供服务，则每个可用区和每个区域中的资源数量，应等于可用区或区域出现故障时的总体需求量。
- 在 Amazon S3 存储中配置关键数据，设计为在可用区出现故障时可确保所存储数据的静态稳定性。如果使用 Amazon S3 One Zone-IA 存储类，则不应将其视为静态稳定，因为丢失该可用区会将对所存储数据的可访问性降到最低。
- 负载均衡器 有时服务于特定可用区，这可能是因为配置不正确，也可能是有意设计成这样。在这种情况下，静态稳定的设计可能需要采用更复杂的设计，将工作负载分布到多个可用区。出于安全、延迟或成本方面的考虑，可能会使用最初的设计来减少区域间流量。

资源

相关的 Well-Architected 最佳实践：

- 可用性定义
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP04 恢复期间依赖于数据平面而不是控制平面

相关文档：

- 尽可能减小灾难恢复计划中的依赖关系
- Amazon Builders' Library：使用可用区的静态稳定性
- 故障隔离界限
- 使用可用区的静态稳定性
- 多可用区 RDS
- 尽可能减小灾难恢复计划中的依赖关系
- 跨区域 DNS 路由
- MRAP Amazon S3 多区域路由
- AWS Global Accelerator
- Route 53 ARC
- 单区 Amazon S3
- 跨可用区负载均衡

相关视频：

- Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

相关示例：

- Amazon Builders' Library：使用可用区的静态稳定性

REL11-BP06 当事件影响可用性时发出通知

在检测到突破阈值时发送通知，即使导致问题的事件已自动解决。

自动修复使您的工作负载变得可靠。不过，它也可能会掩盖需要处理的潜在问题。实施适当的监控和措施，以便检测问题的模式，包括那些被自动修复的问题，从而从根本上解决问题。

弹性系统经过精心设计，可以立即将性能下降事件传达给相应的团队。这些通知应通过一个或多个通信渠道发送。

期望的结果： 在突破阈值 [例如错误率、延迟或其他重要的关键绩效指标（KPI）] 时，系统会立即向运营团队发送警报，以便尽快解决这些问题，从而避免或最大限度地减少对用户的影响。

常见反模式：

- 发送的警报过多。
- 发送不可操作的警报。
- 将警报阈值设置得太高（过于敏感）或太低（不够敏感）。
- 不针对外部依赖项发送警报。
- 在设计监控和警报时，没有考虑 灰色故障 。
- 执行自动修复，但未通知相应的团队需要进行该修复。

建立此最佳实践的好处： 恢复通知可以让运营和业务团队了解到服务性能下降的情况，这样他们就可以立即做出反应，尽可能地缩短平均检测时间（MTTD，Mean Time To Detect）和平均修复时间（MTTR，Mean Time To Repair）。恢复事件通知还可确保您不会忽略不经常发生的问题。

未建立这种最佳实践的情况下暴露的风险等级： 中。未能实施适当的监控和事件通知机制，会导致未能检测到出现的问题模式，包括那些被自动修复的问题。只有当用户联系客户服务或者在偶然的情况下，团队才会了解到系统性能下降的情况。

实施指导

在定义监控策略时，触发的警报是常见事件。此事件可能包含警报的标识符、警报状态（例如 IN ALARM 或 OK），以及触发该警报的对象的详细信息。在许多情况下，系统应该检测到警报事件并发送电子邮件通知。这是对警报执行操作的示例。警报通知对于可观测性至关重要，因为它会告知相关人员所存在的问题。不过，当您的可观测性解决方案能够针对事件采取合理的操作时，它可以自动修复问题，而无需人工干预。

建立 KPI 监控警报后，在超过阈值时，应向相应的团队发送警报。这些警报还可用于触发自动流程，尝试修复性能下降问题。

对于更复杂的阈值监控，应考虑使用复合警报。复合警报使用多个 KPI 监控警报，根据运营业务逻辑创建警报。CloudWatch 警报可被配置为发送电子邮件，或使用 Amazon SNS 集成或 Amazon EventBridge 将事件记录到第三方事件跟踪系统。

实施步骤

根据监控工作负载的方式，创建各种类型的警报，例如：

- 使用应用程序警报，检测工作负载的任何部分未正常工作的情况。
- 基础设施警报 指明何时扩展资源。警报可以直观地显示在控制面板上，通过 Amazon SNS 或电子邮件发送，并与 Auto Scaling 结合使用来扩展或缩减工作负载资源。
- 您可以创建简单的 静态警报 ，用于监控在指定数量的评估周期内，指标突破静态阈值的情况。
- 复合警报 可以处理来自多个来源的复杂警报。
- 创建警报后，请创建相应的通知事件。您可以直接调用 Amazon SNS API 来发送通知，并关联任何自动化的修复或通信机制。
- 集成 Amazon Health Aware 监控功能，以便监控可能出现性能下降的 AWS 资源。对于关键业务工作负载，此解决方案可提供对 AWS 服务的主动实时警报的访问。

资源

相关的 Well-Architected 最佳实践：

- 可用性定义

相关文档：

- 基于静态阈值创建 CloudWatch 警报

- 什么是 Amazon EventBridge？
- 什么是 Amazon Simple Notification Service？
- 发布自定义指标
- 使用 Amazon CloudWatch 警报
- Amazon Health Aware（AHA）
- 设置 CloudWatch 复合警报
- What's new in AWS Observability at re:Invent 2022

相关工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP07 构造您的产品以满足可用性目标和正常运行时间服务等级协议（SLA）

构造您的产品以满足可用性目标和正常运行时间服务等级协议（SLA）。如果您公开或私下同意可用性目标或正常运行时间 SLA，请确认您设计了架构和运营流程来支持它们。

期望结果：每个应用程序的性能指标都有明确的可用性和 SLA 目标，可以监控和维护目标，以便符合业务成果。

常见反模式：

- 在不设置任何 SLA 的情况下设计和部署工作负载。
- 在没有理由或业务需求的情况下将 SLA 指标设置得很高。
- 在设置 SLA 时不考虑依赖项及其基础 SLA。
- 创建应用程序设计时不考虑弹性的责任共担模式。

建立此最佳实践的好处：根据关键弹性目标设计应用程序有助于实现业务目标和满足客户期望。这些目标有助于推动评估不同技术并考虑各种权衡的应用程序设计过程。

未建立此最佳实践暴露的风险等级： 中

实施指导

应用程序设计必须考虑因业务、运营和财务目标而产生的各种要求。根据运营要求，工作负载需要具有特定的弹性指标目标，以便可以适当地监控和支持它们。不得在部署工作负载之后才设置或引出弹性指标。而应该在设计阶段定义这些指标，并帮助指导作出各种决策和权衡。

- 每个工作负载都应有自己的一组弹性指标。这些指标可能与其他业务应用程序的指标不同。
- 减少依赖项会对可用性产生积极影响。每个工作负载都应考虑其依赖项及其 SLA。一般来说，选择可用性目标等于或大于工作负载目标的依赖项。
- 在可能的情况下，考虑采用松散耦合设计，以便您的工作负载可以在依赖项受损的情况下正常运行。
- 减少控制面板依赖项，特别是在恢复或降级期间。评估对于任务关键型工作负载保持静态稳定的设计。使用资源节约来提高工作负载中这些依赖项的可用性。
- 可观测性和仪表化对于通过减少平均检测时间（MTTD）和平均修复时间（MTTR）来实现 SLA 至关重要。
- 更低的故障频率（更长的 MTBF）、更短的故障检测时间（更短的 MTTD）和更短的修复时间（更短的 MTTR）是用于提高分布式系统中的可用性的三个因素。
- 建立和满足工作负载的弹性指标，这是所有有效设计的基础。这些设计必须考虑设计复杂性、服务依赖项、性能、扩展和成本之间的权衡。

实施步骤

- 检查并记录工作负载设计，考虑以下问题：
  - 在工作负载中的什么地方使用控制面板？
  - 工作负载如何实施容错？
  - 扩缩、弹性伸缩、冗余和高可用性组件的设计模式是什么？
  - 数据一致性和可用性的要求是什么？
  - 是否考虑了资源节约或资源静态稳定性？
  - 服务依赖项是什么？
- 与利益攸关方合作时，根据工作负载架构定义 SLA 指标。考虑工作负载使用的所有依赖项的 SLA。
- 设置 SLA 目标后，优化架构以满足 SLA。
- 设置满足 SLA 的设计后，实施运营更改、流程自动化和运行手册，这些操作也将侧重于减少 MTTD 和 MTTR。
- 部署之后，监控和报告 SLA。

资源

相关最佳实践：

- REL03-BP01 选择如何划分工作负载
- REL10-BP01 将工作负载部署到多个位置
- REL11-BP01 监控工作负载的所有组件以检测故障
- REL11-BP03 自动修复所有层
- REL12-BP05 使用混沌工程测试弹性
- REL13-BP01 定义停机和数据丢失的恢复目标
- 了解工作负载运行状况

相关文档：

- 通过冗余实现可用性
- 可靠性支柱 - 可用性
- 衡量可用性
- AWS 故障隔离界限
- 弹性的责任共担模式
- 使用可用区的静态稳定性
- AWS 服务等级协议（SLA）
- AWS 上基于 cell 的架构指南
- AWS 基础设施
- 高级多可用区弹性模式白皮书

相关服务：

- Amazon CloudWatch
- AWS Config
- AWS Trusted Advisor

测试可靠性

在为您的工作负载采用弹性设计以应对生产压力以后，测试是确保其按设计预期运行，并且提供您所预期弹性的唯一方式。

通过测试验证您的工作负载满足功能及非功能要求，因为错误或性能瓶颈可能会影响您的工作负载的可靠性。测试工作负载的弹性，以帮助您发现只会在生产环境中出现的潜在错误。定期执行这些测试。

最佳实践

- REL12-BP01 使用行动手册调查故障
- REL12-BP02 执行事后分析
- REL12-BP03 测试功能要求
- REL12-BP04 测试扩展和性能要求
- REL12-BP05 使用混沌工程测试弹性
- REL12-BP06 定期进行实际试用

REL12-BP01 使用行动手册调查故障

通过在行动手册中记录调查流程，实现对并不十分了解的故障场景做出一致且及时的响应。行动手册是在确定哪些因素导致故障场景时要执行的预定义步骤。所有流程步骤的结果都将用于确定要采取的后续步骤，直到问题得到确定或上报。

行动手册是您必须要执行的主动计划，以便有效采取响应措施。当在生产中遇到行动手册未涉及的故障场景时，首先要解决问题（灭火）。然后回过头来思考您在解决问题时采取的措施，并将这些措施作为新条目添加到行动手册中。

请注意，行动手册可用于对特定事件做出响应，运行手册则用来达成特定的结果。通常，运行手册适用于例行活动，而行动手册则被用于对非例行事件做出响应。

常见反模式：

- 计划在以下情况下部署工作负载：不清楚诊断问题或响应意外事件的流程。
- 关于在对事件进行调查时从哪些系统收集日志和指标的计划外的决定。
- 指标和事件保留的时间不够长，无法检索到数据。

建立此最佳实践的好处： 使用行动手册可确保始终如一地遵循程序。编写行动手册可以减少手动操作导致的错误。通过实现行动手册自动化，可以消除团队成员干预的需要，或者在他们开始干预时便向他们提供更多信息，从而缩短事件响应时间。

未建立此最佳实践暴露的风险等级： 高

实施指导

- 使用行动手册来发现问题。管理手册是用于调查问题的书面程序。在行动手册中记录流程，实现对故障场景的一致而及时的响应。行动手册必须包含所需的信息和指导，让足够熟练的员工能够收集适用信息、确定故障的潜在来源、隔离故障，并确定成因（在意外事件发生后执行分析）。
  - 以代码形式实施行动手册。为行动手册编写脚本，以代码形式执行运营，以确保一致性并减少由手动流程引起的错误。行动手册可以由代表不同步骤的多个脚本组成，这些步骤可能是确定问题成因所必需的。系统可能会在运行手册活动过程中触发或执行行动手册活动，也可能针对响应发现的事件而提示执行行动手册活动。
    - 使用 AWS Systems Manager 自动执行您的运营手册
    - AWS Systems Manager Run Command
    - AWS Systems Manager Automation
    - 什么是 AWS Lambda？
    - 什么是 Amazon EventBridge？
    - 使用 Amazon CloudWatch 告警

资源

相关文档：

- AWS Systems Manager Automation
- AWS Systems Manager Run Command
- 使用 AWS Systems Manager 自动执行您的运营手册
- 使用 Amazon CloudWatch 告警
- 使用金丝雀（Amazon CloudWatch Synthetics）
- 什么是 Amazon EventBridge？
- 什么是 AWS Lambda？

相关示例：

- 使用行动手册和运行手册自动完成操作

REL12-BP02 执行事后分析

审核影响客户的事件，确定这些事件的成因和预防措施。利用这些信息来制定缓解措施，以限制或防止再次发生同类事件。制定程序以迅速有效地做出响应。根据目标受众，适当传达事件成因和纠正措施。如果需要，可将这些原因告知他人。

评估为什么现有测试找不到问题。如果还没有，增设测试。

期望结果：您的团队采用商定的一致方法来进行事后分析。一种机制是错误更正（COE）流程。COE 流程有助于您的团队识别、理解和解决事件的根本原因，同时还可以建立防护机制，以限制同一事件再次发生的可能性。

常见反面模式：

- 查找事件成因，但不继续深入探究其他潜在问题和缓解问题的方法。
- 只找出人为错误原因，但不提供任何培训或可防止人为错误的自动化功能。
- 只注重追究责任，而不去了解根本原因，营造恐惧文化，阻碍开诚布公的交流
- 见解分享不畅，事件分析结果仅限于一小群人知道，其他人无法从中吸取经验教训
- 没有收集制度性知识的机制，因此无法以最新最佳实践的形式保存经验教训，从而失去宝贵见解，导致根源相同或相似的事件反复发生

建立此最佳实践的好处：执行事后分析并且分享分析结果，可以减轻其他实施了相同故障因素的工作负载发生故障的风险，并且让它们能够在事件发生前就实施缓解措施或自动恢复。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

有效的事后分析让您有机会针对系统中其他地方使用的架构模式存在的问题提出常见的解决方案。

COE 流程的基础是记录和解决问题。建议定义一种标准化的方法来记录关键的根本原因，并确保其得到分析和处理。为事后分析流程分配明确的责任人。指派负责的团队或个人来监督事件调查和后续行动。

鼓励注重学习和改进而不是互相推脱责任的文化。强调目标是防止将来发生事件，而不是惩罚某些人。

为执行事后分析制定明确定义的程序。这些程序应概述要采取的步骤、要收集的信息以及在分析期间要解决的关键问题。彻底调查事件，除直接原因外，还要找出根本原因和影响因素。使用诸如“五个为什么”之类的技巧来深入研究潜在问题。

维护从事件分析中吸取的经验教训的存储库。这些制度性知识可以作为未来的事件和预防工作的参考。分享在事后分析中发现的结果和洞察，并考虑召开公开的事后总结会议，讨论经验教训。

实施步骤

- 在执行事后分析时，确保整个过程不是以追究责任为目的。这使事件中涉及到的人员能够冷静地看待建议的纠正措施，并促进诚实的自我评测和团队间的协作。
- 定义记录关键问题的标准化方法。此类文档的示例结构如下所示：
  - 发生了什么？
  - 客户和您的业务受到了什么影响？
  - 根本原因是什么？
  - 您有哪些数据来支持这一点？
    - 例如，指标和图表
  - 对关键支柱有什么影响，特别是在安全方面？
    - 在构造工作负载时，您需要基于您的业务环境在各个支柱之间做出权衡。这些业务决策可以确定设计优先事项。在开发环境中，您可能会通过降低可靠性来降低成本；而对于任务关键型解决方案，您可能会通过增加成本来提高可靠性。安全始终是头等大事，因为您必须保护您的客户。
  - 您获得了哪些经验教训？
  - 您采取了哪些纠正措施？
    - 行动项
    - 相关事项
- 为执行事后分析制定明确定义的标准操作程序。
- 设置标准化事件报告流程。全面记录所有事件，包括最初的事件报告、日志、通信和事件期间采取的行动。
- 请记住，并不是发生了停机才叫做事件。这可能是未遂事件，也可能是系统能够履行其业务功能，但以意外的方式运行。
- 根据反馈和经验教训，持续改进事后分析流程。
- 在知识管理系统中记录关键调查发现，并考虑应添加到开发人员指南或部署前清单中的任何模式。

资源

相关文档：

- 为什么您应该制定错误更正（COE）措施

相关视频：

- 亚马逊成功应对失败的方法
- AWS re:Invent 2021 - Amazon Builders' Library：亚马逊的卓越运营

REL12-BP03 测试功能要求

使用的技术包括用于验证所需功能的单元测试和集成测试。

如果这些测试作为构建和部署措施的一部分自动运行，则您可以获得最佳的结果。例如，使用 AWS CodePipeline，开发人员会在 CodePipeline 自动检测到变更时提交对源存储库的更改。执行更改，然后加以测试。在测试完成以后，构建的代码会被部署到用于测试的暂存服务器。CodePipeline 会从暂存服务器运行更多测试，如集成或负载测试等。在成功完成此类测试以后，CodePipeline 会将经过测试并获得批准的代码部署到生产实例。

此外，过去的经验告诉我们，可运行合成事务测试（又被称作 金丝雀测试，但不要和金丝雀部署相混淆）模拟用户行为，这是最重要的测试流程之一。从不同的远程位置针对您的工作负载端点持续地运行此类测试。Amazon CloudWatch Synthetics 使您能够 创建 Canary 以便监控您的终端节点和 API。

未建立此最佳实践暴露的风险等级： 高

实施指导

- 测试功能要求。这包括用于验证所需功能的单元测试和集成测试。
  - 将 CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行构建
  - AWS CodePipeline 增加了对通过 AWS CodeBuild 进行单位和自定义集成测试的支持
  - 持续交付和持续集成
  - 使用金丝雀（Amazon CloudWatch Synthetics）
  - 软件测试自动化

资源

相关文档：

- AWS 合作伙伴：可帮助实施持续集成管道的合作伙伴
- AWS CodePipeline 增加了对通过 AWS CodeBuild 进行单位和自定义集成测试的支持
- AWS Marketplace：可用于实现持续集成的产品
- 持续交付和持续集成
- 软件测试自动化
- 将 CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行构建
- 使用金丝雀（Amazon CloudWatch Synthetics）

REL12-BP04 测试扩展和性能要求

使用的技术包括负载测试以验证工作负载是否满足扩展和性能要求。

在云中，您可以按需为您的工作负载创建生产规模环境。如果在缩减的基础设施上运行这些测试，您必须根据您认为在生产中将会发生的情况扩展您观察到的结果。如果不想影响实际用户，您可以在生产中开展负载和性能测试，并且对您的测试数据进行标记，以避免它与真实的用户数据、损坏的使用情况统计或生产报告混在一起。

通过测试确保您的基础资源、扩展设置、服务限额和弹性设计能够在负载之下如预期运行。

未建立这种最佳实践的情况下暴露的风险等级： 高

实施指导

- 测试扩展和性能要求。执行负载测试以验证工作负载是否满足扩展和性能要求。
  - AWS 上的分布式负载测试：模拟数千个连接的用户
  - Apache JMeter
    - 将您的应用程序部署在与生产环境相同的环境中，然后执行负载测试。
      - 使用基础设施即代码概念，以创建尽可能类似于生产环境的环境。

资源

相关文档：

- AWS 上的分布式负载测试：模拟数千个连接的用户
- Apache JMeter

REL12-BP05 使用混沌工程测试弹性

在处于或尽可能接近生产的环境中定期运行混沌试验，以了解系统如何应对不利条件。

期望结果：

除了在事件期间验证已知预期工作负载行为的弹性测试之外，还可以通过以故障注入实验或注入意外负载的形式应用混沌工程，定期验证工作负载的弹性。将混沌工程和弹性测试结合起来，您可以提升信心，相信工作负载能够经受组件故障，并可从意外中断中恢复，而影响极小甚至没有影响。

常见反模式：

- 进行弹性设计，但不验证故障发生时工作负载如何作为一个整体运行。
- 从不在真实环境和预期负载下进行试验。
- 不将实验视为代码，也不在整个开发周期中维护它们。
- 不将混沌实验作为 CI/CD 管道的一部分，也不在部署之外运行。
- 在确定要对哪些故障进行试验时，没有想到使用过去的意外事件后分析。

建立此最佳实践的好处： 注入故障来验证工作负载的弹性，这可以让您提升信心，相信您的弹性设计的恢复程序将在真正发生故障的情况下能够发挥作用。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

利用混沌工程，您的团队能够在服务提供商、基础设施、工作负载和组件级别，以可控的方式不断注入真实世界的干扰（模拟），而对客户的影响极小甚至没有影响。它使您的团队能够从故障中学习，观察、测量和提高工作负载的弹性，并验证在发生事件时，系统会发出警报并通知团队。

当持续执行时，混沌工程会突出工作负载中的缺陷，这些缺陷若不加以解决，可能会对可用性和运营产生负面影响。

> **Note**
>
> 混沌工程是对系统进行试验以让人们确信系统能够在生产中经受住混乱情形的规范。– 混沌工程的原则

如果系统能够经受住这些干扰，那么应将混沌实验作为自动回归测试来加以维护。这样一来，应将混沌实验作为系统开发生命周期（SDLC）的一部分，以及作为 CI/CD 管道的一部分来执行。

为了确保您的工作负载能够承受组件故障，请在实验中注入实际事件。例如，对 Amazon EC2 实例的丢失或主 Amazon RDS 数据库实例的失效转移进行试验，并验证您的工作负载没有受到影响（或影响极小）。使用组件故障的组合来模拟可能因可用区中断而引起的事件。

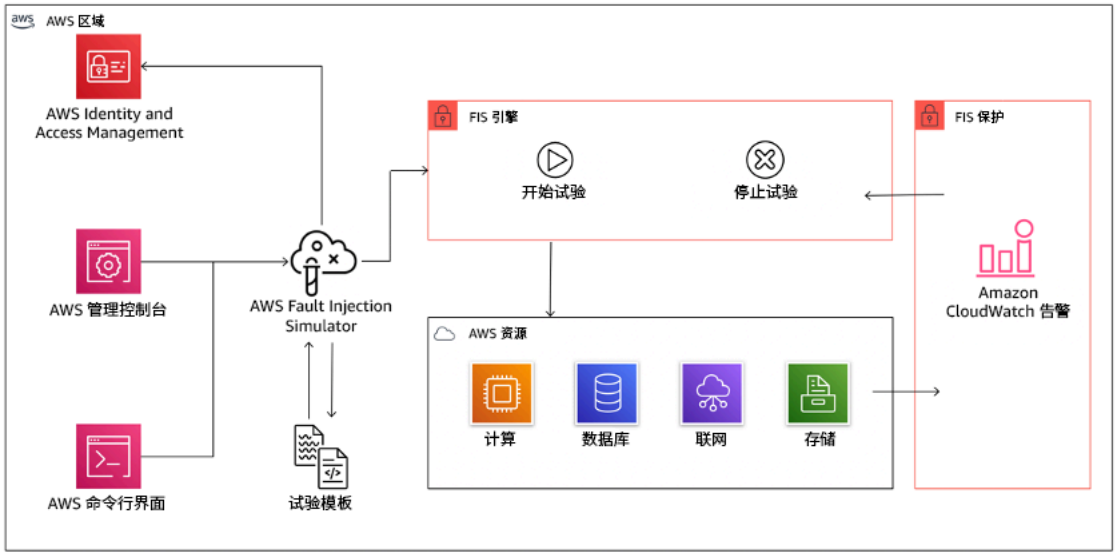
对于应用程序级故障（如崩溃），您可以从内存和 CPU 耗尽等压力源开始。

为了验证因间歇性网络中断而引发的外部依赖项的 回退或失效转移机制 ，您的组件应通过在指定时间段（从几秒到几小时不等）内阻止对第三方提供商的访问来模拟此类事件。

其他降级模式可能会影响功能的使用并降低响应速度，这通常会导致服务中断。性能下降的常见原因是，关键服务的延迟增加以及网络通信不可靠（丢包）。对于这些故障（包括延迟、丢弃的消息和 DNS 故障等网络效应）的实验可能包括无法解析名称、无法访问 DNS 服务或无法建立与依赖服务的连接。

混沌工程工具：

AWS Fault Injection Service（AWS FIS）是一项完全托管式服务，用于运行故障注入实验，而这些实验可用作 CD 管道的一部分，或在管道之外使用。AWS FIS 是在混沌工程实际试用期间使用的一个不错选择。它支持在不同类型的资源中同时引入故障，包括 Amazon EC2、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service（Amazon EKS）和 Amazon RDS 等资源。这些故障包括终止资源、强制失效转移、对 CPU 或内存施加压力、节流、延迟和数据包丢失。由于它与 Amazon CloudWatch 警报集成，因此您可以设置停止条件作为防护机制，以在实验导致意外影响时回滚。



AWS Fault Injection Service 与 AWS 资源集成，使您能够为您的工作负载运行故障注入实验。

故障注入实验也有多种第三方选项。其中包括开源工具，例如 Chaos ToolKit、Chaos Mesh和 Litmus Chaos以及商业选项，如 Gremlin。为了扩大可在 AWS 上注入的故障范围，AWS FIS 与 Chaos Mesh 和 Litmus Chaos 集成，使您能够在多个工具之间协调故障注入工作流。例如，您可以使用 Chaos Mesh 或 Litmus 故障对容器组（pod）的 CPU 运行压力测试，同时使用 AWS FIS 故障操作终止随机选择的集群节点百分比。

实施步骤

- 确定哪些故障要用于实验。

  评估工作负载的设计是否具有弹性。这种设计（使用 Well-Architected Framework的最佳实践创建）考虑到了基于关键依赖关系、过去的事件、已知问题和合规性要求的风险。列出每个旨在保持弹性的设计元素及其旨在缓解的故障。有关创建此类列表的更多信息，请参阅 《运营准备就绪情况审核》白皮书 ，该白皮书指导您如何创建流程来防止以前的事件再次发生。故障模式与影响分析（FMEA）流程为您提供了一个框架，用于对故障及其对工作负载的影响执行组件级分析。Adrian Cockcroft 在 《Failure Modes and Continuous Resilience》中更详细地概述了 FMEA。

- 为每个故障指定一个优先级。

  先进行粗略的分类，如高、中或低。要评估优先级，请考虑故障的频率和故障对整体工作负载的影响。

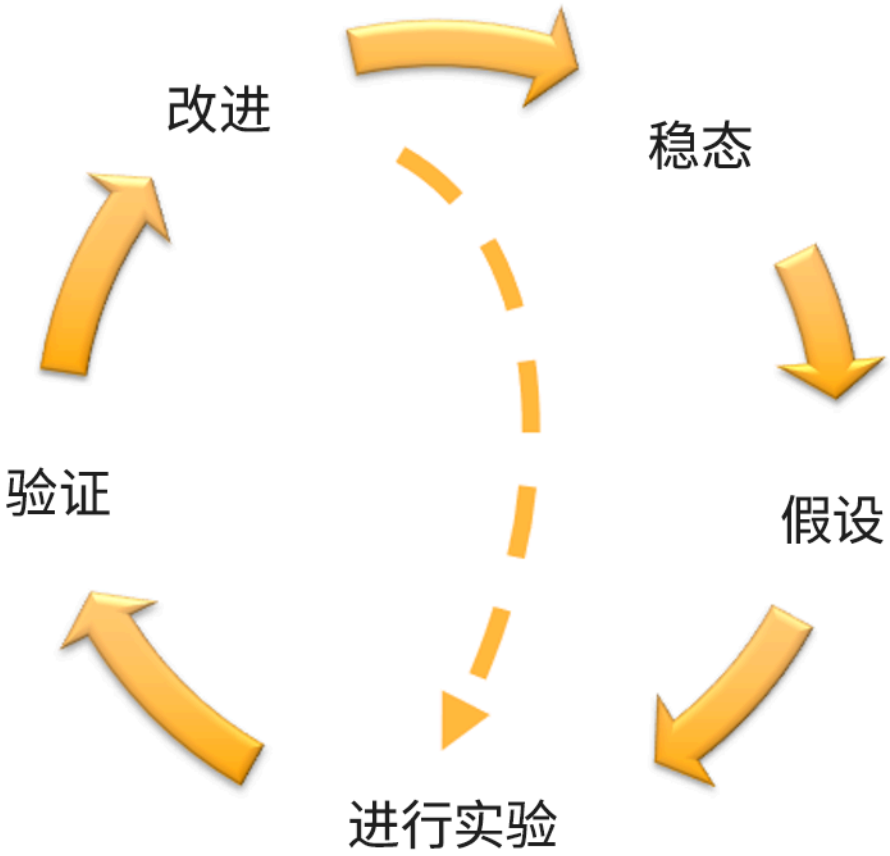
  考虑给定故障的频率时，请分析此工作负载的以往数据（如有）。如果没有以往数据，则使用在类似环境中运行的其他工作负载的数据。

考虑给定故障的影响时，故障的范围越大，一般来说影响也越大。还要考虑工作负载设计和目的。例如，访问源数据存储的能力对于进行数据转换和分析的工作负载至关重要。在这种情况下，您将确定关于访问故障以及节流访问和延迟插入的实验的优先级。

意外事件后分析是了解故障模式的频率和影响的良好数据来源。

使用指定的优先级来确定首先用哪些故障进行实验，以及开发新的故障注入实验的顺序。

- 对于您执行的每项实验，请遵循混沌工程和连续弹性飞轮。



Adrian Hornsby 采用科学方法制作的混沌工程和连续弹性飞轮。

- 将稳态定义为指示正常行为的工作负载的一些可测量输出。

  如果工作负载运行可靠且符合预期，则显示为稳态。因此，定义稳态之前，请验证您的工作负载正常运行。稳态并不一定意味着故障发生时对工作负载没有影响，因为一定百分比的故障可能在可接受的范围内。稳态是您在实验期间将观察到的基线，如果您在下一步中定义的假设结果不符合预期，它将突出显示异常。

例如，可以将某个支付系统的稳态定义为处理 300TPS，成功率为 99%，且往返时间为 500ms。

- 形成一个关于工作负载如何应对故障的假设。

  一个好的假设是基于工作负载预计如何缓解故障以保持稳态。该假设指出，如果发生特定类型的故障，系统或工作负载将继续保持稳态，因为该工作负载在设计时就有特定的缓解措施。应在假设中具体说明特定的故障类型和缓解措施。

  假设可以使用以下模板（但其他措辞也可以接受）：

  > **Note**
  >
  > 如果发生 #### ，则 ###### 工作负载将 ######## 维持 #########。

  例如：

  - 如果 Amazon EKS 节点组中 20% 的节点出现故障，则 Transaction Create API 将在不到 100ms 的时间内继续处理 99% 的请求（稳态）。Amazon EKS 节点将在五分钟内恢复，容器组（pod）将在实验开始后八分钟内得到调度并处理流量。警报将在三分钟内将发出。
  - 如果发生单个 Amazon EC2 实例故障，订单系统的 Elastic Load Balancing 运行状况检查将导致 Elastic Load Balancing 仅向剩余的运行状况良好的实例发送请求，而 Amazon EC2 Auto Scaling 将替换故障实例，从而保持服务器端（5xx）错误增长率低于 0.01%（稳态）。
  - 如果主 Amazon RDS 数据库实例发生故障，则供应链数据收集工作负载将失效转移并连接到备用 Amazon RDS 数据库实例，以保持不到 1 分钟的数据库读写错误（稳态）。

- 通过注入故障来进行实验。

  默认情况下，实验应具有故障保护机制，可承受工作负载。如果您知道工作负载将发生故障，则不要进行实验。混沌工程应该用于寻找已知的不确定因素或未知的不确定因素。已知的不确定因素是您知道但不完全理解的东西，而 未知的不确定因素 是您既不知道也不完全理解的东西。对您知道已经发生故障的工作负载进行试验不会为您提供新的见解。您应该对实验仔细规划，明确一个影响范围，并提供一种可在出现意外动荡时应用的回滚机制。如果尽职调查表明您的工作负载应该能经受住实验，那就继续这项实验。有几种注入故障的选项。对于 AWS 上的工作负载，AWS FIS 提供了许多称为 操作的预定义故障模拟。您还可以定义在 AWS FIS 中运行的自定义操作（使用 AWS Systems Manager 文档）。

  我们不鼓励使用自定义脚本进行混沌实验，除非这些脚本能够了解工作负载的当前状态，能够发出日志，并在可能的情况下提供回滚和停止条件的机制。

支持混沌工程的有效框架或工具集应跟踪实验的当前状态，发出日志，并提供回滚机制以支持实验的受控执行。从 AWS FIS 这样的成熟服务开始，该服务支持您在明确定义的范围内和安全机制下进行实验，如果实验引入了意外的动荡，则可以回滚实验。要了解更多使用 AWS FIS 的实验，另请参阅 “通过混沌工程构建弹性且架构完善的应用程序”实验室。此外， AWS Resilience Hub 将分析您的工作负载，并创建您可以选择在 AWS FIS 中实施和运行的实验。

> **Note**
>
> 对于每项实验，要清楚地了解其范围及影响。我们建议首先在非生产环境中模拟故障，然后再在生产环境中运行。

应使用实际负载，通过 金丝雀部署 在生产环境中进行实验，尽可能同时启动控制和实验系统部署。在非高峰时间进行实验是一种很好的做法，可以减小首次在生产环境中试验时的潜在影响。此外，如果使用实际的客户流量会带来太大的风险，您可以在生产基础设施上针对控制和实验部署使用合成流量进行实验。当不能使用生产环境时，在尽可能接近生产环境的预生产环境中进行实验。

您必须建立和监控防护机制，确保实验对生产流量或其他系统的影响不会超过可接受的限度。建立停止条件，以便在实验达到您定义的防护机制指标的阈值时停止实验。这应该包括工作负载的稳态指标，以及针对您要注入故障的组件的指标。A 合成监控器 （也称为用户金丝雀）是一个通常应作为用户代理包含的指标。AWS FIS 的停止条件 应纳入实验模板中，每个模板最多可以有五个停止条件。

混沌的原则之一是尽量缩小实验范围并减小其影响：

虽然必须考虑到一些短期负面影响，但混沌工程师有责任和义务确保实验产生的影响极小且可控。

验证范围和潜在影响的一种方法是首先在非生产环境中进行实验（验证停止条件的阈值在实验期间按预期激活，并且可观测性到位以捕获异常），而不是直接在生产环境中进行实验。

运行故障注入实验时，确保所有责任方均知情。与适当的团队（如运营团队、服务可靠性团队和客户支持团队）沟通，让他们知道实验将在何时运行以及预期会发生什么。为这些团队提供沟通工具，以便在他们看到任何不利影响时通知进行实验的人员。

必须将工作负载及其底层系统恢复到最初的已知良好状态。通常，工作负载的弹性设计会自我修复。但一些故障设计或失败的实验可能会使您的工作负载处于意外的失败状态。在实验结束时，您必须意识到这一点，并恢复工作负载和系统。使用 AWS FIS，您可以在操作参数中设置回滚配置

（也称为后期操作）。后期操作将目标返回到运行该操作之前的状态。无论是自动执行（如使用 AWS FIS）还是手动执行，这些后期操作都应包含在描述如何检测和处理故障的行动手册中。

- 验证假设。

  混沌工程的原则 为如何验证工作负载的稳态提供了以下指导：

  关注系统的可测量输出，而不是系统的内部属性。短时间内对该输出的测量构成了系统稳态的代理。整个系统的吞吐量、错误率和延迟百分比都可以是代表稳态行为的相关指标。通过关注实验过程中的系统行为模式，混沌工程验证系统确实在工作，而不是试图验证它如何工作。

  在之前的两个示例中，我们包括了服务器端（5xx）错误增长率低于 0.01% 和数据库读写错误持续时间不到 1 分钟的稳态指标。

  5xx 错误是一个很好的指标，因为它们是工作负载客户端将直接经历的故障模式的结果。数据库错误测量适合作为故障的直接结果，但是还应补充一个客户端影响测量，例如失败的客户请求或向客户端显示的错误。此外，在工作负载客户端直接访问的任何 API 或 URI 上包括一个合成监控器（也称为用户金丝雀）。

- 改进工作负载设计，以提高弹性。

  如果未保持稳态，则调查如何改进工作负载设计以缓解故障，应用 AWS Well-Architected 可靠性支柱的最佳实践。可以在 AWS Builder's Library中找到其他指导和资源，其中包含有关如何 改进运行状况检查 或 在应用程序代码中结合采用重试与回退的文章，等等。

  实施这些更改后，再次进行实验（如混沌工程飞轮中的虚线所示），以确定其有效性。如果验证步骤表明假设成立，那么工作负载将处于稳态，循环将继续。

- 定期进行实验。

  混沌实验是一个循环，作为混沌工程的一部分，应定期进行实验。在工作负载满足实验的假设后，实验应实现自动化，作为 CI/CD 管道的回归部分持续运行。要了解如何做到这一点，请参阅关于 如何使用 AWS CodePipeline 进行 AWS FIS 实验的博客。这个关于反复 在 CI/CD 管道中进行 AWS FIS 实验 的实验室使您能够动手实践。

  故障注入实验也是实际试用的一部分（请参阅 REL12-BP06 定期进行实际试用）。实际试用会模拟故障或事件，以便验证系统、流程和团队的响应。其目的是实际执行团队在发生意外事件时会执行的操作。

- 捕获和存储实验结果。

  必须捕获并持久保存故障注入实验的结果。包括所有必要的数据（如时间、工作负载和条件），以便以后能够分析实验结果和趋势。结果示例可能包括控制面板的屏幕截图、从指标数据库进行的 CSV

转储，或实验中事件和观察结果的手写记录。使用 AWS FIS 进行实验记录 可作为这种数据捕获的一部分。

资源

相关最佳实践：

- REL08-BP03 将韧性测试作为部署的一部分进行集成
- REL13-BP03 测试灾难恢复实施以验证实施效果

相关文档：

- 什么是 AWS Fault Injection Service？
- 什么是 AWS Resilience Hub？
- 混沌工程的原则
- 混沌工程：规划您的首次实验
- 弹性工程：学会接受故障
- 混沌工程案例
- 避免在分布式系统中回退
- 用于混沌实验的金丝雀部署

相关视频：

- AWS re:Invent 2020：使用混沌工程测试弹性（ARC316）
- AWS re:Invent 2019：通过混沌工程提高弹性（DOP309-R1）
- AWS re:Invent 2019：在无服务器世界中执行混沌工程（CMY301）

相关示例：

- Well-Architected 实验室：第 300 级：测试 Amazon EC2、Amazon RDS 和 Amazon S3 的弹性
- “混沌工程在 AWS 上的应用”实验室
- “通过混沌工程构建弹性且架构完善的应用程序”实验室
- “无服务器混沌”实验室

- “使用 AWS Resilience Hub 测量和提高应用程序弹性”实验室

相关工具：

- AWS Fault Injection Service
- AWS Marketplace： Gremlin 混沌工程平台
- Chaos ToolKit
- Chaos Mesh
- Litmus

REL12-BP06 定期进行实际试用

利用实际试用活动，在尽可能接近生产环境的环境中（包括在生产环境中），与将参与实际故障情景的人员一起为应对事件和故障而练习如何使用您的程序。实际试用会强制执行相关措施，以确保生产事件不会影响用户。

实际试用会模拟故障或事件，以便测试系统、流程和团队的响应。其目的是实际执行团队在发生意外事件时会执行的操作。这将帮助您了解可以从哪些方面作出改进，并有助于培养组织处理各种事件的经验。这些操作应该定期进行，让团队建立起关于响应方式的 肌肉记忆 。

在非生产环境中对您的弹性设计进行测试以后，可通过 Game Day 确保生产中的一切按照计划运行。Game Day，尤其如果是首次开展，是所有人员都应该参加的活动，工程师和运营团队都会得到关于开展时间以及活动内容的信息。运行手册准备就绪。以规定的方式在生产系统中执行模拟事件（包括可能出现的故障事件），并评估影响。如果所有系统如设计运行，检测和自我修复不会产生或只会产生非常轻微的影响。但如果观察到负面影响，测试将会回滚，并且（使用运行手册）修复问题，在必要时手动修复。由于实际试用经常在生产中进行，所以应采取全部预防措施，以确保不会对客户造成可用性影响。

常见反模式：

- 记录您的程序，但不要执行。
- 不要让业务决策者参与测试练习。

建立此最佳实践的好处： 定期执行实际试用可确保在发生实际事件时，所有员工都遵守策略和程序，并且能够验证这些策略和程序是否合适。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

- 安排实际试用，以定期运用运行手册和行动手册。生产事件中涉及的所有人员均需参与实际试用：业务负责人、开发人员、运营人员和事件响应团队。
  - 运行负载或性能测试，然后运行故障注入。
  - 寻找运行手册中的异常，并利用这些异常机会练习使用行动手册。
    - 如果您违反运行手册，请完善运行手册或纠正相应行为。如果练习使用行动手册，请确定应使用的运行手册，或者创建一个新运行手册。

资源

相关文档：

- 什么是 AWS 实际试用？

相关视频：

- AWS re:Invent 2019：通过混沌工程提高弹性（DOP309-R1）

相关示例：

- AWS Well-Architected 实验室 – 测试弹性

灾难恢复 (DR) 计划

拥有适当的备份和冗余工作负载组件是您的 DR 策略的开始。RTO 和 RPO 是您恢复工作负载的目标。根据业务需求设置这些目标。通过实施策略来实现这些目标，同时考虑工作负载资源和数据的位置和功能。中断概率和恢复成本也是关键因素，有助于了解为工作负载提供灾难恢复的商业价值。

可用性和灾难恢复都依赖于相同的最佳实践，例如监控故障、部署到多个位置和自动失效转移。然而，可用性侧重于工作负载的组件，而灾难恢复侧重于整个工作负载相互独立的副本。灾难恢复的目标与可用性不同，主要关注发生灾难后进行恢复所需的时间。

最佳实践

- REL13-BP01 定义停机和数据丢失的恢复目标
- REL13-BP02 使用定义的恢复策略来实现恢复目标

- REL13-BP03 测试灾难恢复实施以验证实施效果
- REL13-BP04 管理 DR 站点或区域的配置偏差
- REL13-BP05 自动执行恢复

REL13-BP01 定义停机和数据丢失的恢复目标

工作负载具有恢复时间目标（RTO）和恢复点目标（RPO）。

恢复时间目标 (RTO) 是指服务中断和服务恢复之间的最大可接受延迟。这可以确定在服务不可用时被视为可接受的时间窗口。

恢复点目标 (RPO) 是指自上一个数据恢复点以来的最大可接受时间。这可以确定在上一个恢复点和服务中断之间可接受的数据丢失程度。

在为您的工作负载选择合适的灾难恢复（DR，Disaster Recovery）策略时，RTO 和 RPO 值是重要的考虑因素。这些目标由业务部门确定，然后由技术团队用来选择和实施 DR 策略。

期望结果：

每个工作负载都有一个根据业务影响定义的指定 RTO 和 RPO。工作负载被分配到一个预定义的层，该层定义服务可用性和可接受的数据丢失，以及关联的 RTO 和 RPO。如果无法进行这样的分层，那么可以为每个工作负载分配定制的分层，用于以后创建层。在为工作负载选择灾难恢复策略实施时，使用 RTO 和 RPO 作为主要考虑因素之一。在选择 DR 策略时还要考虑成本约束、工作负载依赖关系和运维需求。

对于 RTO，了解基于中断持续时间的影响。是线性的还是非线性的影响？（例如，四小时后，您关闭一条生产线，直到下一班开始）。

如下所示的灾难恢复矩阵可以帮助您了解工作负载的重要性与恢复目标之间的关系。（请注意，X 轴和 Y 轴的实际值应根据您组织的需求进行定制）。

灾难恢复矩阵						
		恢复点目标				
		少于 1 分钟	少于 1 小时	少于 6 小时	少于 1 天	多于 1 天
恢复时间目标	少于 10 分钟	严重	严重	高	中	中
	少于 2 小时	严重	高	中	中	低
	少于 8 小时	高	中	中	低	低
	少于 24 小时	中	中	低	低	低
	多于 24 小时	中	低	低	低	低

图 16：灾难恢复矩阵

常见反模式：

- 未定义恢复目标。
- 选择任意恢复目标。
- 选择过于宽松并且不符合业务目标的恢复目标。
- 不了解停机和数据丢失的影响。
- 选择不切实际的恢复目标，如零恢复时间和零数据丢失，这对于您的工作负载配置可能无法实现。
- 选择比实际业务目标更严格的恢复目标。这将强制实施比工作负载所需的成本更高并且更复杂的 DR。
- 选择与所依赖工作负载的恢复目标不兼容的恢复目标。
- 您的恢复目标没有考虑法规合规性要求。
- 为工作负载定义了 RTO 和 RPO，但从未测试过。

建立此最佳实践的好处： 在指导您的 DR 实施时，需要您的恢复时间目标和数据丢失恢复目标。

未建立此最佳实践暴露的风险等级： 高

实施指导

对于给定的工作负载，您必须了解停机和数据丢失对业务的影响。随着停机时间或数据丢失的增加，影响通常会越来越大，但这种增长的形式可能会因工作负载类型而异。例如，您也许可以容忍长达一小时的停机时间而没有多大影响，但在一小时之后影响会迅速上升。对业务的影响表现为多种形式，包括货

币成本（如收入损失）、客户信任（以及对声誉的影响）、运维问题（如错过工资发放或生产力下降）和监管风险。使用以下步骤了解这些影响，并为您的工作负载设置 RTO 和 RPO。

实施步骤

1. 确定此工作负载的业务利益相关者，并与他们一起实施这些步骤。工作负载的恢复目标是一项业务决策。然后，技术团队与业务利益相关者合作，使用这些目标来选择 DR 策略。

   > **Note**
   >
   > 对于步骤 2 和 3，您可以使用 the section called “实施工作表”.

2. 通过回答以下问题，收集必要的信息来做出决策。
3. 在组织中，您是否对工作负载影响的重要性进行了分类或分级？
   a. 如果有，请将此工作负载分配到一个类别
   b. 如果没有，则建立这些类别。创建不超过五个类别，并细化每个类别的恢复时间目标范围。类别示例包括：关键、高、中、低。要了解工作负载如何映射到类别，请考虑工作负载是任务关键型、业务重要型还是非业务驱动型。
   c. 根据类别设置工作负载 RTO 和 RPO。始终选择比进入此步骤时计算的原始值更严格的类别（更低的 RTO 和 RPO）。如果这导致值发生了不适当的较大改变，那么考虑创建一个新类别。
4. 根据这些答案，为工作负载分配 RTO 和 RPO 值。这可以直接完成，也可以通过将工作负载分配给预定义的服务层来完成。
5. 在工作负载团队和利益相关者可访问的位置，记录此工作负载的灾难恢复计划（DRP，disaster recovery plan），此计划是组织的 业务连续性计划（BCP，Business Continuity Plan）的一部分
   a. 记录 RTO 和 RPO，以及用于确定这些值的信息。包括用于评估工作负载对业务影响的策略
   b. 除 RTO 和 RPO 之外，记录您根据灾难恢复目标正在跟踪或计划跟踪的其他指标
   c. 在进行创建时，您将 DR 策略和运行手册的详细信息添加到此计划中。
6. 通过在如图 15 所示的矩阵中查找工作负载的重要性，您可以开始建立为组织定义的预定义服务层。
7. 根据 实施 DR 策略（或 DR 策略的概念验证）之后，the section called “REL13-BP02 使用定义的恢复策略来实现恢复目标”测试此策略以确定工作负载的实际 RTC（Recovery Time Capability，恢复时间能力）和 RPC（Recovery Point Capability，恢复点能力）。如果这些能力没有达到所预期的恢复目标，那么，要么与您的业务利益相关者一起调整这些目标，要么对 DR 策略进行更改以便实现预期的目标。

主要问题

1. 在对业务产生严重影响之前，工作负载可以停止的最长时间是多少
   a. 确定在工作负载中断时，每分钟业务的货币成本（直接财务影响）。
   b. 请注意，影响并不总是线性的。影响可能在一开始是有限的，然后在超过一个关键时间点后迅速增加。
2. 在对业务造成严重影响之前，可以丢失的最大数据量是多少
   a. 对于最关键的数据存储，请考虑此值。确定其他数据存储的各自关键性。
   b. 如果工作负载数据丢失，是否可以重新创建？ 如果这在操作上比备份和还原更容易，那么根据用于重新创建工作负载数据的源数据的重要性来选择 RPO。
3. 此工作负载所依赖的工作负载（下游）或依赖于此工作负载的工作负载（上游）的恢复目标和可用性期望是什么？
   a. 选择使此工作负载能够满足上游依赖项要求的恢复目标
   b. 根据下游依赖项的恢复能力，选择可实现的恢复目标。非关键的下游依赖项（您可以“绕过”它们）可以排除。或者，处理关键的下游依赖项，在必要时提高其恢复能力。

其他问题

考虑以下问题，以及它们如何应用于此工作负载：

4. 根据中断类型（区域与可用区等），您是否有不同的 RTO 和 RPO？
5. 您的 RTO/RPO 是否会在特定时间（季节性、销售活动、产品发布）发生变化？ 如果是这样，不同的测量和时间边界是什么？
6. 如果工作负载中断，会有多少客户受到影响？
7. 如果工作负载中断，对声誉有何影响？
8. 如果工作负载中断，可能会产生哪些其他运营影响？ 例如，如果电子邮件系统不可用或工资单系统无法提交事务，则会影响员工的工作效率。
9. 工作负载 RTO 和 RPO 如何与业务线和组织 DR 策略保持一致？
10. 是否存在提供服务的内部合同义务？ 不履行这些义务会受到处罚吗？
11. 数据的监管或合规性约束是什么？

实施工作表

您可以将此工作表用于实施步骤 2 和 3。您可以调整此工作表以满足您的特定需求，例如添加其他问题。

步骤 2：主要问题	适用于工作负载？	工作负载 RTO	工作负载 RPO	RTO 调整。	RPO 调整。	说明
[1] 工作负载可以停止的最长时间						以从中断开始到恢复的时间进行衡量
[2] 可以丢失的最大数据量						以从最后一个已知的可恢复数据集算起的时间进行衡量
[3a] 上游依赖关系						输入最严格的上游恢复目标
[3b] 下游依赖关系						输入最不严格的下游恢复目标
[3a] 协调后的上游依赖关系						如果上游值小于当前值，而下游值大于当前值，则使用依赖关系进行协调，
[3b] 协调后的下游依赖关系						并在此处输入协调后的值
[3] 依赖关系						降低值以满足上游依赖关系，或者根据下游依赖关系的能力提高值
步骤 2：其他问题						指出问题是否适用。如果问题不适用，则跳过
基本 RTO/RPO						将 RTO 和 RPO 值从上方向下移到此处
[4] 中断类型	[ ]Y / [ ]N					输入要求最严格的事件类型的恢复目标
[5] 基于时间的具体目标	[ ]Y / [ ]N					输入要求最严格的恢复时间目标
[6] 客户中断	[ ]Y / [ ]N					根据停机时间或数据丢失情况绘制受影响客户的图表。 根据客户影响输入允许的最大的 RTO 和 RPO
[7] 声誉影响	[ ]Y / [ ]N					与业务部门合作，根据对声誉的影响确定最大的 RTO 和 RPO
[8] 运营影响	[ ]Y / [ ]N					根据运营影响输入最大的 RTO 和 RPO
[9] 组织一致性	[ ]Y / [ ]N					根据 LOB 和组织要求，输入此类工作负载的最大 RTO 和 RPO
[10] 合同义务	[ ]Y / [ ]N					根据合同义务输入最大的 RTO 和 RPO
[11] 监管合规性	[ ]Y / [ ]N					根据适用的监管合规性输入最大的 RTO 和 RPO
基于其他问题的目标						从问题 4-11 中取最小值（更严格的值）并在此处输入
调整后的目标						如果无法满足上述目标，请与利益相关者一起放松约束， 并在此处输入新的最小值
调整后的 RTO/RPO						输入基本 RPO/RTO 值或调整后的目标值，以较低者为准
步骤 3						
映射到预定义的类别或层						将这两个值向下调整（更严格），以符合最接近的定义层

工作表

实施计划的工作量级别： 低

资源

相关最佳实践：

- the section called “REL09-BP04 定期执行数据恢复以验证备份完整性和流程”
- the section called “REL13-BP02 使用定义的恢复策略来实现恢复目标”
- the section called “REL13-BP03 测试灾难恢复实施以验证实施效果”

相关文档：

- AWS 架构博客：灾难恢复系列
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- 使用 AWS Resilience Hub 管理弹性策略
- AWS 合作伙伴：可以帮助进行灾难恢复的合作伙伴

- AWS Marketplace：可以用于灾难恢复的产品

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）
- AWS 上工作负载的灾难恢复

REL13-BP02 使用定义的恢复策略来实现恢复目标

定义满足工作负载恢复目标的灾难恢复（DR）策略。选择一种策略，例如备份和还原、备用（主动/被动）或主动/主动。

期望结果：对于每个工作负载，都有一个已定义和实施的 DR 策略，使该工作负载能够实现 DR 目标。工作负载之间的 DR 策略利用可重用模式（例如，前面描述的策略）。

常见反模式：

- 为具有类似 DR 目标的工作负载实施不一致的恢复过程。
- 在发生灾难时临时实施 DR 策略。
- 没有针对灾难恢复的计划。
- 恢复期间依赖于控制面板操作。

建立此最佳实践的好处：

- 通过定义恢复策略，您可以使用常用工具和测试步骤。
- 使用定义的恢复策略，改进团队之间的知识共享，并在他们自己的工作负载上实施 DR。

在未建立这种最佳实践的情况下暴露的风险等级：高。若没有经过计划、实施和测试的 DR 策略，在发生灾难时不太可能实现恢复目标。

实施指导

DR 策略依赖于在主位置无法运行工作负载的情况下，在恢复站点中支持工作负载的能力。最常见的恢复目标是 RTO 和 RPO，相关讨论内容位于 REL13-BP01 定义停机和数据丢失的恢复目标。

跨单个 AWS 区域 内的多个可用区（AZ）的 DR 策略可以缓解火灾、洪水和重大停电等灾难事件。如果需要实施保护措施，为工作负载无法在给定 AWS 区域 中运行这种不太可能发生的事件提供保护，您可以使用跨多个区域的 DR 策略。

在跨多个区域构建 DR 策略时，您应该选择以下策略之一。这些策略按成本和复杂性升序排列，按 RTO 和 RPO 降序排列。 恢复区域指的是 AWS 区域，而不是用于工作负载的主要区域。

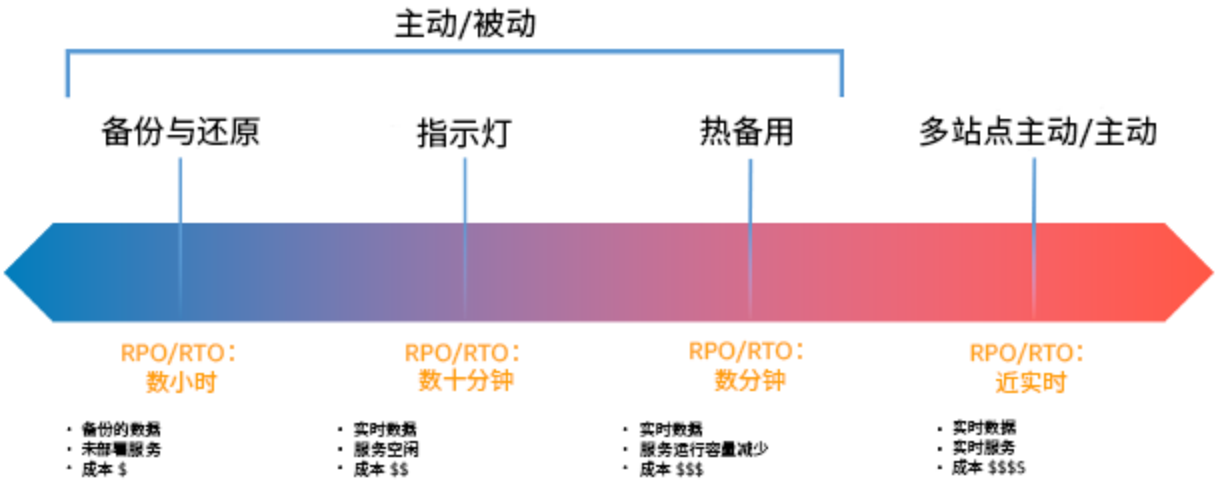


图 17：灾难恢复（DR）策略

- 备份和还原（RPO 以小时为单位，RTO 为 24 小时或更短）：将您的数据和应用程序备份到恢复区域。使用自动或连续备份可以实现时间点故障恢复，在某些情况下，可以将 RPO 降低到 5 分钟。在发生灾难的情况下，您将部署基础设施（使用基础设施即代码来减少 RTO）、部署代码并还原备份的数据，以便在恢复区域从灾难中恢复。
- 指示灯（RPO 以分钟为单位，RTO 为数十分钟）：在恢复区域中预置核心工作负载基础设施的副本。将您的数据复制到恢复区域并在那里创建数据备份。支持数据复制和备份所需的资源（如数据库和对象存储）始终处于启用状态。其他元素（如应用程序服务器或无服务器计算）未部署，但可以在需要时使用必要的配置和应用程序代码创建。
- 热备用（RPO 以秒为单位，RTO 以分钟为单位）：保证在恢复区域中始终运行缩减但功能齐全版本的工作负载。业务关键型系统是完全重复，而且始终可用的系统，只是其队列的规模经过缩减。数据在恢复区域中复制并留存。在需要恢复时，系统会快速扩展以处理生产负载。热备用的规模越大，RTO 和控制面板依赖度就越低。当完全扩展时，这称为热备用服务器。
- 多区域（多站点）主动-主动（RPO 接近于零，RTO 可能为零）：您的工作负载部署到多个 AWS 区域，并且主动处理来自这些区域的流量。此策略要求您跨区域同步数据。必须避免或处理在两个不同区域副本中写入同一记录可能引起的冲突，这会很复杂。数据复制对于数据同步非常有用，并且可以

防止某些类型的灾难，但是它不能防止数据损坏或破坏，除非您的解决方案还包含时间点故障恢复选项。

> **Note**
>
> 指示灯和热备用之间的差异有时难以区分。两者都在恢复区域中包含一个环境，其中具有主区域资产的副本。区别在于，如果不先采取额外措施，指示灯无法处理请求，而热备用可以立即处理流量（容量级别降低）。指示灯将要求您启用服务器，可能需要部署额外的（非核心）基础设施并纵向扩展，而热备用只需要您纵向扩展（所有内容都已部署并运行）。根据您的 RTO 和 RPO 需求在两者之间进行选择。
>
> 当成本是一个问题，并且您希望实现与热备用策略中定义的类似 RPO 和 RTO 目标时，您可以考虑云原生解决方案（例如，AWS Elastic Disaster Recovery），该解决方案采用指示灯方法并提供改进的 RPO 和 RTO 目标。

实施步骤

1. 确定将满足此工作负载恢复要求的 DR 策略。

选择 DR 策略是在减少停机时间和数据丢失（RTO 和 RPO）与策略实施的成本和复杂性之间进行权衡。您应该避免实施比所需策略更严格的策略，因为这会产生不必要的成本。

例如，在下图中，企业已经确定了他们允许的最大 RTO 以及他们可以在服务恢复策略上花费的费用限额。鉴于企业目标，指示灯或热备用这样的 DR 策略将同时满足 RTO 和成本标准。

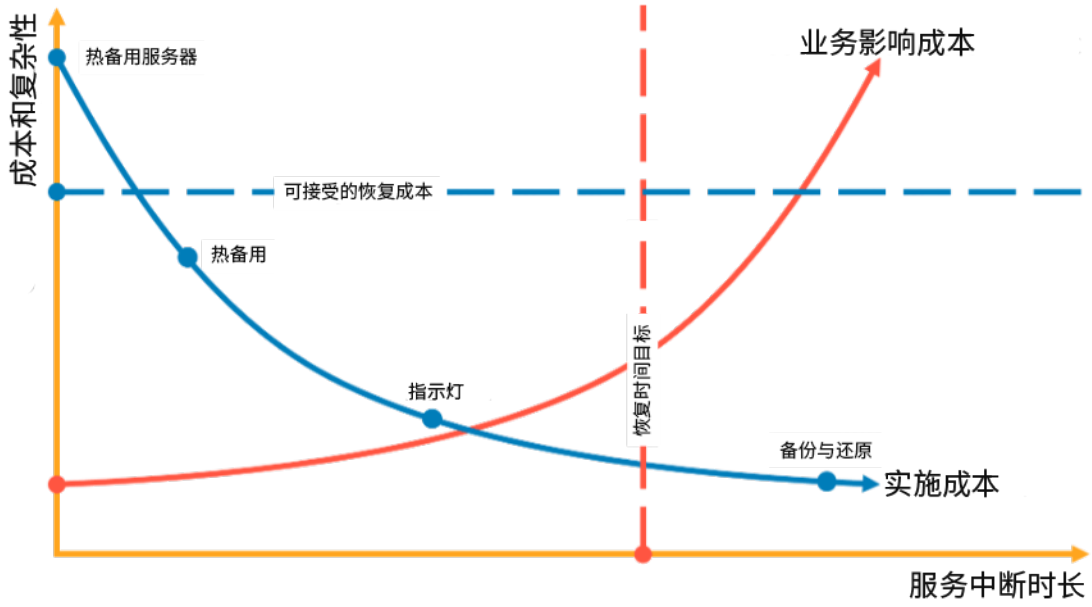


图 18：根据 RTO 和成本选择 DR 策略

如需了解更多信息，请参阅业务连续性计划（BCP）

2. 查看如何实施所选 DR 策略的模式。

这一步是了解如何实施所选策略。这些策略可以解释为使用多个 AWS 区域 作为主要站点和恢复站点。不过，您也可以选择使用单个区域内的多个可用区作为 DR 策略，这将利用多个策略的元素。

在后续步骤中，您可以对特定的工作负载应用策略。

备份和还原

备份和还原是实施起来最简单的策略，但需要更多时间和工作来恢复工作负载，从而导致更高的 RTO 和 RPO。最好的做法是，始终备份数据并将数据备份复制到另一个站点（如另一个 AWS 区域）。

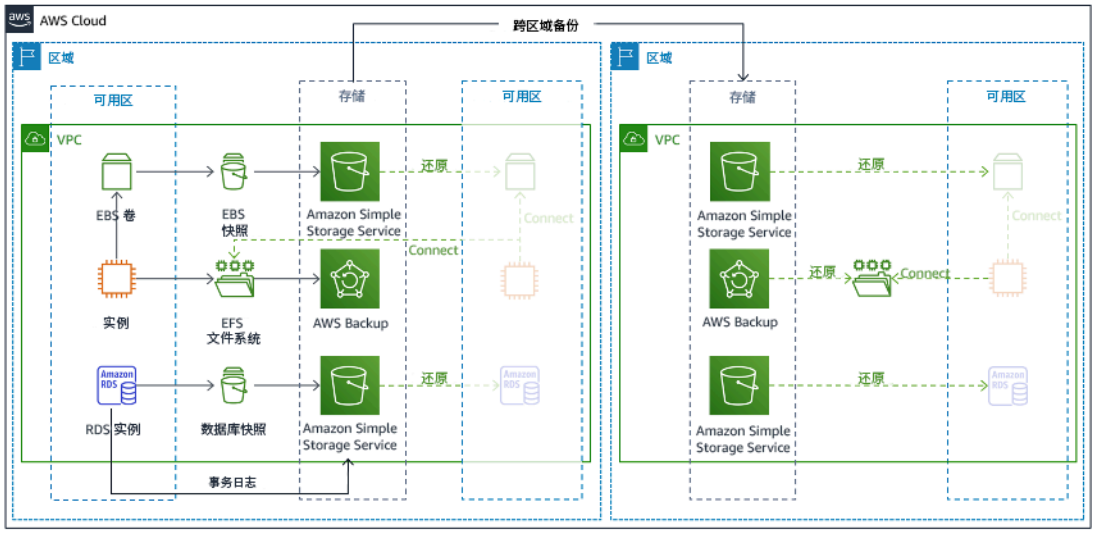


图 19：备份和还原架构

有关此策略的更多详细信息，请参阅 AWS 上的灾难恢复（DR）架构，第 II 部分：使用快速恢复功能的备份与还原。

指示灯

利用指示灯方法，您可以将数据从主要区域复制到恢复区域。用于工作负载基础设施的核心资源部署在恢复区域中，但仍需要额外的资源和所有依赖项才能使此恢复区域成为功能堆栈。例如，在图 20 中，没有部署计算实例。

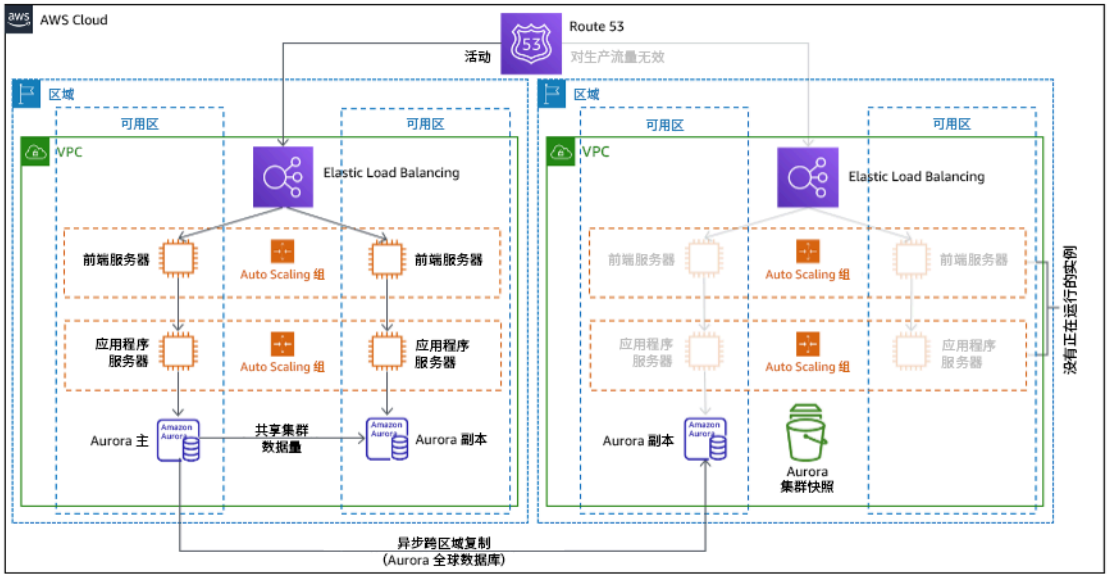


图 20：指示灯架构

有关此策略的更多详细信息，请参阅 AWS 上的灾难恢复（DR）架构，第 III 部分：指示灯和热备用。

热备用

热备用方法涉及到确保在另一个区域中存在生产环境的规模缩减但功能齐全的副本。这种方法扩展了指示灯概念并减少了恢复时间，因为您的工作负载始终在另一个区域中运行。如果以全部容量部署恢复区域，那么这种方式称为热备用。

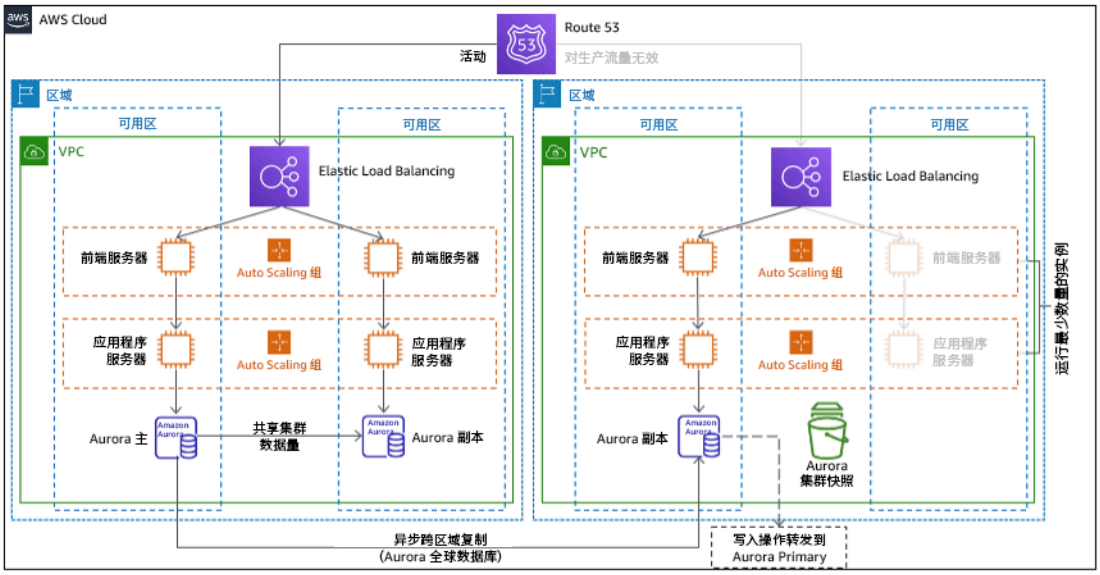


图 21：热备用架构

使用热备用或指示灯需要扩展恢复区域中的资源。为确保在需要时有可用的容量，请考虑使用 EC2 实例的容量预留。如果使用 AWS Lambda，那么预置并发可以提供执行环境，以便它们准备好立即响应函数的调用。

有关此策略的更多详细信息，请参阅 AWS 上的灾难恢复（DR）架构，第 III 部分：指示灯和热备用。

多站点主动/主动

作为多站点主动/主动策略的一部分，您可以在多个区域中同时运行工作负载。多站点主动/主动策略处理来自其部署到的所有区域的流量。客户可能会出于 DR 以外的原因选择此策略。此策略可以用于提高可用性，或者在向全球受众部署工作负载时（使端点更靠近用户和/或部署针对该区域受众的本地化堆栈）使用此策略。作为一种 DR 策略，如果工作负载在部署此策略的某个 AWS 区域 中不能得到支持，那么该区域将被撤出，使用其余区域维持可用性。多站点主动/主动策略是 DR 策略中操作最复杂的策略，只有在业务需求时才应选择它。

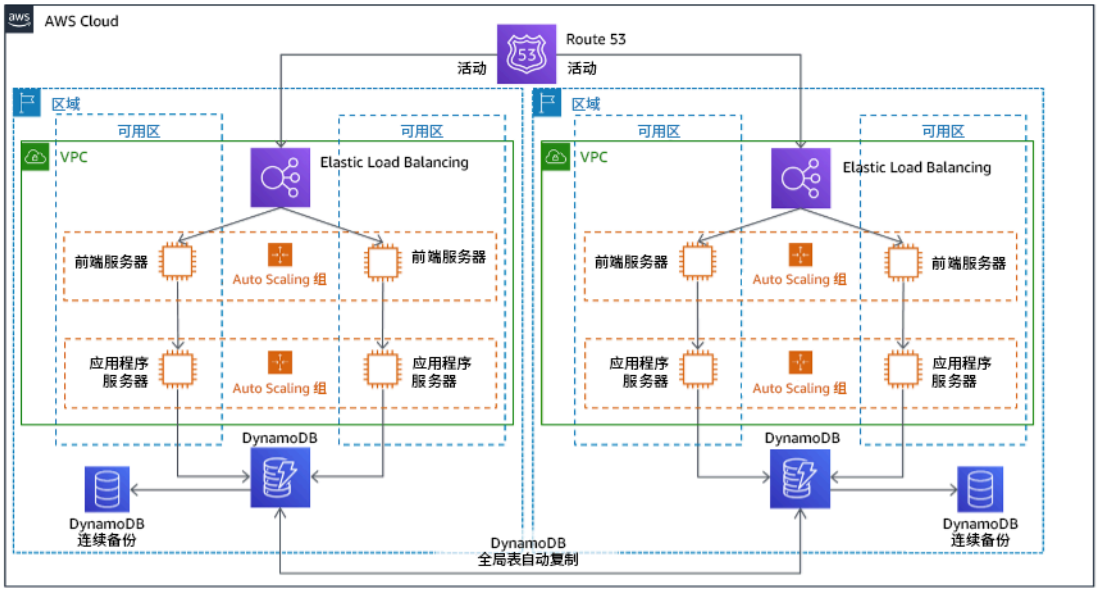


图 22：多站点主动/主动架构

有关此策略的更多详细信息，请参阅 AWS 上的灾难恢复（DR）架构，第 IV 部分：多站点主动/主动。

AWS Elastic Disaster Recovery

如果您正考虑为灾难恢复使用指示灯或热备用策略，AWS Elastic Disaster Recovery 可以提供一种带来更多好处的替代方法。Elastic Disaster Recovery 可以提供类似于热备用方法的 RPO 和 RTO 目标，同时保持指示灯方法的低成本。Elastic Disaster Recovery 将数据从主区域复制到恢复区域，使用持续数据保护来实现以秒为单位的 RPO 和以分钟为单位的 RTO。在恢复区域中仅部署复制数据所需的资源，从而降低成本，类似于指示灯策略。使用 Elastic Disaster Recovery 时，如果在失效转移或演练过程中启动，则服务会协调和编排计算资源的恢复。

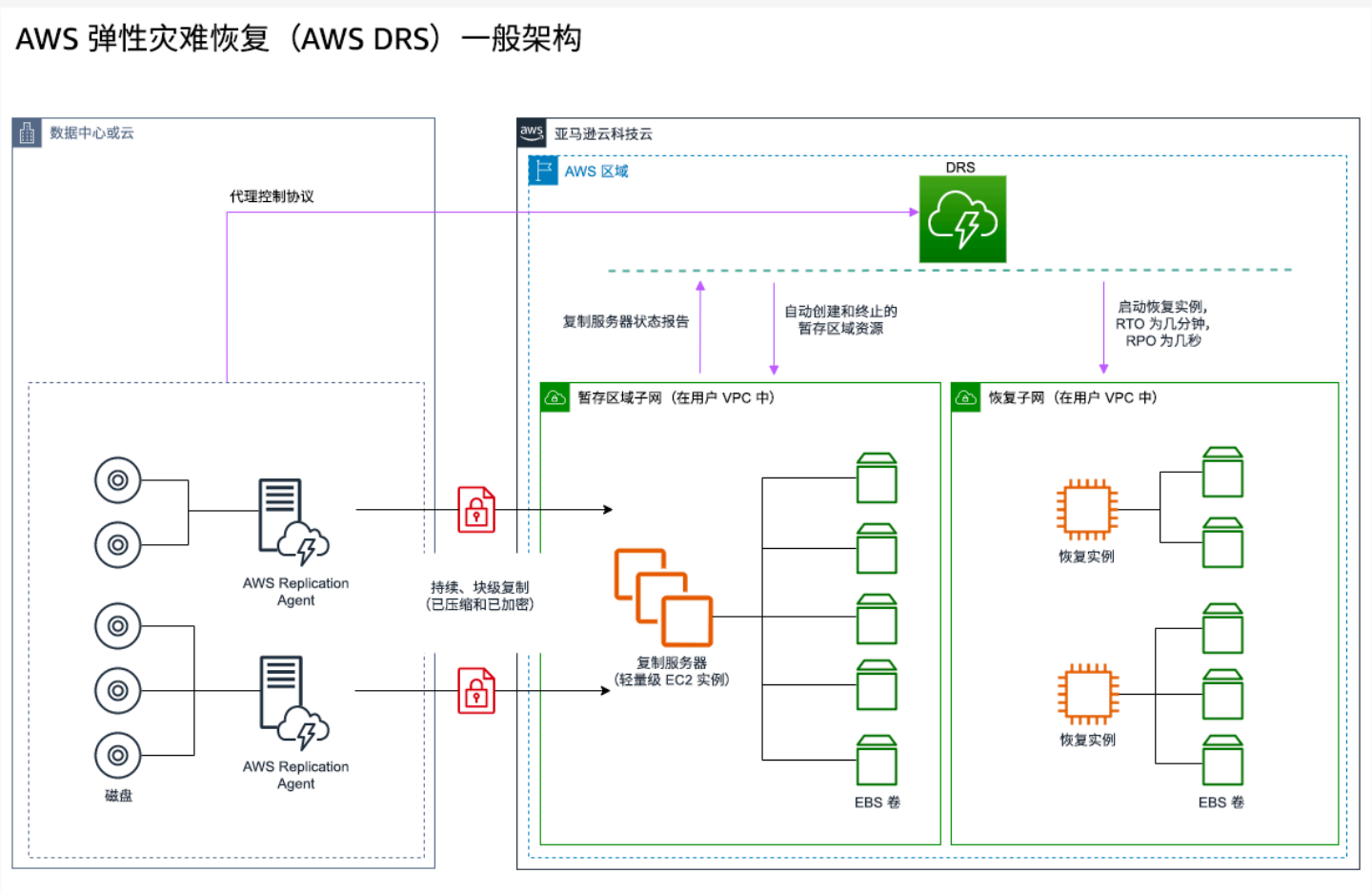


图 23：AWS Elastic Disaster Recovery 架构

其他保护数据的实践

对于所有这些策略，您还必须减轻数据灾难的影响。持续的数据复制可以防止某些类型的灾难，但它可能无法防止数据损坏或破坏，除非您的策略还包括存储数据的版本控制或用于时间点故障恢复的选项。除了副本之外，您还必须备份恢复站点中的复制数据以创建时间点备份。

使用单个 AWS 区域 内的多个可用区（AZ）

使用单个区域内的多个可用区时，您的 DR 实施会使用上述策略的多个元素。首先，您必须使用多个可用区创建一个高可用性（HA）架构，如图 23 所示。此架构使用多站点主动/主动方法，因为 Amazon EC2 实例和 Elastic Load Balancer 在多个可用区中部署了资源，主动处理请求。此架构还演示了热备用服务器方法，如果主 Amazon RDS 实例出现故障（或可用区本身出现故障），则备用实例将提升为主实例。

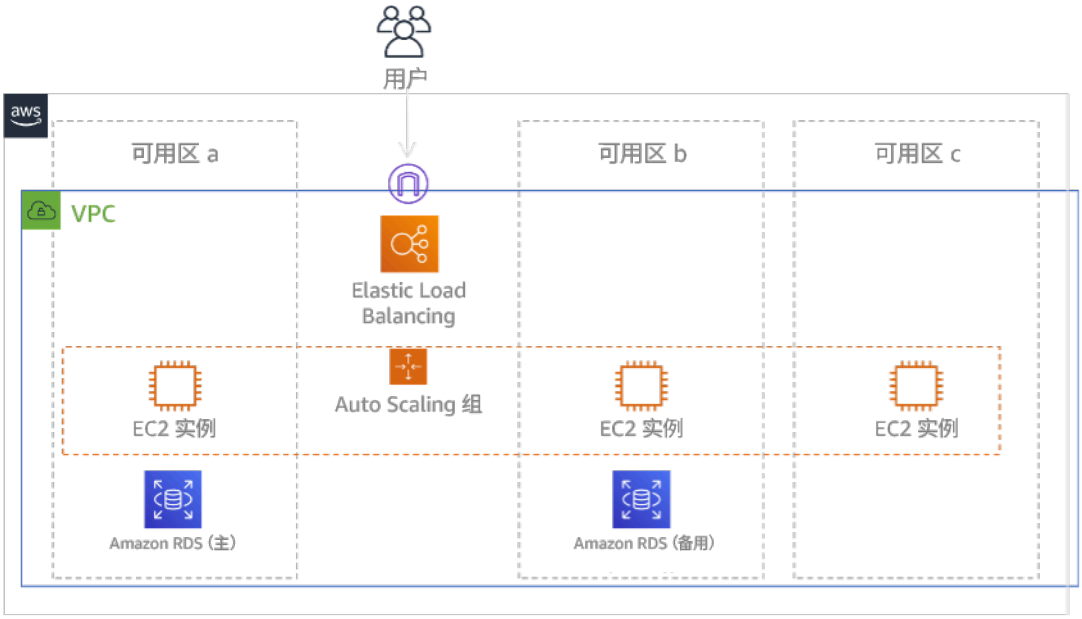


图 24：多可用区架构

除了这种 HA 架构之外，您还需要添加运行工作负载所需的所有数据的备份。这对于限制在单个区的数据尤其重要，例如 Amazon EBS 卷或 Amazon Redshift 集群。如果一个可用区发生故障，您需要将这些数据恢复到另一个可用区。如果可能，您还应该将数据备份复制到另一个 AWS 区域，提供另一层保护。

下面的博客文章中介绍了一种不太常见的单区域多可用区 DR 的替代方法：使用 Amazon Route 53 Application Recovery Controller 构建高弹性应用程序，第 1 部分：单区域堆栈。这里的策略是尽可能保持可用区之间的隔离，就像区域的运作方式一样。使用这种替代策略，您可以选择主动/主动或主动/被动方法。

> **Note**
>
> 某些工作负载具有数据驻留法规要求。如果这适用于当前只有一个 AWS 区域的位置的工作负载，那么多区域将不适合您的业务需求。多可用区策略可以很好地抵御大多数灾难。

3. 评估工作负载的资源，以及失效转移之前（正常操作期间）恢复区域中的资源配置。

对于基础设施和 AWS 资源，使用基础设施即代码功能（如 AWS CloudFormation）或第三方工具（如 Hashicorp Terraform）。要使用单个操作跨多个账户和区域部署，您可以使用 AWS CloudFormation StackSets。对于多站点主动/主动和热备用服务器策略，恢复区域中部署的基础设施具有与主区域相同的资源。对于指示灯和热备用策略，部署的基础设施将需要额外的操作才可用于生产。使用

CloudFormation 参数和条件逻辑，您可以通过单个模板控制部署的堆栈是活动还是备用。使用 Elastic Disaster Recovery 时，服务会复制和编排应用程序配置和计算资源的还原。

所有 DR 策略都要求在 AWS 区域内备份数据源，然后将这些备份复制到恢复区域。AWS Backup 提供了一个集中视图，您可以在其中配置、调度和监控这些资源的备份。对于指示灯、热备用和多站点主动/主动方法，您还应该将数据从主区域复制到恢复区域中的数据资源，例如 Amazon Relational Database Service（Amazon RDS）数据库实例或 Amazon DynamoDB 表。因此，这些数据资源处于活动状态，可以随时处理恢复区域中的请求。

要了解更多关于 AWS 服务如何跨区域运行的信息，请参阅以下博客系列：使用 AWS 服务创建多区域应用程序。

4. 确定并实施措施，让恢复区域在需要时（在灾难事件期间）可以进行失效转移。

对于多站点主动/主动策略，失效转移意味着撤离一个区域，并依赖剩余的活动区域。通常，这些区域已准备好接受流量。对于指示灯和热备用策略，恢复操作将需要部署缺失的资源（如图 20 中的 EC2 实例），以及任何其他缺失的资源。

对于上述所有策略，您可能需要将数据库的只读实例提升为主读/写实例。

对于备份和还原，从备份中还原数据时会为该数据创建资源，例如 EBS 卷、RDS 数据库实例和 DynamoDB 表。您还需要还原基础设施并部署代码。您可以使用 AWS Backup 来还原恢复区域中的数据。请参阅 REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据 了解更多详细信息。重建基础设施包括创建资源，例如，EC2 实例以及所需的 Amazon Virtual Private Cloud（Amazon VPC）、子网和安全组。您可以自动执行大部分还原过程。要了解具体方法，请参阅这篇博客文章。

5. 确定并实施措施，以在需要时（在灾难事件期间）可以重新路由流量进行失效转移。

此失效转移操作可以自动或手动启动。应谨慎使用基于运行状况检查或警报自动启动的失效转移，因为不必要的失效转移（误报）会产生不可用和数据丢失等成本。因此，通常会使用手动启动的失效转移。在这种情况下，您仍然应该自动执行失效转移步骤，这样手动启动就像按一下按钮一样简单。

在使用 AWS 服务时，需要考虑几个流量管理选项。一个选项是使用 Amazon Route 53。使用 Amazon Route 53，您可以将一个或多个 AWS 区域中的多个 IP 端点与一个 Route 53 域名相关联。要实施手动启动的失效转移，您可以使用 Amazon Route 53 Application Recovery Controller，它提供高度可用的数据面板 API 以将流量重新路由到恢复区域。实施失效转移时，使用数据面板操作并避免控制面板操作，如以下部分所述： REL11-BP04 恢复期间依赖于数据平面而不是控制平面。

要了解有关此选项及其他选项的更多信息，请参阅灾难恢复白皮书的这一部分。

6. 设计工作负载的失效自动恢复计划。

失效自动恢复是指在灾难事件消除后将工作负载运营恢复到主区域。向主区域预置基础设施和代码通常遵循最初使用的相同步骤，依赖于基础设施即代码和代码部署管道。失效自动恢复面临的挑战是还原数据存储，并确保它们与运行中的恢复区域保持一致。

在失效转移状态下，恢复区域中的数据库处于活动状态，并且具有最新数据。然后，目标是从恢复区域重新同步到主区域，确保主区域是最新的。

某些 AWS 服务会自动执行此操作。如果使用 Amazon DynamoDB 全局表，即使主区域中的表不可用，当它重新联机时，DynamoDB 也会继续传播任何挂起的写操作。如果使用 Amazon Aurora 全局数据库并使用托管的计划失效转移，则维护 Aurora 全局数据库的现有复制拓扑。因此，主区域中以前的读/写实例将成为副本，并从恢复区域接收更新。

如果这不是自动执行的，您将需要在主区域中重新建立数据库，作为恢复区域中数据库的副本。在许多情况下，这将涉及删除旧的主数据库，然后创建新的副本。例如，有关如何使用 Amazon Aurora 全局数据库对计划外失效转移执行此操作的说明，请参阅下面的实验：全局数据库的失效自动恢复。

失效转移后，如果您可以继续在恢复区域中运行，请考虑将此区域设为新的主区域。您仍然需要执行上述所有步骤，将以前的主区域变成恢复区域。有些组织会进行定期轮换，定期交换其主区域和恢复区域（例如每三个月一次）。

失效转移和失效自动恢复所需的所有步骤都应保存在行动手册且可供所有团队成员使用，并定期进行审查。

使用 Elastic Disaster Recovery 时，服务会协助编排和自动执行失效自动恢复流程。有关更多详细信息，请参阅执行失效自动恢复。

实施计划的工作量级别：高

资源

相关最佳实践：

- the section called “REL09-BP01 识别和备份需要备份的所有数据，或从源复制数据”
- the section called “REL11-BP04 恢复期间依赖于数据平面而不是控制平面”
- the section called “REL13-BP01 定义停机和数据丢失的恢复目标”

相关文档：

- AWS 架构博客：灾难恢复系列
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- 云中的灾难恢复选项
- 在一小时内构建无服务器多区域、双活后端解决方案
- 多区域无服务器后端 – 重新加载
- RDS：跨区域复制只读副本
- Route 53：配置 DNS 故障转移
- S3：跨区域复制
- 什么是 AWS Backup？
- 什么是 Route 53 Application Recovery Controller？
- AWS 弹性灾难恢复
- HashiCorp Terraform：入门 – AWS
- AWS 合作伙伴：可以帮助进行灾难恢复的合作伙伴
- AWS Marketplace：可以用于灾难恢复的产品

相关视频：

- AWS 上工作负载的灾难恢复
- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）
- 开始使用 AWS 弹性灾难恢复 | Amazon Web Services

相关示例：

- Well-Architected 实验室 - 灾难恢复 - 说明 DR 策略的系列研讨会

REL13-BP03 测试灾难恢复实施以验证实施效果

定期测试到恢复站点的失效转移，以验证是否在正常运行，并满足 RTO 和 RPO。

常见反模式：

- 切勿在生产环境中练习失效转移。

建立此最佳实践的好处：定期测试您的灾难恢复计划，确认该计划在需要时能够正常发挥作用，并且您的团队知道如何执行该策略。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

要避免的模式是制定了恢复路径但很少测试。例如，您可能有一个用于只读查询的辅助数据存储。当您写入某个数据存储，却发现主存储故障时，您可能希望将故障转移到辅助数据存储。如果您不经常测试此故障转移，可能会发现您关于辅助数据存储容量的假设是错误的。辅助数据存储容量在您上次测试时可能是足够的，但可能无法再容纳这次情况下的负载。我们的经验表明，唯一有效的错误恢复是您经常测试的路径。因此，最好只开发几条恢复路径。您可以建立恢复模式并定期对其进行测试。如果恢复路径比较复杂或至关重要，您仍需定期在生产环境中测试该故障，确保恢复路径有效。在我们刚才讨论的示例中，您应该定期将故障转移到备用存储，无论是否有需要。

实施步骤

1. 为灾难恢复设计工作负载。定期测试恢复路径。面向恢复的计算可识别系统中能够增强恢复功能的特性：隔离和冗余，系统范围回滚更改的能力，监控并确定运行状况的能力，提供诊断、自动恢复、模块化设计的能力，以及重启的能力。练习恢复路径，以确认您可以在指定时间内恢复到指定状态。在此恢复过程中使用运行手册来记录问题，并在下一次测试之前找到问题的解决方案。
2. 对于基于 Amazon EC2 的工作负载，使用 AWS Elastic Disaster Recovery 为 DR 策略实施和启动演练实例。AWS Elastic Disaster Recovery 可以高效地运行演练，从而帮助您为失效转移事件做好准备。您还可以使用 Elastic Disaster Recovery 频繁地启动实例以用于测试和演练目的，无需重定向流量。

资源

相关文档：

- AWS 合作伙伴：可以帮助进行灾难恢复的合作伙伴
- AWS 架构博客：灾难恢复系列
- AWS Marketplace：可以用于灾难恢复的产品
- AWS Elastic Disaster Recovery
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- AWS Elastic Disaster Recovery 为失效转移做准备
- 加州大学伯克利分校/斯坦福大学的面向恢复的计算项目

- 什么是 AWS Fault Injection Simulator？

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式
- AWS re:Invent 2019：AWS 的备份与还原，以及灾难恢复解决方案

相关示例：

- Well-Architected 实验室 – 测试弹性

REL13-BP04 管理 DR 站点或区域的配置偏差

确保 DR 站点或区域的基础设施、数据和配置满足需求。例如，检查 AMI 和服务限额是否为最新。

AWS Config 会持续监控和记录 AWS 资源配置。它可以检测到偏差并触发 AWS Systems Manager Automation 进行修复和发出警报。AWS CloudFormation 还可以在您已部署的堆栈中检测到偏差。

常见反模式：

- 在主位置进行配置或基础设施更改时，未能在恢复位置进行更新。
- 不考虑主位置和恢复位置的潜在限制（如服务区别）。

建立此最佳实践的好处： 确保您的 DR 环境与现有环境一致，可保证完整恢复。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

- 确保您的交付管道可交付到主站点和备份站点。用于将应用程序部署到生产中的交付管道必须分布到所有指定的灾难恢复策略位置，包括开发和测试环境。
- 启用 AWS Config 来跟踪潜在偏差位置。使用 AWS Config 规则来创建可强制实施灾难恢复策略并在检测到偏差时生成提醒的系统。
  - 按照 AWS Config 规则 修正不合规 AWS 资源
  - AWS Systems Manager Automation
- 使用 AWS CloudFormation 部署基础设施。AWS CloudFormation 可以检测 CloudFormation 模板指定的内容和实际部署内容之间的偏差。

- AWS CloudFormation：在整个 CloudFormation 堆栈上检测偏差

资源

相关文档：

- AWS 合作伙伴：可以帮助进行灾难恢复的合作伙伴
- AWS 架构博客：灾难恢复系列
- AWS CloudFormation：在整个 CloudFormation 堆栈上检测偏差
- AWS Marketplace：可以用于灾难恢复的产品
- AWS Systems Manager Automation
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- 如何在 AWS 上实施基础设施配置管理解决方案？
- 按照 AWS Config 规则 修正不合规 AWS 资源

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）

REL13-BP05 自动执行恢复

利用 AWS 或第三方工具自动进行系统恢复，并将流量路由至 DR 站点或区域。

根据已配置的运行状况检查，Elastic Load Balancing 和 AWS Auto Scaling 等 AWS 服务可将负载分配到运行正常的可用区，而 Amazon Route 53 和 AWS Global Accelerator 等服务则可将负载路由到运行正常的 AWS 区域。Amazon Route 53 Application Recovery Controller 可帮助您使用就绪检查和路由控制功能来管理和协调失效转移操作。这些功能持续监控您的应用程序从故障中恢复的能力，因此您可以跨多个 AWS 区域、可用区和本地部署控制您的应用程序恢复。

对于现有的物理或虚拟数据中心或私有云上的工作负载，AWS 弹性灾难恢复（通过 AWS Marketplace 提供）使组织能够设置自动向 AWS 进行灾难恢复的策略。CloudEndure 还支持 AWS 中的跨区域/跨可用区灾难恢复。

常见反模式：

- 实施相同的自动故障转移和故障恢复可能会导致在故障时发生摆动。

建立此最佳实践的好处： 自动恢复通过消除发生手动错误的可能性来缩短恢复时间。

未建立这种最佳实践的情况下暴露的风险等级： 中

实施指导

- 恢复路径自动化。如果恢复时间很短，人工判断和操作无法用于可用性非常高的场景。在这种情况下，系统每次必须自动进行恢复。
  - 使用 CloudEndure Disaster Recovery 自动执行失效转移和故障恢复操作。CloudEndure Disaster Recovery 可持续将您的计算机（包括操作系统、系统状态配置、数据库、应用程序和文件）复制到目标 AWS 账户和首选区域中的低成本暂存区域。在发生灾难时，您可以指示 CloudEndure Disaster Recovery 在几分钟内自动启动数千台处于完全预置状态的计算机。
    - 执行灾难恢复故障转移和故障恢复
    - CloudEndure Disaster Recovery

资源

相关文档：

- AWS 合作伙伴：可以帮助进行灾难恢复的合作伙伴
- AWS 架构博客：灾难恢复系列
- AWS Marketplace：可以用于灾难恢复的产品
- AWS Systems Manager Automation
- AWS 的 CloudEndure Disaster Recovery
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）

相关视频：

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式（ARC209-R2）

可用性目标的实施示例

在这个部分，我们将介绍采用典型 Web 应用程序部署的工作负载设计，其中包括：反向代理、Amazon S3 上的静态内容、应用程序服务器以及用于持久性数据存储的 SQL 数据库。对于每个可用性目标，我们都将介绍一个实施示例。此工作负载可能将容器或 AWS Lambda 用于数据库的计算和 NoSQL（如 Amazon DynamoDB），但所采用的方法都是相似的。在每个情境中，我们将介绍如何通过针对此类主题的工作负载设计来达成可用性目标：

主题	有关更多信息，请参阅此章节
监控资源	监控工作负载资源
适应需求的变化	设计工作负载以适应需求的变化
实施变更	实施变更
备份数据	备份数据
构建弹性	使用故障隔离，以保护您的工作负载 将工作负载设计为能够承受组件故障的影响
测试弹性	测试可靠性
灾难恢复 (DR) 计划	灾难恢复 (DR) 计划

依赖项选择

我们已选择将 Amazon EC2 用于应用程序。我们将介绍如何使用 Amazon RDS 和多个可用区来提高应用程序的可用性。我们会将 Amazon Route 53 用于 DNS。使用多个可用区时，我们将使用 Elastic Load Balancing。Amazon S3 用于备份和静态内容。由于我们的设计追求更高的可靠性，因此必须采用本身具有更高可用性的服务。

单区域场景

主题

- 2 个 9（99%）场景

2 个 9（99%）场景

这些工作负载对业务有帮助，但如果不可用， 它们只会带来 不便。此类工作负载可以是内部工具、内部知识管理或项目跟踪。或者，它们可以是来自实验性服务但实际面向客户的工作负载，并且具有在必要时切换以隐藏该服务的功能。

这些工作负载可以部署为一个区域和一个可用区。

监控资源

我们将进行简单的监控，看看服务主页是否会返回 HTTP 200 OK 状态。出现问题后，我们的行动手册会说明，可以使用实例中的日志来确定根本原因。

适应需求的变化

我们拥有关于常见硬件故障、紧急软件更新和其他破坏性更改的行动手册。

实施变更

我们将使用 AWS CloudFormation 来定义我们的“基础设施即代码”，特别是在发生故障时加快重建速度。

软件更新会使用运行手册手动执行，安装和重新启动服务需要停机。如果在部署过程中出现问题，运行手册会介绍如何回滚到早期版本。

运维和开发团队将使用日志分析来更正全部错误，并且在确定修复工作的优先级并完成修复工作以后才部署更正。

备份数据

我们将使用供应商或专用的备份解决方案，按照运行手册，将加密备份数据发送给 Amazon S3。我们将按照运行手册，通过定期还原数据并确保能使用它们，测试备份数据能正常使用。我们会在 Amazon S3 对象上启用版本控制，并取消备份删除权限。我们会根据要求，使用 Amazon S3 存储桶生命周期策略，进行存档或永久删除操作。

构建弹性

工作负载会部署为一个区域和一个可用区。我们会在单个实例中部署应用程序，包括数据库。

测试弹性

新软件的部署管道已安排好，并进行了一些单元测试，主要是组装工作负载的白箱/黑箱测试。

灾难恢复 (DR) 计划

如果发生了故障，我们会等待故障结束，选择通过运行手册，使用 DNS 修改将请求路由到静态网站。具体的恢复时间取决于可部署基础设施的速度以及数据库还原到最近备份的速度。按照运行手册，如果某个可用区发生了故障，可在同一可用区或其他可用区中实施部署。

可用性设计目标

我们用 30 分钟了解并决定执行恢复，用 10 分钟在 AWS CloudFormation 中部署整个堆栈，假设我们部署到一个新的可用区，并假设数据库可以在 30 分钟内还原。这就意味着要从故障中恢复，大约需要 70 分钟的时间。假设每季度发生一次故障，预计全年的受影响时间为 280 分钟，也就是 4 小时 40 分钟。

这意味着可用性上限是 99.9%。实际可用性还取决于实际故障率、故障持续时间以及每次故障的实际恢复速度。在这种架构中，应用程序需要离线才能更新（每年预计 24 小时：每年六次更改，每次四小时），还要考虑实际事件。因此，参考白皮书前面的应用程序可用性表，我们看到 可用性设计目标 是 99%。

总结

主题	实施
监控资源	仅站点运行状况检查；无提醒。
适应需求的变化	通过重新部署以进行垂直扩展。
实施变更	用于部署与回滚的运行手册。
备份数据	用于备份与还原的运行手册。
构建弹性	完整重建；从备份恢复。
测试弹性	完整重建；从备份恢复。
灾难恢复 (DR) 计划	备份加密，在必要时还原至不同的可用区。

3 个 9（99.9%）场景

下一个可用性目标针对务必要具备高可用性的应用程序，但是这些应用程序可以承受短时间的不可用。这种类型的工作负载通常用于内部操作，而且此类操作在发生故障时会对员工造成影响。这种类型的工作负载也可能是面向客户的，它们不会带来很高的业务收入，并且可以承受较长的恢复时间或恢复点。这类工作负载包括账户或信息管理的管理应用程序。

我们可以使用两个可用区进行部署，并将应用程序分到不同的层中，从而改进工作负载的可用性。

监控资源

通过在主页上检查 HTTP 200 OK 状态，监控可以扩展为对网站上的可用性发出提醒。此外，每次更换 Web 服务器时、数据库故障转移时，系统都会发出提醒。我们还将监控 Amazon S3 上的静态内容以了解可用性，并在其不可用时发出提醒。日志记录将被汇总，以便于管理并帮助进行根本原因分析。

适应需求的变化

配置自动扩展以监控 EC2 实例上的 CPU 利用率，通过添加或移除实例将 CPU 目标维持在 70%，而且每个可用区内有不少于一个的 EC2 实例。若 RDS 实例上的负载模式显示需要进行扩展，我们将在维护时段更改实例类型。

实施变更

基础设施部署技术与之前的场景相同。

按每两到四周一次的固定日程安排交付新软件。软件更新将自动完成，不是使用 Canary 部署或蓝/绿部署模式，而是使用在位替换。回滚决策将使用运行手册做出。

我们将使用行动手册来找出问题的根本原因。在确定根本原因之后，运营和开发团队会共同确定错误修正方案，并在开发出解决方案后进行部署。

备份数据

备份和还原操作可以通过使用 Amazon RDS 完成。它将使用运行手册定期执行，以确保我们可以满足恢复要求。

构建弹性

我们可以使用两个可用区进行部署，并将应用程序分到不同的层中，从而改进应用程序的可用性。我们将使用跨多个可用区工作的服务，如 Elastic Load Balancing、Auto Scaling 和 Amazon RDS 多可用区部署，并通过 AWS Key Management Service 实现加密存储。这将确保在资源级别和可用区级别上能够容忍故障。

负载均衡器只会将流量路由到正常运行的应用程序实例。运行状况检查需要在数据平面/应用程序层上进行，以表明应用程序在实例上的容量。此检查不应针对控制平面。系统将提供 Web 应用程序运行状况检查 URL，并配置为可供负载均衡器和 Auto Scaling 使用，以便能够删除和替换发生故障的实例。如果实例在主可用区内发生故障，Amazon RDS 将管理活动数据库引擎，使其在第二个可用区可用，然后执行修复以还原到相同弹性。

在分层之后，我们可以使用分布式系统弹性模式提高应用程序的可靠性，甚至当数据库在可用区故障转移过程中暂时不可用时，让应用程序仍然可用。

测试弹性

与之前的场景一样，我们会执行功能测试。我们不会测试 ELB、自动扩展或 RDS 故障转移的自我修复功能。

.我们将提供行动手册，其中包含针对常见数据库问题、安全相关事件，以及部署失败的相关解决方案。

灾难恢复 (DR) 计划

整个工作负载恢复和常用报告都有运行手册。恢复会使用备份，而且这些备份被存储在与工作负载相同的区域。

可用性设计目标

假设有一些故障必须要人工决定执行恢复。但为了尽可能提高这种场景下的自动化程度，我们假设每年只有两个事件需要做这种决定。我们需要 30 分钟决定执行恢复，并在 30 分钟内完成恢复。这意味着从故障中恢复需要 60 分钟。假设一年发生两次故障，预计每年受影响的时间为 120 分钟，

这意味着可用性上限是 99.95%。实际可用性还取决于实际故障率、故障持续时间以及每次故障的实际恢复速度。对于此架构，我们需要应用程序暂时离线以执行更新，但这些更新自动进行。我们估计每年需要为此花费 150 分钟：每次更改 15 分钟，每年 10 次。服务不可用的时间是每年总计 270 分钟，所以我们的 可用性设计目标 是 99.9%。

总结

主题	实施
监控资源	仅站点运行状况检查；在发生故障时发出提醒。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；调整多可用区 RDS 的大小。

主题	实施
实施变更	自动部署到位和用于回滚的运行手册。
备份数据	通过 RDS 自动化备份，以满足 RPO 和用于恢复的运行手册的要求。
构建弹性	自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区。
测试弹性	ELB 和应用程序会自我修复；RDS 是多可用区；无显式测试。
灾难恢复 (DR) 计划	通过 RDS 将加密备份存储于相同的 AWS 区域。

4 个 9（99.99%）场景

这个应用程序可用性目标需要应用程序具有较高的可用性并且可以承受组件故障。应用程序必须能够承受故障，而无需购买更多资源。此可用性目标针对的是任务关键型应用程序，这些应用程序是电子商务网站、企业对企业 Web 服务或高流量的内容/媒体站点等此类企业主要或重要的收入推动因素。

我们可以使用区域内静态稳定 的架构 进一步提高可用性。此可用性目标不需要控制平面改变工作负载的行为来承受故障。例如，应该会有足够的容量来承受一个可用区的损失。我们不要求更新到 Amazon Route 53 DNS。我们不需要创建任何新的基础设施，无论它是创建或修改 S3 存储桶、创建新的 IAM 策略（或修改策略），还是修改 Amazon ECS 任务配置。

监控资源

监控内容包括成功指标以及发生问题时的提醒。此外，每次更换出现故障的 Web 服务器时、数据库故障转移时以及可用区出现故障时，系统都会发出提醒。

适应需求的变化

我们将使用 Amazon Aurora 作为我们的 RDS，它让只读副本可以进行自动扩展。对于这些应用程序，主要内容的读取可用性优于写入可用性的设计也是一个关键的架构决策。Aurora 还可以视需要自动增加存储，以 10 GB 为单位，最高可达 64 TB。

实施变更

我们将使用 Canary 部署或蓝绿部署，将更新独立部署到每个隔离区中。部署是完全自动化的，包括在 KPI 表明存在问题时的回滚。

运行手册中应提出严格的报告要求和性能跟踪。如果成功运营趋向于无法实现性能或可用性目标，则将使用行动手册来确定导致这一趋势的原因。行动手册可用于确定未发现的故障模式和安全事件，还可用于确定发生故障的根本原因。我们还将与 AWS Support 一起提供基础设施事件管理产品。

负责构建与运营网站的团队需要确定任何意外故障的错误更正，并确定实施修复程序后进行部署的优先级。

备份数据

备份和还原操作可以通过使用 Amazon RDS 完成。它将使用运行手册定期执行，以确保我们可以满足恢复要求。

构建弹性

我们建议针对此方法使用三个可用区。使用三可用区部署，每个可用区最多拥有 50% 的静态容量。也可以使用两个可用区，但静态稳定容量的成本会更高，因为两个区域都必须拥有 100% 的峰值容量。我们将添加 Amazon CloudFront 以提供地理缓存，以及减少应用程序数据平面上的请求。

我们将使用 Amazon Aurora 作为我们的 RDS，并在所有三个区域内部署只读副本。

应用程序将在所有层上使用软件/应用程序弹性模式进行构建。

测试弹性

部署管道具有完整的测试套件，包括性能、负载和故障注入测试.

我们将在实际试用期间不断训练我们的故障恢复程序，使用运行手册确保我们可以执行任务而不会偏离程序。构建网站的团队也负责运营该网站。

灾难恢复 (DR) 计划

整个工作负载恢复和常用报告都有运行手册。恢复会使用备份，而且这些备份被存储在与工作负载相同的区域。Game Day 期间会定期演练还原程序。

可用性设计目标

假设有一些故障必须要手动决定执行恢复，尽管有更好的自动化选项。假定每年只有两个事件需要做这种决定，那么恢复操作会很快。我们需要 10 分钟决定执行恢复，并在五分钟内完成恢复。这意味着故障恢复时间为 15 分钟。假设一年发生两次故障，预计每年受影响的时间为 30 分钟，

这意味着可用性上限是 99.99%。实际可用性还将取决于实际故障率、故障持续时间以及每个因素的实际恢复速度。对于这种架构，我们假设应用程序通过更新持续在线。基于此，我们的 可用性设计目标是 99.99%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在多个区域为 Aurora RDS 自动扩展存储与只读副本。
实施变更	自动 Canary 或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚。隔离区域会执行部署。
备份数据	通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。
构建弹性	为应用程序实施故障隔离区；自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程。
灾难恢复 (DR) 计划	通过 RDS 将加密备份存储于相同的 AWS 区域，并在实际试用期间演练。

多区域场景

在多个 AWS 区域中实施应用程序将增加运营成本，部分原因是我们需要隔离区域以保持其自主权。采用这种方法是一项深思熟虑的决定。也就是说，区域提供了强大的隔离边界，我们要竭尽全力避免跨区域的相关故障。使用多个区域，在区域性 AWS 服务发生硬件依赖性故障时，可以更好地控制恢复时间。在本部分中，我们将讨论各种实施模式及其常规可用性。

主题

3½ 个 9 (99.95%)，故障恢复时间介于 5 到 30 分钟

应用程序的这种可用性目标要求停机时间极短，且特定时间内的数据丢失极少。具有此可用性目标的应用程序涉及以下领域：银行、投资、紧急服务和数据捕获。这些应用程序要求非常短的恢复时间和恢复点。

通过使用 热备用 方法（跨两个 AWS 区域），我们可以进一步缩短恢复时间。我们将整个工作负载同时部署到两个区域，缩减我们的被动站点并且使所有数据保持最终一致性。两个部署在其对应的区域内均 静态稳定 。应用程序应采用分布式系统弹性模式进行构建。我们将需要创建轻量级 路由 组件监控工作负载的运行状况，并且可在必要时配置为将流量路由到被动区域。

监控资源

每次更换 Web 服务器时、数据库故障转移时以及区域出现故障时，系统都会发出提醒。我们还将监控 Amazon S3 上的静态内容以了解可用性，并在其不可用时发出提醒。日志记录将被汇总，以便于管理并帮助在每个区域进行根本原因分析。

路由组件会监控我们的应用程序运行状况，以及我们所拥有的任何区域硬依赖关系。

适应需求的变化

与 4 个 9 场景相同。

实施变更

按每两到四周一次的固定日程安排交付新软件。软件更新将使用金丝雀部署或蓝/绿部署模式自动完成。

发生区域故障转移时、这些事件期间发生常见客户问题时，以及需要常规报告时，可以参考运行手册。

我们将提供行动手册，其中提供了常见数据库问题、安全相关事件、部署失败、区域故障转移的意外客户问题，以及确定问题根本原因的相关解决方案。在确定根本原因之后，运营和开发团队会共同确定错误修正方案，并在开发出解决方案后进行部署。

我们还将与 AWS Support 一起提供基础设施事件管理。

备份数据

类似于 4 个 9 场景，我们使用自动 RDS 备份并使用 S3 版本控制。数据会自动并异步从主动区域的 Aurora RDS 集群被复制到被动区域的跨区域只读副本。S3 跨区域副本被用于自动并异步将数据从主动区域移动到被动区域。

构建弹性

与 4 个 9 场景相同，而且区域故障转移也是有可能的。它采用手动管理。在故障转移期间，我们将使用 DNS 故障转移将请求路由到静态网站，直到在第二个区域中恢复。

测试弹性

与 4 个 9 场景相同，我们将使用运行手册，并且通过实际试用验证架构。另外，对于立即实施与部署，RCA 更正优先于功能发布

灾难恢复 (DR) 计划

手动管理区域故障转移。所有数据都会被异步复制。扩展 热备用 内的基础设施。可通过在 AWS Step Functions 上执行的工作流对其进行自动化。AWS Systems Manager (SSM) 也可在您创建 SSM 文档更新 Auto Scaling 组并调整实例大小时帮助其自动化。

可用性设计目标

假设有一些故障必须要手动决定执行恢复，尽管有自动化选项。假定每年只有两个事件需要做这种决定，我们需要 20 分钟决定执行恢复，并在 10 分钟内完成恢复。这就意味着要从故障中恢复，大约需要 30 分钟的时间。假设一年发生两次故障，预计每年受影响的时间为 60 分钟，

这意味着可用性上限是 99.95%。实际可用性还将取决于实际故障率、故障持续时间以及每个因素的实际恢复速度。对于这种架构，我们假设应用程序通过更新持续在线。基于此，我们的 可用性设计目标为 99.95%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查，包括 AWS 区域级别的 DNS 运行状况；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在主动和被动区域内的多个区域为 Aurora RDS 自动扩展存储与只读副本。在 AWS 区域之间同步数据和基础设施以获得静态稳定性。
实施变更	自动金丝雀或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚，每次在一个 AWS 区域内部署到一个隔离的区域。
备份数据	在每个 AWS 区域内通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。Aurora RDS 和 S3 数据会从主动区域被自动并异步复制到被动区域。
构建弹性	自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区；在故障转移时，若出现静态站点，则手动管理区域故障转移。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程，以及用于传达问题出在哪里和如何加以更正或预防的通信路径。对于立即实施与部署，RCA 更正优先于功能发布。
灾难恢复 (DR) 计划	将热备用部署到其他区域。使用通过 AWS Step Functions 或 AWS Systems Manager 文档执行的工作流扩展基础设施。通过 RDS 加密备份。两个 AWS 区域之间的跨区域只读副本。在

主题	实施
	Amazon S3 中跨区域复制静态资产。还原到最新的主动 AWS 区域，在实际试用期间演练，并与 AWS 协调。

5 个 9（99.999%）或更高的场景，恢复时间不到一分钟

应用程序的这种可用性目标要求几乎无停机时间，且特定时间内几乎没有数据丢失。具有此可用性目标的应用程序包括，例如某些银行、投资、金融、政府和关键业务应用程序，它们是极其庞大的创收公司的核心业务。其目标是在所有层实现强一致的数据存储和完全冗余。我们选择了一个基于 SQL 的数据存储。但在某些情况下，我们很难实现非常小的 RPO。如果您可以对数据进行分区，就可能不会出现数据丢失。这可能需要您添加应用程序逻辑和延迟，以确保在多个地理位置之间拥有一致的数据，以及在分区之间移动或复制数据的功能。如果使用 NoSQL 数据库，执行此分区可能会更容易。

我们可以使用 主动-主动 方法（跨多个 AWS 区域）来进一步提高可用性。工作负载将部署到全部所需的跨区域 静态稳定 的区域（因此其余的区域可以在丢失一个区域时处理负载）。A 路由 层将流量传送至运行状况良好的地理位置，并在这些位置的运行状况不佳时自动更改目标，同时临时停止数据复制层。Amazon Route 53 可提供间隔 10 秒的运行运行状况检查，还可以在您的记录集上提供低至 1 秒的 TTL。

监控资源

与 3½ 个 9 场景相同，而且将在检测到区域运行不正常时发出提醒，流量会被路由移出该区域。

适应需求的变化

与 3½ 个 9 场景相同。

实施变更

部署管道具有完整的测试套件，包括性能、负载和故障注入测试。我们将使用金丝雀部署或蓝/绿部署将更新部署到隔离区域中，一次部署到一个区域，部署完成后再从另一个区域开始。在部署期间，旧版本仍将在实例上继续运行以便更快地回滚。这些是完全自动化的，包括在 KPI 表明存在问题时的回滚。监控内容包括成功指标以及发生问题时的提醒。

运行手册中应提出严格的报告要求和性能跟踪。如果成功运营趋向于无法实现性能或可用性目标，则将使用行动手册来确定导致这一趋势的原因。行动手册可用于确定未发现的故障模式和安全事件，还可用于确定发生故障的根本原因。

构建网站的团队也负责运营该网站。团队需要确定任何意外故障的错误更正，并确定实施修复程序后进行部署的优先级。我们还将与 AWS Support 一起提供基础设施事件管理。

备份数据

与 3½ 个 9 场景相同。

构建弹性

应用程序应使用软件/应用程序弹性模式进行构建。实现所需的可用性可能还需要许多其他路由层。不要低估这种额外实施的复杂性。该应用程序将在部署故障隔离区域中实施，并进行分区和部署，这样一来，即使是区域级事件也不会影响所有客户。

测试弹性

我们将在实际试用期间不断验证架构，使用运行手册确保我们可以执行任务而不会偏离程序。

灾难恢复 (DR) 计划

主动-主动 多区域部署，将完整的工作负载基础设施和数据部署到多个区域。使用本地读取、全局写入策略时，一个区域是所有写操作的主数据库，数据将复制到其他区域以供读取。如果主数据库区域出现故障，则需要提升一个新数据库。本地读取，全局写入会把用户分配到主区域，并在该区域处理 DB 写入。这让用户能够从任何区域读取或写入，但需要复杂的逻辑以管理不同区域内的写入之间的潜在数据冲突。

当某区域被检测到运行不正常时，路由层会将流量自动路由到其余运行正常的区域。无需人工干预。

数据存储必须以可解决潜在冲突的方式在区域之间复制。由于存在延迟，您需要创建工具和自动化流程，以在各分区之间复制或移动数据，并平衡每个分区中的请求或数据量。您需要提供额外的运营手册，介绍数据冲突的补救措施。

可用性设计目标

假设我们进行了大量投资以自动执行所有恢复，且恢复可以在一分钟内完成。假设没有手动触发的恢复，但每季度最多有一次自动恢复操作，这意味着每年的恢复时间为四分钟。假设应用程序通过更新持续在线，基于此，我们的 可用性设计目标 是 99.999%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查，包括 AWS 区域级别的 DNS 运行状况；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在主动和被动区域内的多个区域为 Aurora RDS 自动扩展存储与只读副本。在 AWS 区域之间同步数据和基础设施以获得静态稳定性。
实施变更	自动金丝雀或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚，每次在一个 AWS 区域内部署到一个隔离的区域。
备份数据	在每个 AWS 区域内通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。Aurora RDS 和 S3 数据会从主动区域被自动并异步复制到被动区域。
构建弹性	为应用程序实施故障隔离区；自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区；自动区域故障转移。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程，以及用于传达问题出在哪里和如何加以更正或预防的通信路径。对于立即实施与部署，RCA 更正优先于功能发布。
灾难恢复 (DR) 计划	在至少两个区域内执行主动-主动部署。跨区域基础设施完全扩展并且静态稳定。跨区域对数据进行分区与同步。通过 RDS 加密备份。在实际试用期间演练区域故障，并与 AWS 协调。在还原期间，可能需要提升一个新的主数据库。

资源

文档

- Amazon Builders' Library - Amazon 如何构建与运营软件
- AWS Architecture Center

实验室

- AWS Well-Architected 可靠性实验

外部链接

- 适应性队列模式： 大规模故障
- 可用性及其他内容：了解和提高 AWS 上分布式系统的弹性

图书

- Robert S. Hammer“适用于容错软件的模式”
- Andrew Tanenbaum 和 Marten van Steen“分布式系统：原则与范例”

总结

无论您是新手，刚开始接触可用性和可靠性主题，还是经验法丰富的老手，希望寻求见解以最大限度提高关键任务型工作负载的可用性，我们都希望本白皮书能够引发您的思考、提供新想法或引出新的质疑。我们希望它可以帮助您基于业务需求更深入地了解正确的可用性级别，以及如何设计可靠性加以实现。我们鼓励您利用本部分提供的设计、面向运维和面向恢复的建议，以及 AWS 解决方案架构师的知识和经验。我们非常期望收到您的反馈，尤其是您在 AWS 上实现高可用性的成功案例。请联系您的客户团队，或使用 网站的“联系我们”.

贡献者

本文档的贡献者包括：

- Seth Eliot，Amazon Web Services Principal Developer Advocate
- Mahanth Jayadeva，Amazon Web Services Well-Architected 部门的 Solutions Architect
- Amulya Sharma，Amazon Web Services Principal Solutions Architect
- Jason DiDomenico，Amazon Web Services Cloud Foundations 部门的 Senior Solutions Architect
- Marcin Bednarz，Amazon Web Services Principal Solutions Architect
- Tyler Applebaum，Amazon Web Services Senior Solutions Architect
- Rodney Lester，Amazon Web Services App Modernization 部门的 Principal Solutions Architect
- Joe Chapman，Amazon Web Services Senior Solutions Architect
- Adrian Hornsby，Amazon Web Services Principal System Development Engineer
- Kevin Miller，Amazon Web Services S3 部门的 Vice President
- Shannon Richards，Amazon Web Services Principal Technical Program Manager
- Laurent Domb，Amazon Web Services Fed Fin 部门的 Chief Technologist
- Kevin Schwarz，Amazon Web Services Sr. Solutions Architect
- Rob Martell，Amazon Web Services Principal Cloud Resilience Architect
- Priyam Reddy，Amazon Web Services DR 部门的 Senior Solutions Architect Manager
- Jeff Ferris，Amazon Web Services Principal Technologist
- Matias Battaglia，Amazon Web Services Senior Solutions Architect

延伸阅读

如需更多信息，请参阅：

- AWS Well-Architected Framework
- AWS Architecture Center

文档修订

要获得有关此白皮书的更新通知，请订阅 RSS 源。

变更	说明	日期
已更新白皮书	为最佳实践更新了新的实施指导。	June 27, 2024
更新了最佳实践指南	根据以下领域的新指南更新了最佳实践：在分布式系统中设计交互以预防发生故障、在分布式系统中设计交互以缓解或经受住故障的影响、监控工作负载资源、设计工作负载以适应需求的变化、实施变更和测试可靠性。	December 6, 2023
更新了最佳实践指南	根据以下领域的新指南更新了最佳实践：监控工作负载资源和将工作负载设计为能够承受组件故障的影响。	October 3, 2023
更新了最佳实践指南	根据以下领域的新指南更新了最佳实践：设计工作负载服务架构、在分布式系统中设计交互以缓解或经受住故障的影响以及监控工作负载资源。	July 13, 2023
次要更新	删除非包容性用语。	April 13, 2023
针对新框架进行了更新	为最佳实践更新了规范性指南并增加了新的最佳实践。	April 10, 2023
已更新白皮书	为最佳实践更新了新的实施指导。	December 15, 2022

次要更新	更正了图表编号和贯穿全文的次要更改。	November 17, 2022
已更新白皮书	扩展了最佳实践并增加了改进计划。	October 20, 2022
已更新白皮书	在使用故障隔离来保护工作负载和将工作负载设计为能够承受组件故障的影响部分的可靠性支柱中新增了两项最佳实践。	May 5, 2022
次要更新	在简介中添加了可持续性支柱。	December 2, 2021
已更新白皮书	更新灾难恢复指南以包括 Route 53 Application Recovery Controller。添加对 DevOps Guru 的引用。更新了多个资源链接，以及其他小的编辑更改。	October 26, 2021
次要更新	添加了有关 AWS Fault Injection Service（AWS FIS）的信息。	March 15, 2021
次要更新	较小的文本更新。	January 4, 2021

已更新白皮书	更新了附录 A 以更新 Amazon SQS、Amazon SNS 和 Amazon MQ 的可用性设计目标；对表中的行重新排序以方便查找；更好地解释可用性和灾难恢复之间的差异，以及它们如何促进实现韧性；扩大多区域架构（实现可用性）和多区域策略（实现灾难恢复）的覆盖范围；将参考书更新至最新版本；扩展可用性计算以包括基于请求的计算和简化算法；改进对实际试用的描述	December 7, 2020
次要更新	更新了附录 A 以更新 AWS Lambda 的可用性设计目标	October 27, 2020
次要更新	更新了附录 A 以添加 AWS Global Accelerator 的可用性设计目标	July 24, 2020

新框架的更新	大量更新和全新/修订内容，包括：新增“工作负载架构”最佳实践章节，重新整理最佳实践，并将其纳入“变更管理和故障管理”章节，更新了资源，经过更新以包括最新的 AWS 资源和服务，如 AWS Global Accelerator、AWS Service Quotas 和 AWS Transit Gateway，新增/更新可靠性、可用性和韧性的定义，根据用于 Well-Architected 审查的 AWS Well-Architected Tool（问题和最佳实践）对白皮书进行调整，重新排列设计原则的顺序，将自动从故障中恢复移动到测试恢复程序之前，更新了等式的图表和格式，删除了“关键服务”章节，并将关键 AWS 服务的参考整合到最佳实践当中。	July 8, 2020
次要更新	修复错误的链接	October 1, 2019
已更新白皮书	已更新附录 A	April 1, 2019
已更新白皮书	新增特定的 AWS Direct Connect 联网推荐和更多服务设计目标	September 1, 2018
已更新白皮书	新增“设计原则”和“限制管理”章节。更新链接，删除上游/下游术语的歧义，并为可用性场景中的其余“可靠性支柱”主题新增详细的参考。	June 1, 2018

已更新白皮书	已将 DynamoDB 跨区域解决方案更改为 DynamoDB 全局表。新增服务设计目标	March 1, 2018
次要更新	对可用性计算进行细微更正，以使其包括应用程序可用性	December 1, 2017
已更新白皮书	经过更新以提供关于高可用性设计的指导，包括概念、最佳实践和示例实施。	November 1, 2017
原始版本	发布了可靠性支柱 – AWS Well-Architected Framework。	November 1, 2016