AWS Well-Architected 框架

可靠性支柱

可靠性支柱: AWS Well-Architected 框架

Table of Contents

可靠性支柱 – AWS Well-Architected Framework

发布日期：2024 年 11 月 6 日（文档修订）

本白皮书主要介绍 AWS Well-Architected Framework 的可靠性支柱。文中所述指导可帮助客户在 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

AWS Well-Architected Framework 能够帮助您理解在 AWS 上构建工作负载时所做决策的利弊。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。该框架提供了一种统一的方法，让您能够根据最佳实践衡量架构，从而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，您会构建具有强大基础、韧性架构、一致的变更管理和经过验证的故障恢复流程的架构。

本白皮书的目标读者是技术岗位的人员，例如首席技术官（CTO）、架构师、开发人员和运营团队成员。阅读本白皮书后，您将了解可在设计云架构来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。这包括在其全部生命周期内运行和测试工作负载的能力。本白皮书深入介绍了有关在 AWS 中实施可靠工作负载的最佳实践指导。

主题

韧性的责任共担模式

韧性是 AWS 和您的共同责任。您要了解作为韧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 – 云的韧性

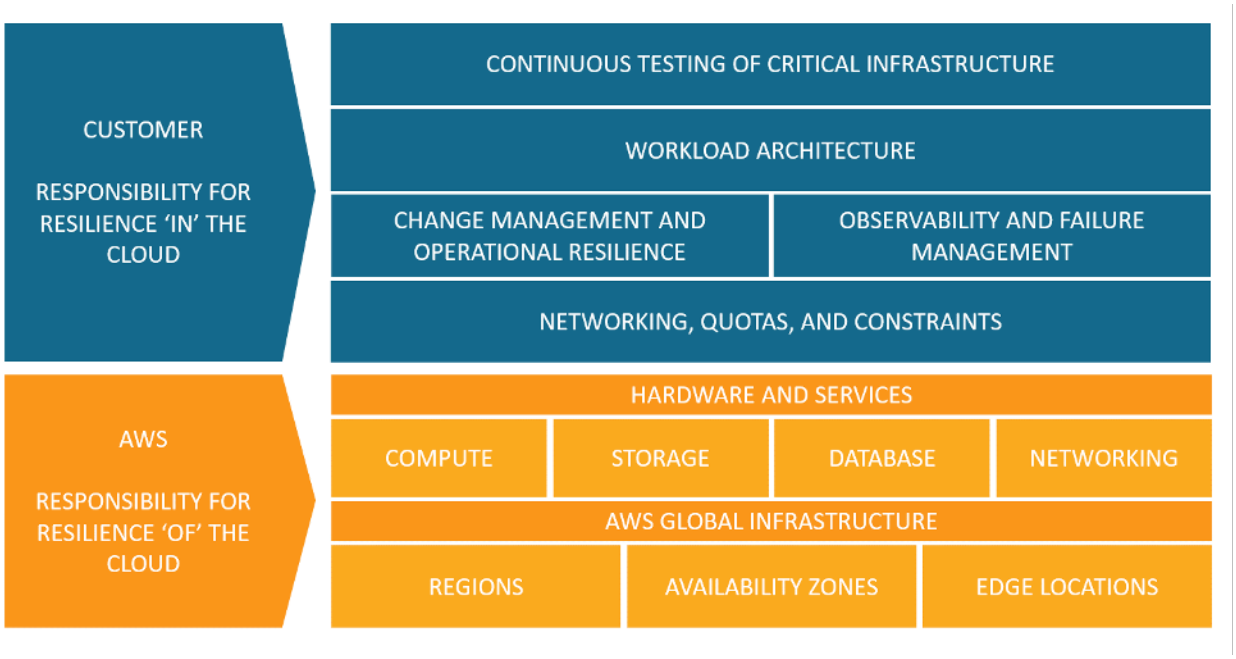
对于运行 AWS Cloud 中提供的所有服务的基础设施，AWS 负责维持其韧性。此基础设施包括运行 AWS Cloud 服务的硬件、软件、网络和设施。AWS 采取合理的商业措施来提供这些 AWS Cloud 服务，确保服务可用性达到或超过 AWS 服务水平协议（SLA）。

AWS 全球云基础设施旨在让客户能够构建高韧性的工作负载架构。每个 AWS 区域 都完全隔离，由多个可用区组成，这些可用区是基础设施的物理隔离分区。可用区会隔离可能影响工作负载韧性的故障，防止这些故障影响区域内的其他可用区。与此同时，AWS 区域 中的所有可用区都通过完全冗余的专用城域光纤，以高带宽、低延迟的网络进行互联，从而在可用区之间实现高吞吐量、低延迟的联网。可用区之间的所有流量都经过加密。网络性能足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，公司可以实现更好的隔离和保护，防止受到停电、雷击、龙卷风、飓风等事故和灾害的影响。

客户责任 – 云中的韧性

您的责任由您选择的 AWS Cloud 服务决定。这决定了您承担韧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韧性配置和管理任务。部署 Amazon EC2 实例的客户负责在多个位置部署 Amazon EC2 实例（例如 AWS可用区），使用自动扩缩等服务实施自我修复，以及对安装在实例上的应用程序使用韧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 会运营基础设施层、操作系统和平台，而客户访问端点即可存储和检索数据。您负责管理数据的韧性，包括备份、版本控制和复制策略。

在 AWS 区域 的多个可用区中部署工作负载是高可用性策略的一部分，该策略将问题隔离在一个可用区，同时使用其他可用区的冗余继续处理请求，以此保护工作负载。多可用区架构也是灾难恢复策略的一部分，旨在更好地隔离工作负载，防止受到停电、雷击、龙卷风、地震等事故和灾害的影响。灾难恢复策略也可以使用多个 AWS 区域。例如，在主动/被动配置中，如果主动区域不再处理请求，则工作负载的服务会从其主动区域失效转移到其灾难恢复区域。



客户和 AWS 负责的云中的韧性与云的韧性

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

联网、配额和限制

- 基础部分详细介绍了责任共担模式这一领域的最佳实践。
- 根据预期的负载请求增加情况（如果适用），规划留足了扩展空间的架构，了解所包含服务的服务配额和限制。
- 设计网络拓扑，使其具有高可用性、冗余性和可扩展性。

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。实施变更需要构建运行手册并让其保持最新状态，也需要为应用程序和基础设施制定部署策略。
- 监控工作负载资源的韧性策略考虑了所有组件，包括技术和业务指标、通知、自动化以及分析。

- 云中的工作负载必须适应需求变化，能根据受损情况或使用量波动情况进行横向缩减。

可观测性和故障管理

- 必须通过监控来观测故障才能自动执行修复，让工作负载能够承受组件故障。
- 故障管理要求备份数据，运用最佳实践让工作负载能够承受组件故障,以及规划灾难恢复。

工作负载架构

- 工作负载架构包括如何围绕业务领域设计服务、运用 SOA 和分布式系统设计来防止发生故障，以及如何内置节流、重试、队列管理、超时和应急措施等功能。
- 依靠经过验证的 AWS 解决方案、Amazon Builders' Library 和 serverless patterns 来与最佳实践保持一致，从而快速启动实施。
- 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引入变更和实现创新的工作，提高完成这类工作的能力。

关键基础设施的持续测试

- 测试可靠性意味着在功能、性能和混乱层面进行测试，也意味着采用事件分析和演练日活动实践来积累专业知识，解决人们不太了解的问题。
- 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。
- 创建并记录可重复的试验，了解事情未按预期发展时系统有何表现。这些测试会证明整体韧性的有效性，并在面对真正的故障场景之前为运营程序提供反馈环路。

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。这可实现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，可以在故障发生之前预测和修复故障。
- 测试恢复程序：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，确保不会共用常见故障点。
- 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，但有些配额是可控的，其他配额也可以管理（请参阅管理服务配额和限制）。
- 通过自动化管理变更：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：

- 基本原理
- 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，而基础是服务配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

- 韧性与可靠性的组件
- 可用性
- 灾难恢复（DR）目标

韧性与可靠性的组件

云中的工作负载可靠性取决于多个因素，其中最主要的要属韧性：

- 韧性是指工作负载从基础设施故障或服务中断中恢复，并能动态获取计算资源来满足需求以及减少中断（如错误配置或暂时性网络问题）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，以及通过运营准备情况审查（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，避免影响可用性。例如，使用加密备份来确保数据安全。
- 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。

韧性是本白皮书的主要关注点。

其他四个因素也很重要，我们将在讨论 AWS Well-Architected Framework 的相应支柱时加以介绍。这里的许多最佳实践也解决了可靠性在这些方面的问题，但重点是韧性。

可用性

可用性（又称为服务可用性）既是定量衡量韧性的常用指标，也是具有针对性的韧性目标。

- 可用性是指工作负载可供使用的时间百分比。

可供使用指的是在需要时能够履行约定的功能。

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（包括计划内和计划外的中断）时，可用性就会降低。我们按以下方式计算可用性：

$$Availability = \frac{Available\ for\ Use\ Time}{Total\ Time}$$

- 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，有些客户选择在总时间中排除计划内的维护停机时间（例如计划内维护）。但不建议采用这种方法，因为用户可能希望在这些时间内使用您的服务。

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最长中断时间。该表包含我们常常会在每个可用性层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、传输和加载作业
99.9%	8 小时 45 分钟	知识管理、项目跟踪等内部工具
99.95%	4 小时 22 分钟	电子商务、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

根据请求测量可用性：对服务而言，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，可以采用如下计算公式：

$$Availability = \frac{Successful\ Responses}{Valid\ Requests}$$

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%。

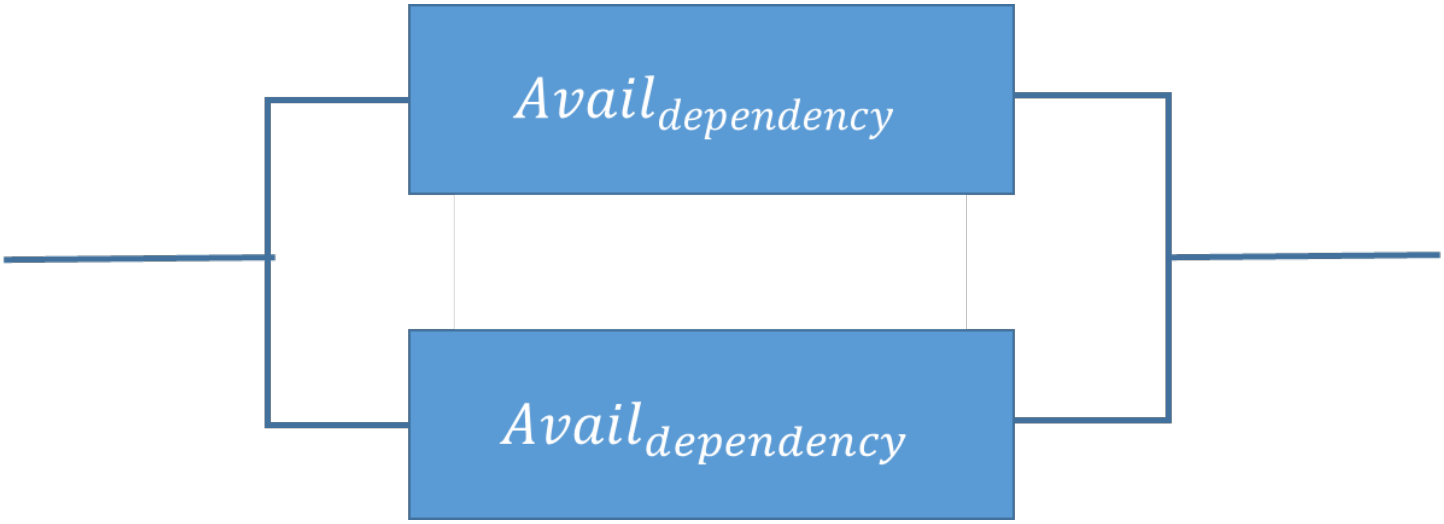
针对硬依赖关系计算可用性：许多系统对其他系统具有硬依赖关系，依赖的系统中的中断会直接转换为调用系统的中断。这与软依赖关系相反，其中依赖的系统的故障会在应用程序中得到弥补。在出现此类硬依赖关系的情况下，调用系统的可用性是依赖的系统可用性的结果。例如，如果您有一个旨在实现 99.99% 可用性的系统，它对两个其他独立系统具有硬依赖关系，这两个系统都旨在实现 99.99% 的可用性，则工作负载在理论上可以实现 99.97% 的可用性：

$Avail_{invok} \times Avail_{dep1} \times Avail_{dep2} = Avail_{workload}$

99.99% × 99.99% × 99.99% = 99.97%

因此，在计算可用性时，您一定要了解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%：



$Avail_{effective} = Avail_{MAX} - ((100\% - Avail_{dependency}) \times (100\% - Avail_{dependency}))$

$99.9999\% = 100\% - (0.1\% \times 0.1\%)$

快捷方式计算：如果计算中所有组件的可用性仅包含数字九，则可以将九的数量相加得出答案。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）和平均恢复时间（MTTR）。可以通过以下公式来确定可用性估算值：

$$Avail_{EST} = \frac{MTBF}{MTBF + MTTR}$$

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

有关更多详细信息，请参阅 Availability and Beyond: Understanding and improving the resilience of distributed systems on AWS，了解如何计算可用性。

可用性的成本：设计应用程序来实现更高级别的可用性通常会导致成本增加，因此在开始设计应用程序之前，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。

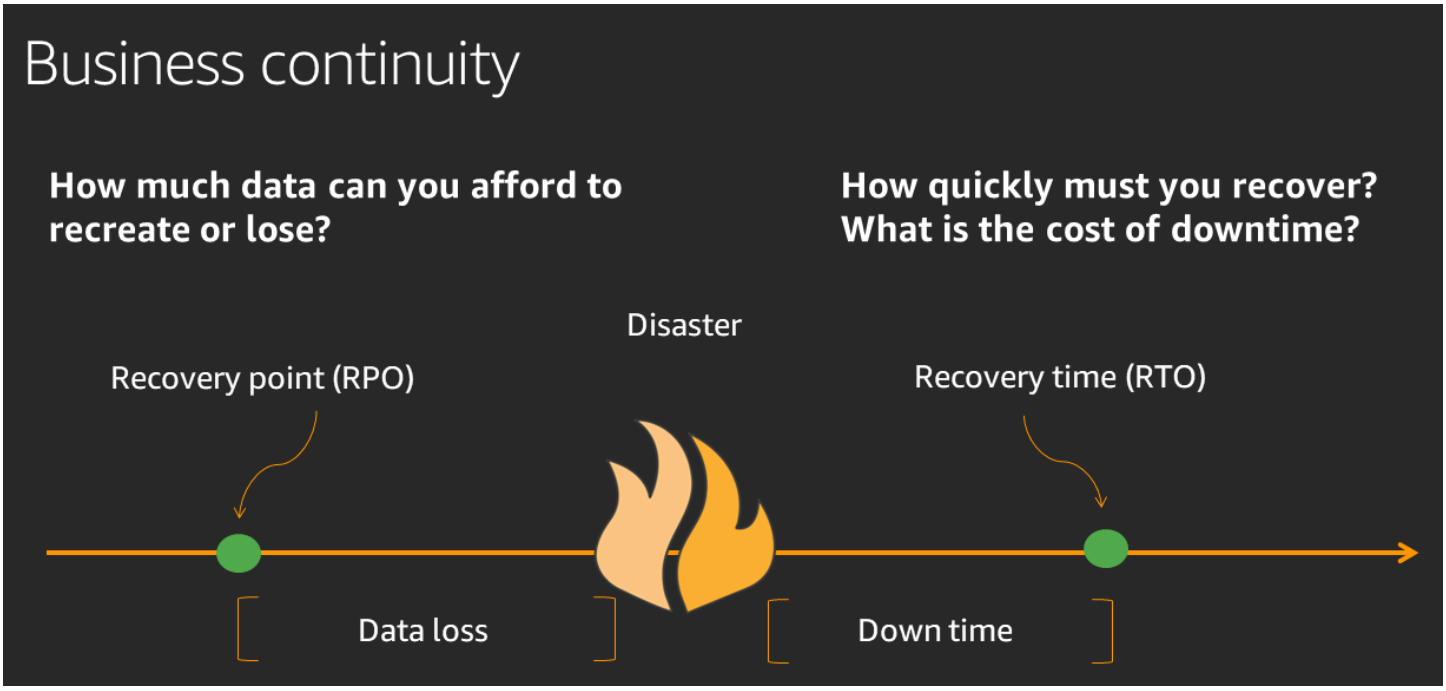
在具有更高可用性设计目标的系统中，成本增加的另一种方式与依赖项的选择有关。在目标较高的情况下，可以选择作为依赖项的软件或服务集减少，具体取决于其中哪些服务已具备我们前面所说的深度投资。随着可用性设计目标的增加，通常要少找一些多用途服务（例如关系数据库），多找一些专用服务。这是因为后者更易于评估、测试和自动化，与包括在内但未使用的功能发生意外交互的可能性也较低。

灾难恢复（DR）目标

除了可用性目标之外，韧性策略还应包括灾难恢复（DR）目标，这些目标基于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，可应对自然灾害、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韧性。

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可接受的最大延迟。这决定了当服务不可用时，什么时间段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可接受的最长时间。这决定了从上一个恢复点到服务中断之间可接受的数据丢失情况。



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。

建议

批判性地评估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来反映业务需求。

在 AWS，我们通常会将服务分为“数据面板”和“控制面板”。数据面板负责交付实时服务，控制面板则用于配置环境。例如，Amazon EC2 实例、Amazon RDS 数据库和 Amazon DynamoDB 表的读/写操作都是数据面板操作。而启动新的 EC2 实例或 RDS 数据库，或者在 DynamoDB 中添加或更改表元数据，都属于控制面板操作。虽然高水平的可用性对所有这些功能来说都很重要，但数据面板的可用性设计目标通常比控制面板更高。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 参与进来，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 和可用性来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了解该框架，了解工作负载对业务需求的影响。

基本原理

基本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，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，由于存在依赖项，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。

以下各节旨在介绍此类可靠性注意事项的最佳实践。

主题

管理服务配额和限制

对于基于云的工作负载架构，存在服务配额（也称为服务限制）。这些限额的存在是为了防止意外调配超出所需的资源，并限制 API 操作的请求率，以保护服务不被滥用。还存在资源限制，例如光缆的比特传输速率或物理磁盘的存储量。

如果您使用的是 AWS Marketplace 应用程序，则必须了解应用程序的限制。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 了解服务配额和约束

了解工作负载架构的默认配额并管理配额提高请求。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：客户可以通过实施适当的指导方针来监控关键指标、基础设施审查和自动化修复步骤，验证是否未达到可能导致服务性能下降或中断的服务配额和约束，从而防止其 AWS 账户 中的服务性能下降或中断。

常见反模式：

- 在不了解所用服务的硬配额或软配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配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。
- 假设配额会自动增加。
- 不了解配额请求的流程和时间表。
- 假设每个服务的默认云服务配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，不评测新流量水平的容量要求。

建立此最佳实践的好处：对服务配额及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性能下降。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韧性。

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

服务配额是一项 AWS 服务，可帮助您从一个位置管理超过 250 项 AWS 服务的配额。除了查找配额值，您还可以通过服务配额控制台或 AWS SDK 请求提高配额并跟踪配额。AWS Trusted Advisor 提供的服务配额检查功能会显示服务使用情况，以及某些服务在某些方面的配额。有关每项服务的默认服务配额，请查看相应服务的 AWS 文档（例如参阅 Amazon VPC 配额）。

通过配置使用计划，可在 Amazon API Gateway 内设置某些服务限制，例如节流 API 的速率限制。可通过配置相应服务进行设置的部分限制包括预调配 IOPS、已分配的 Amazon RDS 存储，以及 Amazon EBS 卷分配。Amazon Elastic Compute Cloud 有自己的服务限制控制面板，可帮助您管理实例、Amazon Elastic Block Store 和弹性 IP 地址限制。如果在某应用场景中，服务配额会对应用程序的性能造成影响，而且无法按需求进行调整，请联系 支持 了解是否有解决的办法。

服务配额可以是区域性的，也可以是全局性的。使用达到配额的 AWS 服务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性能下降。例如，服务配额限制了一个区域中使用的 DL Amazon EC2 实例的数量。在使用自动扩缩组（ASG）的流量扩缩事件期间，可能会达到该限制。

应定期评测每个账户服务配额的使用情况，确定适合该账户的适当服务限制。这些服务配额作为操作防护机制存在，可防止意外地配置超出所需数量的资源；也用于限制 API 操作的请求率，保护服务不被滥用。

服务约束与服务配额不同。服务约束代表由特定资源类型定义的该资源的限制，这些可能是存储容量（例如，gp2 的大小限制为 1 GB - 16 TB）或磁盘吞吐量。必须对资源类型的约束进行设计，并不断评测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。

如果在某应用场景中，服务配额对应用程序的性能造成影响，而且无法根据必要需求进行调整，请联系支持 了解是否有解决办法。有关调整固定配额的更多详细信息，请参阅 REL01-BP03 通过架构适应固定服务配额和约束。

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

- AWS Trusted Advisor 提供的服务配额检查功能会显示服务使用情况，以及某些服务在某些方面的配额。该工具可以帮助识别接近配额的服务。
- AWS 管理控制台 提供了显示服务配额值，管理、请求新配额，监视配额请求状态以及显示配额历史记录的方法。
- AWS CLI 和 CDK 提供了通过编程方式自动管理和监视服务配额水平和使用情况的方法。

实施步骤

服务配额：

- 请查看 AWS Service Quotas。
- 要了解现有的服务配额，请先确定使用的服务（如 IAM Access Analyzer）。大约有 250 项 AWS 服务由服务配额控制。然后，确定每个账户和区域内可能使用的具体服务配额名称。每个区域大约有 3000 个服务配额名称。

- 使用 AWS Config 来增强此配额分析，找出 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。查看通过 AWS 管理控制台 或 `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。
- 确定适用的服务配额。通过 Trusted Advisor 和服务配额使用能够以编程方式访问的信息。
- 建立自动化的监控方法（参见 REL01-BP02 跨多个账户和区域管理服务配额和 REL01-BP04 监控和管理配额），以便在服务配额接近或已达到限制时发出提醒和通知。
- 建立自动化的程序化方法，用于检查同一账户内某项服务配额是否在某一区域发生变更，但在其他地区未发生变更（参见 REL01-BP02 跨多个账户和区域管理服务配额和 REL01-BP04 监控和管理配额）。
- 自动扫描应用程序日志和指标，确定是否存在任何配额或服务约束错误。如果存在这些错误，则向监控系统发送警报。
- 确定特定服务需要更高的配额后，制定工程设计步骤来计算所需的配额变动（参见 REL01-BP05 自动管理配额）。
- 创建一个配置和批准工作流，以请求更改服务配额。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），审查服务配额。

服务约束：

- 建立监视和度量方法，用于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程序，可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，可验证新类型是否为新约束条件下的正确资源类型。
- 使用现有的程序和流程，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配额
- REL01-BP03 通过架构适应固定服务配额和约束

- REL01-BP04 监控和管理配额
- REL01-BP05 自动管理配额
- REL01-BP06 确保当前配额与最大使用量之间存在足够的差距来应对失效转移
- 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-BP04 使用混沌工程测试韧性

相关文档：

- AWS Well-Archited Framework 的可靠性支柱：可用性
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas User Guide
- 适用于 AWS 的配额监控程序
- AWS Fault Isolation Boundaries
- Availability with redundancy
- 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
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads

- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo

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配额

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配额。

期望结果：对于跨账户或区域的配置，或者具有使用区域或账户失效转移的韧性设计的配置，服务和应用程序不应受到服务配额耗尽的影响。

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配额。
- 没有考虑到在非主要区域出现性能下降期间，韧性架构（如主动或被动）对未来配额需求的影响。
- 没有定期评估配额，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。
- 不利用配额请求模板在多个地区和账户中请求提高配额。
- 不更新服务配额，因为错误地认为提高配额会像计算预留请求一样产生成本影响。

建立此最佳实践的好处：验证是否可以在区域服务不可用的情况下处理辅助区域或账户中的当前负载。这可以帮助降低在区域丢失期间发生的错误数量或性能下降水平。

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

每个账户的服务配额都可被跟踪。除非另有说明，否则每个配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配额，避免妨碍测试与开发。保持高度韧性需要持续评测服务配额（无论是自动还是手动）。

由于使用主动/主动、主动/被动 – 热、主动/被动 – 冷以及主动/被动 – 指示灯方法实施设计，跨区域的工作负载越来越多，因此了解所有区域和账户配额级别至关重要。如果服务配额设置正确，过去的流量模式并不总是一个好的指标。

同样重要的是，服务配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配额，以便在负载状态下提供一致的韧性。

协调不同区域（主动区域或被动区域）之间的所有服务配额差异，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 GameDay 演练或桌面演练可能无法发现区域之间的服务配额差异，因此也无法保持正确的限制。

服务配额偏差，即某一特定命名配额的服务配额限制在一个区域发生变更但未在所有区域发生变更的情况，这对于跟踪和评测非常重要。应考虑更改在有流量或可能有流量的区域的配额。

- 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配额。限制的范围具体到账户和区域。应对这些值进行比较，了解差异情况。

实施步骤

- 审查可能已经超出风险使用水平的服务配额值。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）的服务配额值。验证在主区域发生故障时，负载能否在辅助区域成功运行。
- 自动评测同一账户的不同区域之间是否发生了任何服务配额偏移，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，则应更新服务配额模板，反映应该应用于多个区域和账户的任何配额的变化。
  - 创建模板并将区域与配额变化关联起来。
  - 审查所有现有的服务配额模板，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 了解服务配额和约束
- REL01-BP03 通过架构适应固定服务配额和约束
- REL01-BP04 监控和管理配额
- REL01-BP05 自动管理配额
- REL01-BP06 确保当前配额与最大使用量之间存在足够的差距来应对失效转移
- 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-BP04 使用混沌工程测试韧性

相关文档：

- AWS Well-Archited Framework 的可靠性支柱：可用性
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits

- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas User Guide
- 适用于 AWS 的配额监控程序
- AWS Fault Isolation Boundaries
- Availability with redundancy
- 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
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo

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配额和约束

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

示例包括网络带宽、无服务器函数调用有效负载大小、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流量和高流量条件下按预期执行。它们被设计为在相应资源的固定约束或服务配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约束，这些约束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配额。例如，以突发限制运行测试，但运行时间较长。
- 选择的设计在超过固定服务配额时无法扩展或修改。例如，SQS 有效负载大小为 256 KB。
- 没有设计和实施可观测性，无法监控在高流量事件期间可能面临风险的服务配额阈值并发出警报。

建立此最佳实践的好处：确认应用程序会在所有预计的服务负载水平下运行，不会出现中断或性能下降。

在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

与软服务配额或替换为更高容量单位的资源不同，无法更改 AWS 服务的固定配额。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。

服务配额控制台中显示了硬限制。如果列显示 ADJUSTABLE = No，表示服务存在硬限制。一些资源配置页中也显示了硬限制。像是 Lambda 就具有无法调整的特定硬限制。

例如，在设计要在 Lambda 函数中运行的 Python 应用程序时，应评估应用程序，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配额限制，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，则应用程序将遭受性能下降和中断，直到可以补救为止。与软配额不同，即使出现严重性为 1 的紧急事件，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配额限制和硬配额限制。并非所有限制都会在服务配额控制台中显示。有些服务在备用位置描述了这些限制。
- 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、应用场景、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。
- 分析韧性架构，了解在分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应用场景。

资源

相关最佳实践：

- REL01-BP01 了解服务配额和约束
- REL01-BP02 跨多个账户和区域管理服务配额
- REL01-BP04 监控和管理配额
- REL01-BP05 自动管理配额
- REL01-BP06 确保当前配额与最大使用量之间存在足够的差距来应对失效转移
- 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-BP04 使用混沌工程测试韧性

相关文档：

- AWS Well-Archited Framework 的可靠性支柱：可用性
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas User Guide
- 适用于 AWS 的配额监控程序
- AWS Fault Isolation Boundaries
- Availability with redundancy
- 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
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

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 监控和管理配额

评估可能的使用情况，并适当提高配额，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配额使用阈值。根据请求的配额更改主动修复这些问题。

常见反模式：

- 没有配置监控来检查服务配额阈值。
- 没有为硬限制配置监控，即使这些值不能更改。
- 假定请求和确立软配额变化所需的时间是即时或短时间。
- 配置警报在快达到服务配额时发出警报，但没有关于如何对警报做出响应的流程。
- 只为 AWS 服务配额支持的服务配置警报，不监控其他 AWS 服务。
- 不考虑多区域韧性设计（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）的配额管理。
- 不评测区域之间的配额差异。
- 不评测每个区域对特定配额提高请求的需求。
- 不利用模板进行多区域配额管理。

建立此最佳实践的好处：自动跟踪 AWS 服务配额，并根据这些配额监控使用情况，以便了解何时会达到配额限制。您还可以使用此监控数据帮助限制由于配额耗尽而导致的性能下降。

在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

对于支持的服务，您可以配置各种能进行评测的不同服务，再通过发送提醒或警报来监控配额。这有助于监控使用情况，并可在接近配额时发出提醒。这些警报可以从 AWS Config、Lambda 函数、Amazon CloudWatch 或 AWS Trusted Advisor 调用。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配额阈值。

实施步骤

监控：

- 获取当前资源使用情况（例如存储桶或实例）。使用 Amazon EC2 `DescribeInstances` API 等服务 API 操作来收集当前资源使用情况信息。
- 使用以下资源获得必要且适用于服务的当前配额：
  - AWS 服务限额
  - 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)
- 使用 AWS 服务配额（一项 AWS 服务），帮助您从一个地方管理超过 250 项 AWS 服务的配额。
- 使用 Trusted Advisor 服务限制来监控在各种阈值下的当前服务限制。
- 使用服务配额历史记录（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配额变化，形成等效关系。

管理：

- 自动：设置 AWS Config 自定义规则以扫描各个区域的服务配额，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配额，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配额，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配额差异，则根据需要请求更改配额。
- 检查所有请求的结果。

资源

相关最佳实践：

- REL01-BP01 了解服务配额和约束
- REL01-BP02 跨多个账户和区域管理服务配额
- REL01-BP03 通过架构适应固定服务配额和约束
- REL01-BP05 自动管理配额
- REL01-BP06 确保当前配额与最大使用量之间存在足够的差距来应对失效转移
- 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-BP04 使用混沌工程测试韧性

相关文档：

- AWS Well-Archited Framework 的可靠性支柱：可用性
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas User Guide
- 适用于 AWS 的配额监控程序
- AWS Fault Isolation Boundaries

- Availability with redundancy
- 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
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

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 自动管理配额

服务配额（在 AWS 服务中也称为限制）是您的 AWS 账户中资源的最大值。每项 AWS 服务都定义了一组配额及其默认值。为了让工作负载能够访问它所需的所有资源，您可能需要增加服务配额值。

如果超过配额，AWS 资源的工作负载消耗的增长可能会威胁工作负载的稳定性，并影响用户体验。实施相应的工具，以便在工作负载接近限制时发出警报，并考虑自动创建增加配额的请求。

期望结果：为在每个 AWS 账户和区域中运行的工作负载适当配置了配额。

常见反模式：

- 您未能适当考虑和调整配额来满足工作负载要求。
- 您使用可能过时的方法（例如电子表格）来跟踪配额和使用情况。
- 您只按定期计划更新服务限制。
- 您的组织缺乏操作流程，无法查看现有配额并在必要时请求增加服务配额。

建立此最佳实践的好处：

- 增强了工作负载韧性：您可以防止因超出 AWS 资源配额而导致的错误。
- 简化了灾难恢复：在另一个 AWS 区域中进行灾难恢复设置时，您可以重用在主区域中构建的自动配额管理机制。

在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

通过 AWS 服务配额控制台、AWS Command Line Interface（AWS CLI）和 AWS 等机制，查看当前配额并跟踪正在进行的配额消耗。还可以将配置管理数据库（CMDB）和 IT 服务管理（ITSM）系统与 AWS 服务配额 API 集成。

如果配额使用量达到您定义的阈值，则自动生成警报，并定义在收到警报时提交配额增加请求的过程。如果底层工作负载对您的业务至关重要，则可以自动提出配额增加请求，但要仔细测试此项自动功能，以避免出现失控操作的风险，例如增长反馈循环。

较小的配额增加通常会自动获得批准。较大的配额请求可能需要由 AWS 支持人员手动处理，可能需要更多时间来审核和处理。留出额外的时间来处理多个请求或大幅增加配额的请求。

实施步骤

- 对服务配额实施自动监控，并在工作负载的资源利用率增长接近配额限制时发出警报。例如，AWS 的 Quota Monitor 可以提供对服务配额的自动监控。此工具与 AWS Organizations 集成，并使用 Cloudformation StackSets 进行部署，以便在创建新账户时自动进行监控。
- 使用 Service Quotas request templates 或 AWS Control Tower 等功能简化新账户的服务配额设置。
- 构建控制面板来显示您当前在所有 AWS 账户和区域的服务配额使用情况，并在必要时参考这些控制面板来防止超过配额。Trusted Advisor Organizational (TAO) Dashboard 是 Cloud Intelligence Dashboards 的一部分，可让您快速开始使用此类控制面板。
- 跟踪服务限制提高请求。Consolidated Insights from Multiple Accounts(CIMA) 可以提供所有请求的组织级视图。
- 通过在非生产账户中设置较低的配额阈值，测试警报生成和任何自动发出配额增加请求的功能。请勿在生产账户中进行这些测试。

资源

相关最佳实践：

- OPS10-BP07 自动响应事件

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：可帮助跟踪限制的 CMDB 产品
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- 适用于 AWS 的配额监控程序解决方案 - AWS 解决方案
- What is Service Quotas?
- What is Service Quotas request templates?

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

相关工具：

- Quota Monitor for AWS

REL01-BP06 确保当前配额与最大使用量之间存在足够的差距来应对失效转移

本文介绍了如何在资源配额和使用量之间保留一定空间，以及这如何让组织受益。使用完资源后，该资源的使用量配额可能会继续计入该资源。这可能导致资源出现故障或无法访问。确认配额涵盖无法访问的资源及其替换资源的重叠部分，避免资源出现故障。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应用场景。

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配额，而不考虑失效转移场景。
- 在计算服务的峰值配额时不考虑静态稳定性原则。
- 在计算每个区域所需的总配额时，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，使用云实施相关策略，以便从这些事件中恢复。此类策略通常包括创建额外资源来替换无法访问的资源，在不耗尽服务限制的情况下应对失效转移条件。

在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

在评估配额限制时，考虑因性能下降可能导致的失效转移情况。考虑以下失效转移情况：

- VPC 中断或无法访问。
- 子网无法访问。
- 可用区性能下降，影响资源的可访问性。
- 系统阻止或更改了网络路由或入口点和出口点。
- 区域性能下降，影响资源的可访问性。

- 区域或可用区出现故障，影响资源子集。

因为业务影响可能会有很大差异，每种情况的失效转移决策都是独特的。在决定对应用程序或服务进行失效转移之前，请先考虑失效转移位置的资源容量规划和资源配额。

在检查每项服务的配额时，要考虑到高于正常水平的活动峰值。这些峰值可能与因网络或权限而无法访问但仍处于活动状态的资源有关。未终止的活动资源将计入服务配额限制。

实施步骤

- 在服务配额与最高使用量之间保留一定空间，以便应对失效转移或失去可访问性。
- 确定服务配额。要考虑典型的部署模式、可用性要求和使用量增长情况。
- 根据需要请求提高配额。预计配额提高请求需要等待的一段时间。
- 确定可靠性要求（也称为“X 个 9”）。
- 了解潜在的故障情况，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）。
- 在当前配额限制中包含适当的缓冲区。例如设置 15% 的缓冲区。
- 在适当情况下包括静态稳定性（可用区和区域）的计算。
- 预计使用量增长情况并监控使用量趋势。
- 考虑静态稳定性对最关键工作负载的影响。评测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。对于关键的业务计划来说，这是一种值得实施的有用策略，可以在失效转移期间获得正确数量和类型的资源，从而降低潜在的风险。

资源

相关最佳实践：

- REL01-BP01 了解服务配额和约束
- REL01-BP02 跨多个账户和区域管理服务配额
- REL01-BP03 通过架构适应固定服务配额和约束
- REL01-BP04 监控和管理配额
- REL01-BP05 自动管理配额
- 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-BP04 使用混沌工程测试韧性

相关文档：

- AWS Well-Archited Framework 的可靠性支柱：可用性
- AWS Service Quotas (formerly referred to as service limits)
- AWS Trusted Advisor Best Practice Checks（参见“Service limits”小节）
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas User Guide
- 适用于 AWS 的配额监控程序
- AWS Fault Isolation Boundaries
- Availability with redundancy
- 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
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

相关视频：

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

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（可公开访问和私有），可能还包括您现有的数据中心基础架构。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。

通过 Amazon Virtual Private Cloud（Amazon VPC），您可以预置 AWS 云的私有隔离部分，并在其中启动虚拟网络中的 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、构建和实施高度可用的网络连接至关重要。如果工作负载因连接中断而无法访问，即使工作负载正在运行且可用，客户也会看到系统处于关闭状态。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 和弹性负载均衡（ELB）都提供了高度可用的公有端点。Amazon Route 53 为域名解析提供了高度可用的 DNS 服务，可确认公共端点地址是否可以解析。

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。
- 没有测试和规划与工作负载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：设计具有韧性的高可用性网络连接，确保用户可以访问和使用工作负载。

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 等高度可用且可扩展的域名系统（DNS）来管理域名的 DNS 记录。您还可以使用 Amazon Route 53 提供的运行状况检查。运行状况检查会确认应用程序可访问、可用且正常运行，并且支持以模仿用户行为的方式进行设置，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状况良好的端点。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。

要验证工作负载本身是否具有高可用性，请使用弹性负载均衡（ELB）。Amazon Route 53 可用于将流量定向到 ELB，而 ELB 会将流量分配到目标计算实例。您还可以将 Amazon API Gateway 与 AWS Lambda 结合使用，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，工作负载可以处理来自多个区域的流量。在多站点主动/被动模式下，工作负载为来自主动区域的流量提供服务，同时将数据复制到辅助区域，并在主区域出现故障时变为主动区域。然后，可以使用 Route 53 运行状况检查来控制从主区域中的任何端点到辅助区域的端点的 DNS 故障转移，从而验证工作负载是否可以访问并可供用户使用。

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。随着内容负载从服务器转移到 CloudFront 的边缘站点，这也提高了应用程序的可用性。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，都会允许用户流量重定向到运行状况良好的端点，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。

为了帮助保护客户免受 DDoS 攻击，AWS 提供 AWS Shield Standard。Shield Standard 会自动开启，可防范 SYN/UDP 泛洪和反射攻击等常见基础设施（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。要针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，以及要帮助保护 Amazon Elastic Compute Cloud（Amazon EC2）、弹性负载均衡（ELB）、Amazon CloudFront、AWS Global Accelerator 和 Route 53 上运行的应用程序，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. 设置高度可用的 DNS：Amazon Route 53 是一种可用性高、可扩展性强的域名系统（DNS）Web 服务。Route 53 会将用户请求连接到在 AWS 或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，确认只有运行状况良好的目标才可解析。首先创建 Route 53 运行状况检查并配置 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取通知
   d. Amazon Route 53 DNS 的最佳实践
3. 将 DNS 服务连接到端点。
   a. 使用弹性负载均衡作为流量目标时，请使用 Amazon Route 53 创建指向负载均衡器区域端点的别名记录。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. 对于使用 API Gateway 时的无服务器工作负载或私有 API，请使用 Route 53 将流量路由到 API Gateway。
4. 决定内容分发网络。
   a. 要使用离用户更近的边缘站点分发内容，首先要了解 CloudFront 如何分发内容。
   b. 开始使用简单 CloudFront 分发。然后，CloudFront 知道您希望从何处分发内容，还知道有关如何跟踪和管理内容分发的详细信息。在设置 CloudFront 分发时，必须了解和考虑以下方面：
      i. 缓存如何用于 CloudFront 边缘站点
      ii. 增加直接从 CloudFront 缓存提供服务的请求的比例（缓存命中率）
      iii. 使用 Amazon CloudFront Origin Shield
      iv. 通过 CloudFront 源失效转移来优化高可用性
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。若要加深理解，请查看 How AWS WAF works；若准备好实施针对应用层 HTTP POST AND GET 泛洪的保护，请查看 Getting started with AWS WAF。要将 AWS WAF 与 CloudFront 结合使用，请参阅有关 How AWS WAF works with Amazon CloudFront features 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。要进一步

保护在 Amazon EC2、弹性负载均衡、Amazon CloudFront、AWS Global Accelerator 和 Amazon Route 53 上运行的面向互联网的应用程序，可以考虑 AWS Shield Advanced 并查看 Examples of DDoS resilient architectures。要保护工作负载和公共端点免受 DDoS 攻击，请查看 Getting started with 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
- REL11-BP04 恢复期间依赖于数据面板而不是控制面板
- REL11-BP06 当事件影响可用性时发送通知

相关文档：

- APN 合作伙伴：可帮助规划联网的合作伙伴
- AWS Marketplace for Network Infrastructure
- 什么是 AWS Global Accelerator？
- What is Amazon CloudFront?
- What is Amazon Route 53?
- 什么是 Elastic Load Balancing？
- Network Connectivity capability - Establishing Your Cloud Foundations
- 什么是 Amazon API Gateway？
- What are AWS WAF, AWS Shield, and AWS Firewall Manager?
- What is Amazon Application Recovery Controller?
- 配置针对 DNS 故障转移的自定义运行状况检查

相关视频：

- AWS re:Invent 2022 - Improve performance and availability with AWS Global Accelerator
- AWS re:Invent 2020: Global traffic management with Amazon Route 53
- AWS re:Invent 2022 - Operating highly available Multi-AZ applications
- AWS re:Invent 2022 - Dive deep on AWS networking infrastructure

- AWS re:Invent 2022 - Building resilient networks

相关示例：

- Disaster Recovery with Amazon Application Recovery Controller (ARC)
- AWS Global Accelerator 讲习会

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实施冗余连接，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。

常见反模式：

- 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就能够可靠通信。

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。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本地位置，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，实施边界网关协议（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 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，请使用 AWS Transit Gateway 终止 VPN 连接。
- 评估 AWS Marketplace 设备以终止 VPN 或将 SD-WAN 扩展到 AWS。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 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
- Routing policies and BGP communities
- Active/Active and Active/Passive Configurations in AWS Direct Connect
- APN 合作伙伴：可帮助规划联网的合作伙伴

- AWS Marketplace for Network Infrastructure
- Amazon Virtual Private Cloud Connectivity Options 白皮书
- 构建可扩展的安全多 VPC AWS 网络基础结构
- Using redundant Site-to-Site VPN connections to provide failover
- Using the Direct Connect Resiliency Toolkit to get started
- VPC 端点和 VPC 端点服务（AWS PrivateLink）
- 什么是 Amazon VPC？
- What is a transit gateway?
- 什么是 AWS Site-to-Site VPN？
- Working with Direct Connect gateways

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，能够满足工作负载要求，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 IP 地址空间足以满足可能使用的任何 Amazon EC2 实例临时性队列的需求，如适用于机器学习的竞价型实例集、Amazon EMR 集群或 Amazon Redshift 集群。若是 Amazon Elastic Kubernetes Service（Amazon EKS）等 Kubernetes 集群，也应考虑这些因素。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，无法供您使用。
- 注意，最初被分配到您 VPC 的 VPC CIDR 块无法被更改或删除，但可以向 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。
- 考虑其他互联网络（VPC、本地部署或其他云提供商），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，难以与其他网络建立对等连接。

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，并在纵向扩展过程中继续提供可用性。

在未建立这种最佳实践的情况下暴露的风险等级：中

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他服务的集成。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购置或私有网络连接可能难以实施。

- 根据您服务、延迟、法规和灾难恢复（DR）要求，选择相关 AWS 账户 和区域。
- 确定对区域 VPC 部署的需求。
- 确定 VPC 的大小。
  - 确定是否要部署多 VPC 连接。
    - What Is a Transit Gateway?
    - Single Region Multi-VPC Connectivity
  - 确定是否需要隔离网络来满足法规要求。
  - 使用适当大小的 CIDR 块创建 VPC，满足当前和未来的需求。
    - 如果增长预测不明朗，则可能需要偏向更大的 CIDR 块，降低未来重新配置的可能性
  - 在双堆栈 VPC 中，考虑对子网使用 IPv6 寻址。IPv6 非常适合用于包含大量临时实例集或临时容器集的私有子网，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：可帮助规划联网的合作伙伴
- AWS Marketplace for Network Infrastructure
- Amazon Virtual Private Cloud Connectivity Options 白皮书
- 多数据中心 HA 网络连接
- Single Region Multi-VPC Connectivity
- 什么是 Amazon VPC？
- 上的 IPv6AWS
- IPv6 on reference architectures
- Amazon Elastic Kubernetes Service launches IPv6 support
- Recommendations for your VPC - Classic Load Balancers
- Availability Zone subnets - Application Load Balancers
- Availability Zones - Network Load Balancers

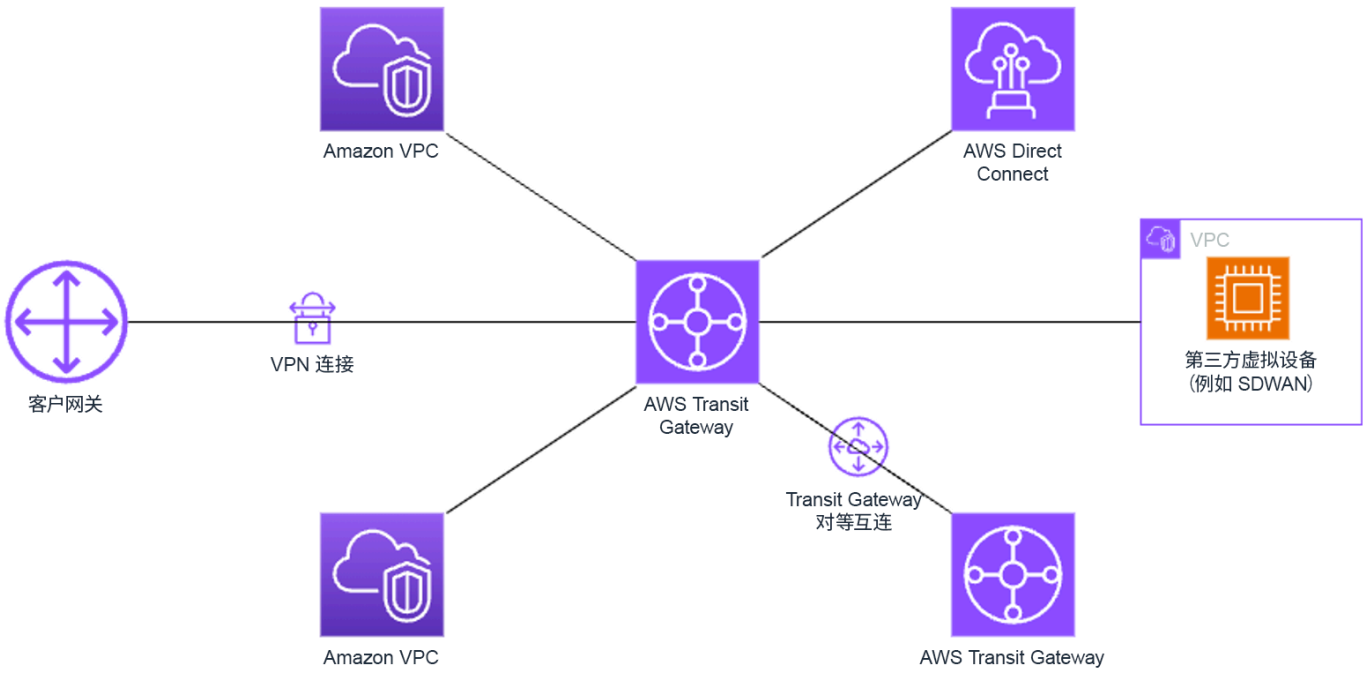
相关视频：

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC (NET303)
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs (NET406-R1)
- AWS re:Invent 2023: AWS Ready for what's next? Designing networks for growth and flexibility (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。



期望结果：您已通过中心枢纽连接虚拟私有云（VPC）和本地网络。可以通过此枢纽配置对等连接，该枢纽充当高度可扩展的云路由器。因为您不必处理复杂的对等关系，所以路由得以简化。网络之间的流量已加密，并且您可以隔离网络。

常见反模式：

- 您构建复杂的网络对等规则。
- 您在不应彼此通信的网络之间提供路由（例如，没有相互依赖关系的独立工作负载）。
- 枢纽实例的治理效率低下。

建立此最佳实践的好处：随着互联网络数量增加，网格连接的管理和扩展变得越来越有挑战性。网格架构会带来其它挑战，例如额外的基础设施组件、配置要求和部署注意事项。网格还为管理和监控数据面板和控制面板组件带来了额外的开销。您必须考虑如何提供网格架构的高可用性，如何监控网格运行状况和性能，以及如何处理网格组件的升级。

另一方面，轴辐式模型可在多个网络之间建立集中式流量路由。它提供了一种更简单的方法来管理和监控数据面板和控制面板组件。

在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

如果网络服务账户不存在，请创建一个此类账户。将枢纽置于组织的网络服务账户中。这种方法可让网络工程师对枢纽进行集中管理。

轴辐式模型的枢纽充当虚拟路由器，用于路由在虚拟私有云（VPC）和本地网络之间流动的流量。这种方法降低了网络复杂性，并且可以更轻松地对网络问题进行故障排除。

考虑您的网络设计，包括要互连的 VPC、AWS Direct Connect 和 Site-to-Site VPN 连接。

考虑为每个中转网关 VPC 连接使用单独的子网。对于每个子网，请使用小型 CIDR（例如 /28），以便您有更多地址空间用于计算资源。此外，创建一个网络 ACL，并将其与已和枢纽关联的所有子网相关联。确保网络 ACL 在入站和出站方向打开。

设计并实现路由表，以便仅在应进行通信的网络之间提供路由。忽略不应彼此通信的网络之间的路由（例如，在没有相互依赖关系的不同工作负载之间）。

实施步骤

1. 规划网络。确定要连接的网络，并确认它们不共享重叠的 CIDR 范围。
2. 创建 AWS Transit Gateway 并连接您的 VPC。
3. 根据需要，创建 VPN 连接或 Direct Connect 网关，并将其与 Transit Gateway 关联。
4. 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。
5. 使用 Amazon CloudWatch 进行监控并根据需要调整配置，从而优化性能和成本。

资源

相关最佳实践：

- REL02-BP03 确保 IP 子网分配考虑扩展和可用性
- 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：

- What Is a Transit Gateway?
- 中转网关设计最佳实践
- 构建可扩展的安全多 VPC AWS 网络基础结构

- Building a global network using AWS Transit Gateway Inter-Region peering
- Amazon Virtual Private Cloud Connectivity Options
- APN 合作伙伴：可帮助规划联网的合作伙伴
- AWS Marketplace for Network Infrastructure

相关视频：

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

相关讲习会：

- AWS Transit Gateway Workshop

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。避免 VPC 与本地环境之间或者与所使用的其他云提供商之间出现 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、本地环境或其他云提供商之间不存在 IP 地址范围冲突。
- 适当的 IP 地址管理支持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要求。

常见反模式：

- 在 VPC 中使用与本地、企业网络或者其他云提供商相同的 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多次出现相同 IP 地址的情况。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

实施指导

使用 IPAM（例如 Amazon VPC IP 地址管理器）来监控并管理 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 地址管理器来发现资源。
- 捕获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 地址管理器来发现资源。
- 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，也可以考虑在需要连接重叠范围时使用私有 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：

- APN 合作伙伴：可帮助规划联网的合作伙伴
- AWS Marketplace for Network Infrastructure
- 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

- 什么是 Amazon VPC？
- 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。您的架构选择会影响六个 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。

以下各节旨在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

设计工作负载服务架构

使用服务导向型架构（SOA）或微服务架构构建高度可扩展的可靠工作负载。服务导向型架构（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。

服务导向型架构（SOA）接口采用常见的通信标准，以便快速地合成到新的工作负载。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。

在 AWS，我们一直采用 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。应尽量避免使用整体架构，仔细考虑哪些应用程序组件可以分解为多项微服务。应用程序具体需求各异，架构到最后往往会将服务导向型架构（SOA）与微服务架构两者结合起来。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，并且尽量做到松耦合。

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：

- 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。

建立此最佳实践的好处：

- 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。

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

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 Microservice Trade-Offs。
- 如果工作负载适合，并且组织可以支持，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。
- 考虑按照 Strangler Fig 模式将整体重构为较小的组件。这包括使用新的应用程序和服务逐步替换特定的应用程序组件。AWS Migration Hub Refactor Spaces 充当增量重构的起点。有关更多详细信息，请参阅 Seamlessly migrate on-premises legacy workloads using a strangler pattern。
- 实施微服务可能需要服务发现机制，让这些分布式服务间能够相互通信。AWS App Mesh 可用于服务导向型架构，提供对服务的可靠发现和访问。AWS Cloud Map 也可以用于基于 DNS 的动态服务发现。
- 如果要从整体架构迁移到 SOA，Amazon MQ 可以在重新设计云中的传统应用程序时作为服务总线帮助缩小差距。
- 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？
- Bounded Context (a central pattern in Domain-Driven Design)
- 在 AWS 上实施微服务
- Microservice Trade-Offs
- Microservices - a definition of this new architectural term
- AWS 上的微服务
- 什么是 AWS App Mesh ？

相关示例：

- Iterative App Modernization 讲习会

相关视频：

- Delivering Excellence with Microservices on AWS

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。

期望结果：工程师和业务利益相关方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。现有整体式架构被分解为限界上下文，而系统设计则朝着服务导向型架构或微服务架构转变。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，服务满足领域特定的韧性要求。

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

领域驱动型设计（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，将应用程序改造为现代化服务。

领域驱动型设计

实施步骤

- 团队可以举办事件风暴讲习会，以简便的便签格式，快速确定事件、命令、聚合和领域。
- 在领域上下文中形成领域实体和功能后，您可以使用限界上下文将领域划分为服务，将具有相似特征和属性的实体分成一组。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。
- Decomposing monoliths into microservices 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，或建立防损层来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。
- 选择最符合业务领域要求和微服务架构的 AWS 服务：
    - AWS 上的无服务器服务让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，让您可以将精力集中于领域要求上。
    - 专用数据库有助于将领域要求与最适合的数据库类型相匹配。
- Building hexagonal architectures on AWS 中概述了一个框架，即从业务领域出发，采用逆向工作方法，将业务逻辑构建到服务中来满足功能要求，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
- How to break a Monolith into Microservices

- Getting Started with DDD when Surrounded by Legacy Systems
- Domain-Driven Design: Tackling Complexity in the Heart of Software
- Building hexagonal architectures on AWS
- Decomposing monoliths into microservices
- Event Storming
- Messages Between Bounded Contexts
- Microservices
- Test-driven development
- Behavior-driven development

相关示例：

- Designing Cloud Native Microservices on AWS (from 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。

期望结果：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。

常见反模式：

- 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，而不是按照域上下文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。

在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，从而将组件与强类型的 API 合同集成。

面向服务 API 合同的 AWS 服务

将包括 Amazon API Gateway、AWS AppSync 和 Amazon EventBridge 在内的 AWS 服务整合到架构中，以便在应用程序中使用 API 服务合同。Amazon API Gateway 有助于直接与原生 AWS 服务和其他网络服务集成。API Gateway 支持 OpenAPI 规范和版本控制。AWS AppSync 属于 GraphQL 托管端点，您可以通过定义 GraphQL 架构来配置该端点，定义用于查询、突变和订阅的服务接口。Amazon EventBridge 使用事件架构来定义事件，为事件生成代码绑定。

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。
- 您可以通过定义 GraphQL 架构文件，使用 AWS AppSync 来定义和管理 GraphQL API，从而生成合同接口，并简化与复杂 REST 模型、多个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。
  - 借助 API Gateway，您可以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
- Microservice Trade-Offs
- Microservices – a definition of this new architectural term
- AWS 上的微服务
- 使用基于 OpenAPI 的 API Gateway 扩展

- OpenAPI-Specification
- GraphQL: Schemas and Types
- Amazon EventBridge code bindings

相关示例：

- Amazon API Gateway：使用 OpenAPI 配置 REST API
- Amazon API Gateway to Amazon DynamoDB CRUD application using OpenAPI
- Modern application integration patterns in a serverless age: API Gateway Service Integration
- Implementing header-based API Gateway versioning with Amazon CloudFront
- AWS AppSync: Building a client application

相关视频：

- Using OpenAPI in AWS SAM to manage API Gateway

相关工具：

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

在分布式系统中设计交互来预防发生故障

分布式系统依靠通信网络来互连组件，例如服务器或服务。即使这些网络中出现数据丢失或延迟情况，您的工作负载也必须可靠运行。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。这些最佳实践可以防止故障并缩短平均故障间隔时间 (MTBF)。

最佳实践

- REL04-BP01 确定所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 使变异操作幂等

REL04-BP01 确定所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：设计能够与同步、异步和批处理依赖项进行有效交互的工作负载。

常见反模式：

- 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：对于给定的工作负载，通常能够在同步、异步和批处理之间实施一种或多种通信方式。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导和具体实施指导。

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 可观测性服务来监控响应时间和错误率，确保依赖项提供的服务达到工作负载所需的水平。
- 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。分布式系统面临着诸多挑战，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。

- 实施强大的错误处理和日志记录，在依赖项遇到问题时帮助排查问题。

同步依赖项

在同步通信中，工作负载会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状况不佳或不可用时，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。有关双模态行为的更多详细信息，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 开发人员指南》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。有关此主题的实用讨论，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

要想临时将工作负载与其依赖项解耦，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。消息收发可让工作负载通过消息代理发送和接收消息，从而与其依赖项进行通信。事件流可让工作负载及其依赖项利用流服务来发布和订阅以连续数据流形式交付且需尽快处理的事件。

- 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，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：消息代理确保使用者能够收到消息。事件流使用器必须跟踪所读取的每一条消息。
  - 消息排序：除非对消息收发使用先进先出（FIFO）方法，否则无法保证按照发送消息的确切顺序接收消息。事件流则始终保留数据生成的顺序。
  - 消息删除：使用消息收发时，使用者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，当工作负载异步调用 Lambda 函数时，Lambda 将请求置于队列中并返回成功响应，而不返回其他信息。处理完成后，Lambda 函数可以将结果发送到目标，该目标可根据成功或失败进行配置。
- 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，则消息收发或流服务将重新发送消息。
- 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，快速获得洞察来排查问题。

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。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。选择一种能让工作负载大规模读取和写入文件的存储服务，例如 Amazon Simple Storage Service（Amazon S3）、Amazon Elastic File System（Amazon EFS）和适用于 Lustre 的 Amazon FSx。
- 如果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 内部或本地运行的批处理作业的编排。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 之类的工具来监控 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 消息收发
- 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 Service（Amazon S3）
- Amazon Elastic File System (Amazon EFS)
- 适用于 Lustre 的 Amazon FSx

- 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。

解耦依赖关系（例如排队系统、流系统和工作流程），有助于最大限度地减少更改或故障对系统的影响。这种分离将组件的行为与依赖该组件的其他行为隔离开来，从而提高了韧性和敏捷性。

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，让您可以横向扩展或甚至改变依赖项的底层实施。

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 队列或是 Amazon Kinesis 或 AWS Step Functions 这样的就流数据平台。

图 4：队列系统和负载均衡器等依赖关系是松散耦合的

Amazon SQS 队列和 AWS Step Functions 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 Amazon EventBridge 可从其依赖的服务（事件使用器）中提取客户端（事件产生器）。如果需要进行高吞吐量、基于推送的多对多消息收发，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将消息扇出到大量订阅用户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。

期望结果：实施松耦合依赖项可以将故障的影响范围最大限度地缩小到组件级别，有助于诊断和解决问题。松耦合还简化了开发周期，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，在组件层面进行横向扩展。

常见反模式：

- 部署整体工作负载。
- 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。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 有助于构建松耦合和分布式的事件驱动架构。
- 在分布式系统中实施队列：可以使用 Amazon Simple Queue Service（Amazon SQS）集成或解耦分布式系统。
- 将组件容器化为微服务：微服务有助于团队构建由小型独立组件组成且通过明确定义的 API 进行通信的应用程序。Amazon Elastic Container Service（Amazon ECS）和 Amazon Elastic Kubernetes Service（Amazon EKS）有助于快速使用容器。
- 使用 Step Functions 管理工作流程：Step Functions 有助于将多个 AWS 服务协调为灵活的工作流程。
- 利用发布-订阅（pub/sub）消息收发架构：Amazon Simple Notification Service（Amazon SNS）提供从发布者到订阅用户（也称为生产者和使用者）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。
  - Building Event-driven Applications with Amazon EventBridge
  - AWS re:Invent 2022 – Designing Event-Driven Integrations using Amazon EventBridge
- 分布式消息收发系统主要有三个部分，需要为基于队列的架构实施这些部分。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。

  - Basic Amazon SQS architecture
  - 使用 Amazon Simple Queue Service 在分布式应用程序间发送消息
- 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
  - Let's Architect! Architecting microservices with containers
- 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，让您可以将应用程序中的依赖关系解耦。
  - 使用 AWS Step Functions 和 AWS Lambda 创建无服务器工作流程
  - AWS Step Functions 入门

资源

相关文档：

- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：Reliability, constant work, and a good cup of coffee
- 什么是 Amazon EventBridge？
- What Is Amazon Simple Queue Service?
- Break up with your monolith
- Orchestrate Queue-based Microservices with AWS Step Functions and Amazon SQS
- Basic Amazon SQS architecture
- Queue-Based Architecture

相关视频：

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)

- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)
- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda
- AWS re:Invent 2022 – Designing Event-Driven Integrations using Amazon EventBridge
- AWS re:Invent 2017: Elastic Load Balancing Deep Dive and Best Practices

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 10 万台服务器，在通常较低的服务器故障率下，它的负载是正常的。但如果发生重大事件让一半的服务器运行状况不佳，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都应发送当前状态的完整快照，10 万台服务器的运行状况状态（每个状态都用一位表示）仅占 12.5 KB 的有效负载。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。这实际上就是 Amazon Route 53 处理对端点（例如 IP 地址）运行状况检查的方式，从而确定最终用户如何路由到这些端点。

在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，这样系统就不会在负载发生骤变时出现故障。
- 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。
  - Amazon Builders' Library：Reliability, constant work, and a good cup of coffee
  - AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes constant work)
    - 以监控 10 万台服务器的运行状况检查系统为例，对工作负载进行设计，确保无论成功或失败次数如何，有效负载大小都保持不变。

资源

相关文档：

- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：Reliability, constant work, and a good cup of coffee

相关视频：

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes constant work)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)

REL04-BP04 使变异操作幂等

幂等服务承诺每个请求只确切处理一次，因此发起多个相同请求与发起单个请求的效果相同。这使客户端可以更轻松地进行重试，而不必担心多次错误地处理请求。要执行此操作，客户端可以发出具有幂等性令牌的 API 请求，每当重复该请求时都会使用此令牌。幂等服务 API 使用令牌来返回响应，该响应与首次完成请求时返回的响应相同，即使系统的底层状态已经改变也是如此。

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）执行某项操作一次相对简单。难就难在要保证某项操作确切执行一次，从而使发出多个相同的请求和发出单个请求具有一样的效果。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，而不需要创建重复的记录或产生副作用。

期望结果：您有一个一致、有据可查且广泛采用的方法来确保跨所有组件和服务的幂等性。

常见反模式：

- 您不加选择地应用幂等性，即使不需要也是如此。
- 您引入过于复杂的逻辑来实现幂等性。
- 您使用时间戳作为幂等性的密钥。由于时钟偏差或多个客户端使用相同的时间戳来应用更改，这可能会导致不准确。
- 您存储整个有效载荷以保持幂等性。在这种方法中，您为每个请求保存完整的数据有效载荷，并对于每个新请求将其覆盖。这可能会降低性能并影响可扩展性。

- 您在不同服务之间以不一致的方式生成密钥。如果密钥不一致，服务可能无法识别重复的请求，从而导致意想不到的结果。

建立此最佳实践的好处：

- 提高了可扩展性：系统可以处理重试和重复的请求，而无需执行额外的逻辑或复杂的状态管理。
- 增强了可靠性：幂等性有助于服务以一致的方式处理多个相同的请求，从而降低意外副作用或重复记录的风险。这在分布式系统中尤其重要，在此类系统中，网络故障和重试很常见。
- 提高了数据一致性：由于同一个请求会产生相同的响应，因此幂等性有助于保持分布式系统间的数据一致性。这对于维护事务和操作的完整性至关重要。
- 错误处理：幂等性令牌使错误处理变得更加简单。如果客户端由于问题而未收到响应，则它可以使用相同的幂等性令牌安全地重新发送请求。
- 操作透明度：通过幂等性，可以更好地进行监控和日志记录。服务可以使用其幂等性令牌记录请求，这可以更轻松地跟踪和调试问题。
- 简化了 API 合约：它可以简化客户端和服务器端系统之间的合约，并减少对错误数据处理的担忧。

在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

在分布式系统中，至多（客户端仅发起一个请求）或至少（客户端持续发起请求直到确认成功）执行某项操作一次相对简单。然而，确切一次 实施此行为则具有挑战性。为了实现这一点，客户端应该为每个请求生成并提供幂等性令牌。

通过使用幂等性令牌，服务可以区分新请求和重复的请求。当服务收到带有幂等性令牌的请求时，它会检查该令牌是否已被使用。如果该令牌已被使用，则该服务会检索并返回存储的响应。如果令牌是新的，则服务会处理请求，将响应与令牌一起存储，然后返回响应。这种机制使所有响应都是幂等的，从而提高了分布式系统的可靠性和一致性。

幂等性也是事件驱动型架构的一项重要行为。这些架构通常由消息队列提供支持，例如 Amazon SQS、Amazon MQ、Amazon Kinesis Streams 或 Amazon Managed Streaming for Apache Kafka（MSK）。在某些情况下，只发布一次的消息可能会意外地传送多次。当发布者生成幂等性令牌并将其包含在消息中时，它要求对收到的任何重复消息的处理不会导致对同一消息执行重复的操作。使用者应跟踪收到的每个令牌，并忽略包含重复令牌的消息。

服务和使用者还应将收到的幂等性令牌传递给它调用的任何下游服务。处理链中的每个下游服务都同样负责确保实施幂等性，以避免多次处理消息的副作用。

实施步骤

1. 确定幂等操作

   确定哪些操作需要幂等性。这些操作通常包括 POST、PUT 和 DELETE HTTP 方法以及数据库插入、更新或删除操作。不变异状态的操作（例如只读查询）通常不需要幂等性，除非它们有副作用。

2. 使用唯一标识符

   在发送者发送的每个幂等操作请求中包含一个唯一的令牌，既可以直接在请求中，也可以作为其元数据的一部分（例如，HTTP 标头）。这可让接收者识别和处理重复的请求或操作。通常用于令牌的标识符包括 Universally Unique Identifiers (UUIDs) 和 K-Sortable Unique Identifiers (KSUIDs)。

3. 跟踪和管理状态

   维护工作负载中每个操作或请求的状态。这可以通过将幂等性令牌和相应的状态（例如待处理、已完成或失败）存储在数据库、缓存或其它持久存储中来实现。此状态信息可让工作负载识别和处理重复的请求或操作。

   如果需要，可通过使用适当的并发控制机制（例如锁定、事务或乐观并发控制）来保持一致性和原子性。这包括记录幂等令牌以及运行与为请求提供服务相关联的所有变异操作的过程。这有助于防止竞争条件并验证幂等操作是否正确运行。

   定期从数据存储中移除旧的幂等性令牌来管理存储和性能。如果存储系统支持此操作，请考虑对数据使用过期时间戳（通常称为生存时间或 TTL 值）。重用幂等性令牌的可能性会随时间推移而降低。

   通常用于存储幂等性令牌和相关状态的常见 AWS 存储选项包括：

   - Amazon DynamoDB：DynamoDB 是一项 NoSQL 数据库服务，可提供低延迟性能和高可用性，因此非常适合存储与幂等性相关的数据。DynamoDB 的键值和文档数据模型支持高效存储和检索幂等性令牌和关联的状态信息。如果应用程序在插入幂等性令牌时设置了 TTL 值，则 DynamoDB 也可以自动使这些令牌过期。
   - Amazon ElastiCache：ElastiCache 能够以高吞吐量、低延迟和低成本存储幂等性令牌。如果应用程序在插入幂等性令牌时设置了 TTL 值，ElastiCache（Redis）和 ElastiCache（Memcached）也可以自动使这些令牌过期。
   - Amazon Relational Database Service（RDS）：可以使用 Amazon RDS 存储幂等性令牌和相关状态信息，特别是在应用程序已经将关系数据库用于其它用途的情况下。
   - Amazon Simple Storage Service（S3）：Amazon S3 是一项高度可扩展和耐用的对象存储服务，可用于存储幂等性令牌和相关元数据。S3 的版本控制功能对于维护幂等操作的状态特别有

用。存储服务的选择通常取决于诸如因素，例如：与幂等性相关的数据量、所需的性能特征、对耐久性和可用性的需求，以及幂等性机制如何与整体工作负载架构集成。

4. 实施幂等操作

   将 API 和工作负载组件设计为幂等的。将幂等性检查纳入工作负载组件。在处理请求或执行操作之前，请检查是否已经处理了唯一标识符。如果已处理，则返回之前的结果，而不是再次执行该操作。例如，如果客户端发送一个创建用户的请求，请检查是否已存在具有相同唯一标识符的用户。如果该用户存在，则应返回现有用户信息，而不是创建新用户。同样，如果队列使用者收到带有重复幂等性令牌的消息，则该使用者应忽略该消息。

   创建全面的测试套件来验证请求的幂等性。它们应涵盖各种各样的场景，例如成功的请求、失败的请求和重复的请求。

   如果工作负载利用 AWS Lambda 函数，请考虑使用 Powertools for AWS Lambda。Powertools for AWS Lambda 是一个开发人员工具包，有助于在使用 AWS Lambda 函数时实施无服务器最佳实践并提高开发人员速度。特别是，它提供了一个实用程序，可将 Lambda 函数转换为可以安全重试的幂等操作。

5. 清晰地传达幂等性

   记录 API 和工作负载组件，以清晰地传达操作的幂等性质。这有助于客户了解预期行为以及如何可靠地与工作负载进行交互。

6. 监控和审计

   实施监控和审计机制，以检测与响应的幂等性相关的任何问题，例如意外的响应变化或过多的重复请求处理。这有助于您检测和调查工作负载中的任何问题或意外行为。

资源

相关最佳实践：

- REL05-BP03 控制与限制重试调用
- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP03 发送通知（实时处理和报警）
- REL08-BP02 将功能测试作为部署的一部分进行集成

相关文档：

- The Amazon Builders' Library: Making retries safe with idempotent APIs
- Amazon Builders' Library：分布式系统相关挑战
- Amazon Builders' Library：Reliability, constant work, and a good cup of coffee
- Amazon Elastic Container Service: Ensuring idempotency
- 如何让我的 Lambda 函数保持幂等性？
- Ensuring idempotency in Amazon EC2 API requests

相关视频：

- Building Distributed Applications with Event-driven Architecture - AWS Online Tech Talks
- AWS re:Invent 2023 - Building next-generation applications with event-driven architecture
- AWS re:Invent 2023 - Advanced integration patterns & trade-offs for loosely coupled systems
- AWS re:Invent 2023 - Advanced event-driven patterns with Amazon EventBridge
- AWS re:Invent 2018 - Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019 - Moving to event-driven architectures (SVS308)

相关工具：

- Idempotency with AWS Lambda Powertools (Java)
- Idempotency with AWS Lambda Powertools (Python)
- AWS Lambda Powertools GitHub page

在分布式系统中设计交互以减少或承受故障

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，并且降低此类损坏的影响。其结果是缩短平均恢复时间（MTTR）。

这些最佳实践可以防止故障并缩短平均故障间隔时间 (MTBF)。

最佳实践

REL05-BP01 实施优雅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。

期望结果：某个组件的依赖项运行状况不佳时，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：优雅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。

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

实施优雅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。

为优雅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。考虑这些依赖项可能出现什么样的故障。思考能在故障期间尽力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优雅降级：

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“实施指导”中的示例）。

2. 下游系统因高负载无法处理请求：如果对下游系统的请求持续失败，则继续重试就没有意义了。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映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，某些应用场景可能需要完整的审核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 Serverless 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：限制对 API 的请求以提高吞吐量
- CircuitBreaker (summarizes Circuit Breaker from “Release It!” book)
- Error Retries and Exponential Backoff in AWS
- Michael Nygard “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：超时、重试和抖动回退

相关视频：

- Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

REL05-BP02 限制请求

限制请求，防范因需求意外增加而导致的资源耗尽情况。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。

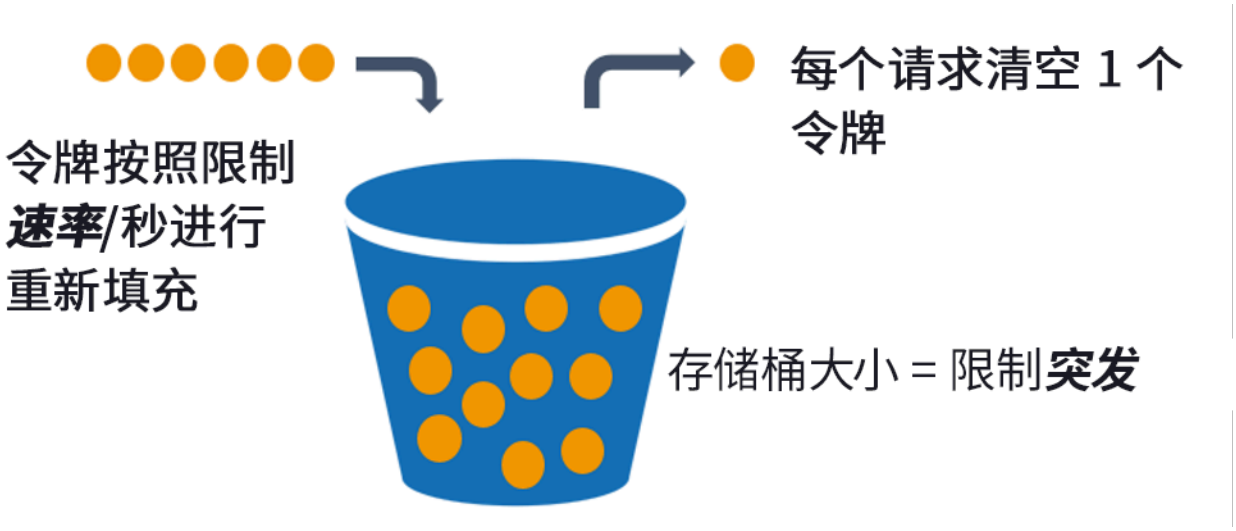
建立此最佳实践的好处：在遇到意外的容量峰值时，设置了节流限制的工作负载能够正常运行，并成功处理已接受的请求负载。API 和队列上突然或持续出现的请求峰值会受到限制，不会耗尽请求处理资源。速率限制会限制单独的请求者，这样来自单个 IP 地址或 API 使用者的大量流量就不会耗尽资源，从而不会影响其他使用者。

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

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 可以缓冲请求来稳定请求速率，并允许对可以处理的请求实施更高的节流速率。最后，您可以使用 AWS WAF 实施速率限制，限制产生异常高负载的特定 API 使用者。

实施步骤

您可以为 API 配置 API Gateway 节流限制，并在超过限制时返回 `429 Too Many Requests` 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，如果系统能够接受异步处理，则可以将消息放入队列或流中，借此加快对服务客户端的响应，这样便可以突增到更高的限制速率。

采用异步处理，在将 Amazon SQS 配置为 AWS Lambda 的事件源时，您可以配置最大并发数，避免高事件速率消耗工作负载或账户中其他服务所需的可用账户并发执行配额。

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）。

- 了解每个区域的账户级别、每个阶段的 API 和每个使用计划级别的 API 密钥的 API Gateway 节流限制，并进行配置。
- 对 API Gateway 和 AWS AppSync 端点应用 AWS WAF 速率限制规则，防范泛洪并阻止恶意 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，请考虑所需节流控制是否超过速率限制；如果超过，则在 AWS AppSync 端点前面配置 API Gateway。

- 在将 Amazon SQS 队列设置为 Lambda 队列使用者的触发器时，请将最大并发数设置为足以满足服务级别目标，但不会消耗会影响其他 Lambda 函数的并发限制的值。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可缓冲请求。
- 如果无法使用 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，然后自行研究找出合适的库。
- 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：限制对 API 的请求以提高吞吐量
- AWS WAF: Rate-based rule statement
- Introducing maximum concurrency of AWS Lambda when using Amazon SQS as an event source
- AWS Lambda: Maximum Concurrency

相关示例：

- The three most important AWS WAF rate-based rules
- Java Bucket4j
- Python token-bucket
- Node token-bucket
- .NET System Threading Rate Limiting

相关视频：

- Implementing GraphQL API security best practices with AWS AppSync

相关工具：

- Amazon API Gateway
- AWS AppSync
- Amazon SQS
- Amazon Kinesis
- AWS WAF
- AWS 上的虚拟等候室

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来随机调整重试间隔。限制最大重试次数。

期望结果：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，避免产生导致亚稳态故障的积压。

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

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。引入抖动来随机调整重试间隔，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。根据应用场景确定超时以及何时停止重试。为重试应用场景构建和演练测试场景。

实施步骤

- 对于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。
- 请先验证服务是否具有幂等性，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，请使用 AWS SDK 和 AWS CLI，并了解重试配置选项。确定默认值是否适用于应用场景，进行测试，并根据需要进行调整。

资源

相关最佳实践：

- REL04-BP04 使变异操作幂等

- 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：

- Error Retries and Exponential Backoff in AWS
- Amazon Builders' Library：超时、重试和抖动回退
- Exponential Backoff and Jitter
- Making retries safe with idempotent APIs

相关示例：

- Spring Retry
- Resilience4j Retry

相关视频：

- Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

相关工具：

- AWS SDKs and Tools: Retry behavior
- AWS Command Line Interface: AWS CLI retries

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。

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

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失效转移。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 检测工具来创建指标和警报。
- 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
- Fail Fast
- 如何防止我的 Amazon SQS 队列中日益积压的消息？
- Elastic Load Balancing: Zonal Shift

- Amazon Application Recovery Controller: Routing control for traffic failover

相关示例：

- Enterprise Integration Patterns: Dead Letter Channel

相关视频：

- AWS re:Invent 2022 – Operating highly available Multi-AZ applications

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。

客户端会根据配置的值连接超时。遇到超时后，客户端会决定回退并重试，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。

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

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，可轻松地处理超时并支持快速失效机制方法。AWSSDK 和 AWS CLI 允许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 函数支持超时

配置，所以借助 AWS Step Functions，您可以利用与 AWS 服务和 SDK 的预构建集成，以低代码方式构建断路器。AWS App MeshEnvoy 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 `connectTimeoutInMillis` 和 `tlsNegotiationTimeoutInMillis` 的 AWS 配置默认值来配置默认超时值。
- 应用命令行选项 `cli-connect-timeout` 和 `cli-read-timeout` 来控制对 AWS 服务的一次性 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。
- 实施对调用错误率、延迟的服务等级目标和延迟异常值的 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。
- 配置 Lambda 函数的超时时间。
- 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，并且不会在集成请求超时时重试。
- 实施断路器模式，避免在超时时进行远程调用。打开断路器避免调用失败，并在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，以便充分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，以此简化工作负载。

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: Retries and Timeouts
- Amazon Builders' Library：超时、重试和抖动回退
- Amazon API Gateway 配额和重要说明
- AWS Command Line Interface: Command line options
- AWS SDK for Java 2.x: Configure API Timeouts
- AWS Botocore using the config object and Config Reference
- 适用于 .NET 的 AWS SDK: Retries and Timeouts
- AWS Lambda：配置 Lambda 函数选项

相关示例：

- Using the circuit breaker pattern with AWS Step Functions and Amazon DynamoDB
- Martin Fowler: CircuitBreaker

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适合水平扩缩，因为可以根据流量和需求的波动情况增加或删除容量。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

实施指导

让应用程序无状态。无状态应用程序支持水平扩缩，并且可以承受单个节点故障。分析并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。无状态架构会解耦用户数据并分流会话数据。这为独立扩展每个组件提供了灵活性，可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。
  - 您可以通过将密钥存储在安全的集中位置来使用 AWS Secrets Manager 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。
  - 考虑采用 Amazon S3 存储大型非结构化数据，例如图像和文档。应用程序可以在需要时检索这些数据，无需将数据存储在内存中。
  - 使用 Amazon DynamoDB 存储用户资料等信息。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。

  - Amazon ElastiCache、Amazon DynamoDB、Amazon Elastic File System（Amazon EFS）和 Amazon MemoryDB 都是可用于分流会话数据的 AWS 服务示例。
- 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：通过实施应急杠杆，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，工作负载应进行优雅降级，并继续执行其关键业务功能。有关优雅降级的更多详细信息，请参阅 REL05-BP01 实施优雅降级以将适用的硬依赖关系转换为软依赖关系。

常见反模式：

- 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析器提供既定的流程来应对意外的需求激增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

实施指导

- 识别工作负载中的关键组件。
- 设计和构建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 Amazon 关键和非关键功能的示例，请参阅 Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second。
  - 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。
- 设计和构建工作负载中的关键组件，使其能够承受非关键组件的故障。
  - 在分析依赖项期间，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。
- 定义和监控相关指标并针对指标发出警报，以便启动紧急杠杆程序。
  - 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，或者是对依赖项请求失败的次数。
- 定义构成紧急杠杆的手动或自动程序。
  - 这可能包括卸除负载、限制请求或实施优雅降级等机制。

资源

相关最佳实践：

- REL05-BP01 实施优雅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：

- 自动实现无需干预的安全部署
- Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second

相关视频：

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

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运行。此类更改包括，外部因素对工作负载造成的更改（如需求高峰），以及内部因素造成的更改（如功能部署和安全补丁）。

以下各节旨在介绍变更管理的最佳实践。

主题

监控工作负载资源

日志和指标是深入了解工作负载运行状况的强大工具。您可以将工作负载配置为监控日志和指标，并在超过阈值或发生重大事件时发送通知。通过监控，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，从而自动恢复以做出响应。

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，即无法直接检测到功能已失效。该故障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进行监控的一个主要目的。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 在多个级别构建应用程序。我们会记录每个请求、所有依赖项和流程内关键操作的延迟、错误率和可用性，也会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。我们不仅会考虑平均延迟，还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，即使有一个的速度过慢，体验还是会变得非常糟糕。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 – 发送通知并自动执行响应
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（实时处理和警报）
- REL06-BP05 分析日志
- REL06-BP06 定期审核监控范围和指标
- 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），并为对应的警报事件设置阈值。确保指标与工作负载的关键性能指标（KPI）相关，并使用指标和日志来识别服务性能下降的早期预警信号。例如，与业务成果相关的指标（例如每分钟成功处理的订单数）可以比技术指标（例如 CPU 利用率）更快地表明工作负载问题。使用 AWS Health 控制面板可提供 AWS 资源底层的 AWS 服务性能和可用性的个性化视图。

云中监控创造新的机会。大多数云提供商都开发了可自定义的挂钩，可以提供见解来帮助您监控多层工作负载。Amazon CloudWatch 等 AWS 服务应用统计和机器学习算法来持续分析系统和应用程序的指标，只需最少的用户干预即可确定正常基线和表面异常。异常检测算法将指标的季节性变化和趋势变化考虑在内。

AWS 提供了大量的监控和日志信息，这些信息可用于定义工作负载特定的指标、需求变化流程，也助于采用机器学习技术，无论是否具备机器学习专业知识如何。

此外，还可监控所有外部端点，确保这些端点独立于基本实施。这种主动监控可通过综合事务实现（有时被称为用户金丝雀，但与金丝雀部署不同），这些事务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。确保这些任务的持续时间较短，不要让工作负载在测试期间过载。Amazon CloudWatch Synthetics 让您可以创建 Synthetics 金丝雀，以便对端点和 API 进行监控。您还可以整合 Synthetics 金丝雀客户端节点和 AWS X-Ray 控制台，精确定位哪些 Synthetics 金丝雀遇到错误、故障，或对指定时段的速率进行限制的问题。

期望结果：

收集并使用来自工作负载所有组件的关键指标，确保工作负载的可靠性和最佳的用户体验。若能检测到工作负载无法实现业务成果，可快速宣布发生灾难并从事件中恢复。

常见反模式：

- 仅监控连接到工作负载的外部接口。
- 不生成任何工作负载特定的指标，只依靠工作负载使用的 AWS 服务所提供的指标。
- 仅使用工作负载中的技术指标，而不监控与工作负载带来的非技术性 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：在工作负载的各个层级进行监控，便于更快地预测并解决构成工作负载的组件中的问题。

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

1. 启用日志记录（如适用）。应从工作负载的所有组件获取监控数据。启用其他日志记录（例如 S3 访问日志），让工作负载记录工作负载特定的数据。收集来自 Amazon ECS、Amazon EKS、Amazon EC2、弹性负载均衡、AWS Auto Scaling 和 Amazon EMR 等服务的 CPU、网络 I/O 和磁盘 I/O 平均值的指标。有关向 CloudWatch 发布指标的 AWS 服务列表，请参阅发布 CloudWatch 指标的 AWS 服务。
2. 审查所有默认指标并探究任何数据收集欠缺。每项服务都会生成默认指标。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对工作负载的影响。您可使用 AWS CLI 或 API 向 CloudWatch 发布自定义指标。
3. 评估所有指标，决定要针对工作负载中每项 AWS 服务的哪些指标发出警报。您可以选择对工作负载可靠性有重大影响的指标子集。关注关键指标和阈值有助于细化警报数量，也助于尽量减少误报。
4. 定义警报以及在调用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、上报和执行必要的步骤，以便从事件中恢复并达到规定的恢复时间目标（RTO）。您可以使用 Amazon CloudWatch 警报调用自动工作流，并根据定义的阈值启动恢复程序。
5. 探索使用综合事务来收集有关工作负载状态的相关数据。综合监控遵循相同的路线并执行与客户相同的操作，让您能够持续验证客户体验，即使应用程序中没有任何客户流量。使用综合事务，您可以早于客户先行发现问题。

资源

相关最佳实践：

- REL11-BP03 自动修复所有层

相关文档：

- Getting started with your AWS Health Dashboard – Your account health
- 发布 CloudWatch 指标的 AWS 服务
- Access Logs for Your Network Load Balancer
- Access logs for your application load balancer
- 访问 AWS Lambda 的 Amazon CloudWatch Logs
- Amazon S3 服务器访问日志记录
- Enable Access Logs for Your Classic Load Balancer
- Exporting log data to Amazon S3
- 在 Amazon EC2 实例上安装 CloudWatch 代理
- 发布自定义指标
- Using Amazon CloudWatch Dashboards
- 使用 Amazon CloudWatch 指标
- 使用金丝雀（Amazon CloudWatch Synthetics）
- What are Amazon CloudWatch Logs?

用户指南：

- 创建跟踪
- Monitoring memory and disk metrics for Amazon EC2 Linux instances
- 将 CloudWatch Logs 与容器实例结合使用
- VPC 流日志
- What is Amazon DevOps Guru?
- 什么是 AWS X-Ray？

相关博客：

- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray

相关示例：

- Amazon Builders' Library：检测分布式系统的运营可见性
- Observability 讲习会

REL06-BP02 定义与计算指标（聚合）

从工作负载组件收集指标和日志，并从中计算相关的聚合指标。这些指标为工作负载提供了广泛而深入的可观测性，并可以显著提高韧性态势。

可观测性不仅仅是从工作负载组件中收集指标以及能够查看指标和针对指标发出警报。其最终目的是对工作负载的行为进行全面的了解。此类行为信息来自工作负载中的所有组件，包括它们所依赖的云服务、精心制定的日志和指标。这些数据使您能够监督工作负载的整体行为，并可以非常详细地了解每个组件与每个工作单元的交互情况。

期望结果：

- 可以从工作负载组件和 AWS 服务依赖关系中收集日志，然后将其发布到一个便于访问和处理的中心位置。
- 日志包含高保真和准确的时间戳。
- 日志包含有关处理上下文的相关信息，例如跟踪标识符、用户或账户标识符以及远程 IP 地址。
- 可以从日志中创建聚合指标，这些指标从高层次视角表示工作负载的行为。
- 可以查询聚合的日志，以获得有关工作负载的深入和相关的见解，并确定实际和潜在的问题。

常见反模式：

- 您未从运行工作负载的计算实例或工作负载使用的云服务中收集相关日志或指标。
- 您忽略了与业务关键绩效指标（KPI）相关的日志和指标的收集。
- 您单独分析与工作负载相关的遥测数据，而没有采用聚合和关联。
- 您让指标和日志过快过期，这会阻碍趋势分析和识别反复出现的问题。

建立这些最佳实践的好处：您可以检测更多异常情况，并使工作负载的不同组件之间的事件和指标相关联。您可以根据日志中包含的信息，从工作负载组件中创建见解，而这些信息通常仅在指标中不可用。通过大规模查询日志，您可以更快地确定失败原因。

在未建立这些最佳实践的情况下暴露的风险等级：高

实施指导

确定与您的工作负载及其组件相关的遥测数据来源。这些数据不仅来自发布指标的组件，例如您的操作系统（OS）和应用程序运行时（例如 Java），还来自应用程序和云服务日志。例如，Web 服务器通常会记录每个请求以及诸如时间戳、处理延迟、用户 ID、远程 IP 地址、路径和查询字符串等详细信息。这些日志中的详细程度有助于您执行详细的查询，并生成原本可能无法得到的指标。

使用适当的工具和流程收集指标和日志。在 Amazon EC2 实例上运行的应用程序生成的日志可以由 Amazon CloudWatch 代理等代理收集，并发布到 Amazon CloudWatch Logs 等中央存储服务。AWS Lambda 和 Amazon Elastic Container Service 等 AWS 托管式计算服务会自动将日志发布到 CloudWatch Logs。为工作负载使用的 AWS 存储和处理服务启用日志收集，如 Amazon CloudFront、Amazon S3、弹性负载均衡和 Amazon API Gateway。

使用维度丰富遥测数据，维度有助于您更清楚地看到行为规律，并将相关问题隔离到相关组件组中。添加后，您可以更详细地观察组件行为，检测相关的故障，并采取适当的补救措施。有用维度的示例包括可用区、EC2 实例 ID 和容器任务或容器组（pod）ID。

收集指标和日志后，您可以编写查询并从中生成聚合指标，从而为正常和异常行为提供有用的见解。例如，您可以使用 Amazon CloudWatch Logs Insights 从应用程序日志中得出自定义指标，使用 Amazon CloudWatch Metrics Insights 大规模查询您的指标，使用 Amazon CloudWatch Container Insights 从容器化应用程序和微服务中收集、聚合和汇总指标和日志，或者，如果您使用的是 AWS Lambda 函数，则可以使用 Amazon CloudWatch Lambda 洞察。要创建聚合错误率指标，可以在每次在组件日志中发现错误响应或消息时递增计数器，或者计算现有错误率指标的聚合值。可以使用这些数据来生成显示尾部行为 的直方图，例如性能最差的请求或进程。还可以使用 CloudWatch Logs anomaly detection 等解决方案实时扫描这些数据，来发现异常规律。这些见解可以放在控制面板上，以便根据您的需求和偏好进行整理。

查询日志有助于您了解工作负载组件如何处理特定的请求，并揭示影响工作负载韧性的请求规律或其它上下文。根据您对应用程序和其它组件行为的了解，提前研究和准备查询可能很有用，这样您就可以更轻松地根据需要运行它们。例如，使用 CloudWatch Logs Insights，您能够以交互方式搜索和分析存储在 CloudWatch Logs 中的日志数据。还可以使用 Amazon Athena 查询来自多个来源（包括许多 AWS 服务）的日志，数据量可达 PB 级。

在定义日志保留策略时，请考虑历史日志的价值。历史日志有助于确定工作负载性能的长期使用情况和行为规律、回归以及改进领域。永久删除的日志以后无法分析。然而，历史日志的价值往往会随着时间推移而减少。选择的策略应能够适当平衡您的需求，并符合您可能需要遵守的任何法律或合同要求。

实施步骤

1. 为您的可观测性数据选择收集、存储、分析和显示机制。

2. 在工作负载的适当组件上安装和配置指标和日志收集器（例如，在 Amazon EC2 实例上和边车容器中）。将这些收集器配置为在意外停止时自动重新启动。为收集器启用磁盘或内存缓冲，这样，临时发布失败就不会影响应用程序或导致数据丢失。
3. 在您用作工作负载一部分的 AWS 服务上启用日志记录，并在需要时将这些日志转发到您选择的存储服务。有关详细说明，请参阅相应服务的用户或开发人员指南。
4. 定义与基于遥测数据的工作负载相关的操作指标。这些指标可能基于从工作负载组件发出的直接指标，其中可能包括与业务 KPI 相关的指标，也可能基于聚合计算的结果，例如总和、比率、百分位数或直方图。使用日志分析器计算这些指标，并根据需要将其放在控制面板上。
5. 根据需要准备相应的日志查询，来分析工作负载组件、请求或事务行为。
6. 为组件日志定义并启用日志保留策略。当日志的时间超过策略允许的时间时，定期删除日志。

资源

相关最佳实践：

- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP03 发送通知（实时处理和报警）
- REL06-BP04 自动响应（实时处理和警报）
- REL06-BP05 分析日志
- REL06-BP06 定期审核监控范围和指标
- REL06-BP07 对系统中的请求进行端到端跟踪监控

相关文档：

- 说明 Amazon CloudWatch 的工作原理
- Amazon Managed Prometheus
- Amazon Managed Grafana。
- Analyzing log data with CloudWatch Logs Insights
- Amazon CloudWatch Lambda 洞察
- Amazon CloudWatch Container Insights
- 使用 CloudWatch Metrics Insights 查询您的指标
- 适用于 OpenTelemetry 的 AWS Distro
- Amazon CloudWatch Logs Insights Sample Queries

- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- Searching and Filtering Log Data
- Sending Logs Directly to Amazon S3
- Amazon Builders' Library：检测分布式系统的运营可见性

相关讲习会：

- One Observability 讲习会

相关工具：

- AWS Distro for OpenTelemetry (GitHub)

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，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。

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

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 有助于您使用基于静态阈值、异常检测和其他标准的 CloudWatch 警报创建指标和复合警报。有关可以使用 CloudWatch 配置的警报类型的更多详细信息，请参阅 CloudWatch 文档的警报部分。

您可以使用 CloudWatch 控制面板为团队自定义 AWS 资源指标和警报的视图。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，或在 AWS Systems Manager 中创建 OpsItem 或事件。

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（使用者）的信息传递。要了解有关设置 Amazon SNS 通知的更多信息，请参阅 Configuring Amazon SNS。

每当 CloudWatch 警报被创建、更新、删除或警报状态发生变化时，CloudWatch 都会发送 EventBridge 事件。您可以将 EventBridge 与这些事件结合使用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 自动化在账户中自动触发事件。

随时了解 AWS Health 的最新信息。AWS Health 是有关 AWS Cloud资源运行状况的权威信息来源。使用 AWS Health 可获取任何已确认的服务事件的通知，以便您可以快速采取措施来减轻任何影响。通过 AWS 用户通知服务 创建要发送到电子邮件和聊天渠道且契合目标的 AWS Health 事件通知，并以编程方式通过 Amazon EventBridge 与监控和警报工具集成。如果您使用 AWS Organizations，则跨账户汇总 AWS Health 事件。

EventBridge 和 Amazon SNS 的使用时机

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。

如果想要构建能够对自己的应用程序、SaaS 应用程序和 AWS 服务中的事件做出反应的应用程序，建议使用 Amazon EventBridge。EventBridge 是唯一直接与第三方 SaaS 合作伙伴集成的基于事件的服务。EventBridge 还可以自动从 200 多项 AWS 服务中提取事件，无需开发者在其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 20 多项 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。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，可以采用任意格式。Amazon SNS 支持将消息转发到六种不同类型的目标，包括 Lambda、Amazon SQS、HTTP/S 端点、短信、移动推送和电子邮件。Amazon SNS 典型延迟不超过 30 毫秒。有许多 AWS 服务通过配置服务来发送 Amazon SNS 消息（超过 30 项服务，包括 Amazon EC2、Amazon S3 和 Amazon RDS）。

实施步骤

1. 使用 Amazon CloudWatch 警报创建警报。
    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。操作可以是向 Amazon SNS 主题发送通知、执行 Amazon EC2 操作或 Amazon EC2 Auto Scaling 操作，或在 AWS Systems Manager 中创建 OpsItem 或事件。
    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。复合警报可以在改变状态时发送 Amazon SNS 通知，并且可以在进入警报状态时创建 Systems Manager OpsItems 或事件，但无法执行 Amazon EC2 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 中创建与指定的 CloudWatch 警报相匹配的规则。每条规则都支持多个目标，包括 Lambda 函数。例如，您可以定义一个会在可用磁盘空间不足时启动的警报，通过 EventBridge 规则触发 Lambda 函数来清理空间。有关 EventBridge 目标的更多详细信息，请参阅 EventBridge targets。

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
- Using Amazon CloudWatch dashboards
- 使用 Amazon CloudWatch 指标
- 设置 Amazon SNS 通知
- CloudWatch 异常检测
- CloudWatch Logs data protection
- Amazon EventBridge
- Amazon Simple Notification Service

相关视频：

- re:Invent 2022 observability 视频
- AWS re:Invent 2022 – Observability best practices at Amazon

相关示例：

- One Observability 讲习会
- Amazon EventBridge to AWS Lambda with feedback control by Amazon CloudWatch Alarms

REL06-BP04 自动响应（实时处理和警报）

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，以便调用适当措施来维护服务级别目标和服务水平协议（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。

常见反模式：

- 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：自动处理警报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

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，判断哪些任务可以自动执行。如果操作可以定义，通常就可以实现自动化。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，您可以使用 Amazon CloudWatch 命令 `describe-alarms` 来使用 AWS CLI。根据设置的警报数量，您可能需要使用分页来检索每个呼叫的警报子集，或者也可以使用 AWS SDK 通过 API 调用获取警报。
2. 记录所有警报操作：更新包含所有警报及其操作的运行手册，无论它们是手动还是自动的。AWS Systems Manager 提供预定义的运行手册。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 View runbook content。
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。

   您还可以使用 Amazon EventBridge 自动响应系统事件，例如应用程序可用性问题或资源更改。您可以创建规则来指示要关注的事件，以及在事件匹配规则时要执行的操作。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 自动执行 Amazon EC2。

4. 标准操作程序（SOP）：根据应用程序组件，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。您还可以根据韧性监测中心的建议构造 SOP，前提是有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测。针对 SOP 的建议由韧性评测生成。

   韧性监测中心与 Systems Manager 结合使用，通过提供大量可用作这些 SOP 基础的 SSM 文档，自动执行 SOP 的步骤。例如，韧性监测中心可能会根据现有的 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，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 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
- Creating an EventBridge Rule That Triggers on an Event from an AWS Resource
- One Observability 讲习会
- Amazon Builders' Library：检测分布式系统的运营可见性
- What is Amazon DevOps Guru?
- 使用自动化文档（行动手册）

相关视频：

- AWS re:Invent 2022 – Observability best practices at Amazon

- AWS re:Invent 2020: Automate anything with AWS Systems Manager
- Introduction to AWS Resilience Hub
- Create Custom Ticket Systems for Amazon DevOps Guru Notifications
- Enable Multi-Account Insight Aggregation with Amazon DevOps Guru

相关示例：

- Amazon CloudWatch and Systems Manager 讲习会

REL06-BP05 分析日志

收集日志文件和指标历史记录并加以分析，获得更全面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，可用于分析日志数据。Amazon CloudWatch Logs 还支持订阅，允许数据无缝流动到 Amazon S3（可在其中使用数据）或 Amazon Athena（可在其中查询数据）。该服务支持查询多种格式。请参阅《Amazon Athena 用户指南》，了解支持的 SerDes 和数据格式。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 Glacier 进行长期存储，然后在保留期结束后让它们过期。S3 Intelligent-Tiering 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来优化成本，却不会对性能或运营开销产生影响。

在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

- 您可以使用 CloudWatch Logs Insights，通过交互方式搜索并分析 Amazon CloudWatch Logs 中的日志数据。
  - Analyzing Log Data with CloudWatch Logs Insights
  - Amazon CloudWatch Logs Insights Sample Queries

- 使用 Amazon CloudWatch Logs 将日志发送到 Amazon S3 以供使用，或发送到 Amazon Athena 以查询数据。
  - 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 为每个查询分析的数据量。
      - 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 Sample Queries
- Analyzing Log Data with CloudWatch Logs Insights
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- 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？
- One Observability 讲习会
- Amazon Builders' Library：检测分布式系统的运营可见性

REL06-BP06 定期审核监控范围和指标

经常审核工作负载监控的实施情况，并根据工作负载及其架构的发展进行更新。定期审计监控有助于降低遗漏或忽视故障指标的风险，并进一步协助工作负载实现其可用性目标。

有效的监控以关键业务指标为基础，这些指标会随着业务优先级变化而变化。监控审核过程应强调服务级别指标（SLI），并纳入来自基础设施、应用程序、客户和用户的见解。

期望结果：您拥有有效的监控策略，该策略会定期进行审核和更新，并在发生任何重大事件或变更后进行更新。随着工作负载和业务需求发生变化，您可以验证关键的应用程序运行状况指标是否仍然相关。

常见反模式：

- 您仅收集默认指标。
- 您设置了监控策略，但从不对其进行审核。
- 部署重大更改时，您不讨论监控。
- 您信任过时的指标来确定工作负载运行状况。

- 由于指标和阈值过时，误报的警报让您的运营团队不堪重负。
- 您对未受监控的应用程序组件缺乏可观测性。
- 在监控中，您只关注低级技术指标，而不关注业务指标。

建立这种最佳实践的好处：当您定期审核监控时，您可以预测潜在的问题，并验证自己是否有能力发现这些问题。它还可让您找出之前的审核中可能错过的盲点，从而进一步提高您发现问题的能力。

在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

在 operational readiness review (ORR) 过程中审核监控指标和范围。按照一致的时间表定期进行运营准备情况审查，以评估您当前的工作负载与您配置的监控之间是否存在任何差距。定期开展运营性能审查和知识共享，有助于增强运营团队提高绩效的能力。验证现有的警报阈值是否仍然适合，并检查运营团队是否收到误报的警报，或者是否未监控应用程序的应受监控的各个方面。

Resilience Analysis Framework 提供了有用的指导，有助于您驾驭整个过程。该框架的重点是确定潜在的故障模式，以及可用于减轻其影响的预防和纠正控制措施。这些知识有助于您确定要监控和发出警报的正确指标和事件。

实施步骤

1. 计划并执行工作负载控制面板常规检查。您可能对检查深度具有不同的安排。
2. 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。
3. 检查指标中是否存在离群值和异常值，这些值可能会被平均值或中位数掩盖。查看时间范围内的最高值和最低值，并调查观测结果远超正常范围的原因。随着您持续消除这些原因，您可以收紧预期的指标范围，以提高工作负载性能的一致性。
4. 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已发生变化，或者出现了需要添加额外指标进行跟踪的外部因素。
5. 审核当前的监控策略是否仍然与应用程序保持相关。根据对先前事件的分析（或韧性分析框架），评测该应用程序中是否还有其它方面应纳入监控范围。
6. 查看您的真实用户监控（RUM）指标，以确定应用程序功能覆盖范围是否存在任何差距。
7. 审查您的更改管理流程。如有必要，请更新相关过程，来包括应在批准更改之前执行的监控分析步骤。
8. 实施监控审核，以此作为运营准备情况审查和错误更正流程的一部分。

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-BP07 对系统中的请求进行端到端跟踪监控
- REL12-BP02 执行事后分析
- REL12-BP06 定期进行 GameDay 活动

相关文档：

- Why you should develop a correction of error (COE)
- 使用 Amazon CloudWatch 控制面板
- 构建控制面板以获取操作可见性
- Advanced Multi-AZ Resilience Patterns - Gray failures
- Amazon CloudWatch Logs Insights Sample Queries
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- One Observability 讲习会
- Amazon Builders' Library：检测分布式系统的运营可见性
- Using Amazon CloudWatch Dashboards
- AWS Observability Best Practices
- Resilience Analysis Framework
- Resilience Analysis Framework - Observability
- Operational Readiness Review - ORR

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 金丝雀或真实用户监控（RUM）未配置跟踪功能。没有金丝雀或 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。

在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 金丝雀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水平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，在请求通过应用程序时提供请求的完整视图。X-Ray 会收集应用程序遥测，有助于跨有效负载、函数、跟踪、服务、API 对其进行可视化和筛选，并且可以通过无代码或低代码的方式系统组件启用。CloudWatch 应用程序监控包括 ServiceLens，可将跟踪与指标、日志和警报集成。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

- 在所有支持的本机服务上使用 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 管理控制台 来启用 X-Ray。
- 检测应用程序（适用于 OpenTelemetry 的 AWS Distro 和 X-Ray）或第三方收集代理。
- 查看《AWS X-Ray Developer Guide》，了解编程语言特定的实施。这些文档部分详细介绍了如何检测 HTTP 请求、SQL 查询和应用程序编程语言特定的其他进程。

- 使用适用于 Amazon CloudWatch Synthetics 金丝雀 和 Amazon CloudWatch RUM 的 X-Ray 追踪，对最终用户客户端通过下游 AWS 基础设施的请求路径进行分析。
- 根据资源运行状况和金丝雀遥测数据来配置 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。
- 如果使用第三方工具作为主要的追踪解决方案，则将 X-Ray 与 Datadog、New Relic 或 Dynatrace 等第三方追踪工具集成。

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
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- Amazon Builders' Library：检测分布式系统的运营可见性
- Integrating AWS X-Ray with other AWS services
- AWS Distro for OpenTelemetry and AWS X-Ray
- Amazon CloudWatch：使用综合监控
- Amazon CloudWatch：使用 CloudWatch RUM
- Set up Amazon CloudWatch synthetics canary and Amazon CloudWatch alarm
- Availability and Beyond: Understanding and Improving the Resilience of Distributed Systems on AWS

相关示例：

- One Observability 讲习会

相关视频：

- AWS re:Invent 2022 – How to monitor applications across multiple accounts
- How to Monitor your AWS Applications

相关工具：

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

设计工作负载来适应需求变化

可扩展工作负载提供了自动添加或删除资源的弹性，因此资源在任何给定时间点都非常符合当前需求。

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 在检测到某个工作负载需要更多资源时获取资源
- 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

云端可靠性的基石是基础设施和资源的程序化定义、预置和管理。自动化有助于您简化资源预置，促进一致和安全的部署，并在整个基础设施中扩展资源。

期望结果：管理基础设施即代码（IaC）。您可以在版本控制系统（VCS）中定义和维护基础设施代码。您可以将预置 AWS 资源的过程委托给自动机制，并利用托管式服务，例如应用程序负载均衡器（ALB）、网络负载均衡器（NLB）和自动扩缩组。您可以使用持续集成/持续交付（CI/CD）管道来预置资源，以便代码更改自动启动资源更新，包括对自动扩缩配置的更新。

常见反模式：

- 您使用命令行或在 AWS 管理控制台中（也称为单击操作）手动部署资源。
- 您将应用程序组件或资源紧密耦合，从而创建了不灵活的架构。
- 您实施的扩展策略不灵活，无法适应不断变化的业务需求、流量规律或新的资源类型。
- 您手动估算容量来满足预期需求。

建立这种最佳实践的好处：基础设施即代码（IaC）支持以编程方式定义基础设施。这有助于您在与应用程序更改相同的软件开发生命周期中管理基础设施更改，从而提高一致性和可重复性，并降低手动、易出错任务的风险。通过使用自动交付管道实施 IaC，可以进一步简化预置和更新资源的流程。您无需手动干预，即可可靠、高效地部署基础设施更新。在扩展资源以满足不断变化的需求时，这种敏捷性尤其重要。

您可以结合使用 IaC 和交付管道来实现动态、自动的资源扩展。通过监控关键指标和应用预定义的扩展策略，自动扩缩可以根据需要自动预置或取消预置资源，从而提高性能和成本效益。这样可以减少因应用程序或工作负载要求发生变化而导致手动错误或延迟的可能性。

IaC、自动交付管道和自动扩缩相结合，有助于组织放心地预置、更新和扩展其环境。这种自动化对于维护响应迅速、富有韧性和高效管理的云基础设施至关重要。

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

要为 AWS 架构的 CI/CD 管道和基础设施即代码（IaC）设置自动化，请选择版本控制系统（例如 Git）来存储 IaC 模板和配置。这些模板可以使用诸如 AWS CloudFormation 之类的工具编写。首先，请在这些模板中定义基础设施组件（例如 AWS VPC、Amazon EC2 Auto Scaling 组和 Amazon RDS 数据库）。

接下来，将这些 IaC 模板与 CI/CD 管道集成，以实现部署过程的自动化。AWS CodePipeline 提供了无缝的 AWS 原生解决方案，您也可以使用其它第三方 CI/CD 解决方案。创建一个在版本控制存储库发生更改时激活的管道。将管道配置为包括以下各个阶段：整理和验证 IaC 模板、将基础设施部署到暂存环境、运行自动测试以及最终部署到生产环境。必要时纳入审批步骤，以保持对变更的控制。这种自动化的管道不仅加快了部署速度，而且促进了跨环境的一致性和可靠性。

在 IaC 中配置诸如 Amazon EC2 实例、Amazon ECS 任务和数据库副本等资源的自动扩缩，以根据需要提供自动横向扩展和横向缩减。这种方法通过根据需求动态调整资源，来增强应用程序可用性和性能并优化成本。有关支持的资源列表，请参阅 Amazon EC2 Auto Scaling 和 AWS Auto Scaling。

实施步骤

1. 创建并使用源代码存储库，来存储控制基础设施配置的代码。提交对此存储库的更改，来反映您要进行的任何持续更改。
2. 选择基础设施即代码解决方案（例如 AWS CloudFormation），以使您的基础设施保持最新状态并检测与预期状态的不一致性（偏差）。
3. 将 IaC 平台与 CI/CD 管道集成来实现部署自动化。

4. 确定并收集用于自动扩缩资源的适当指标。
5. 使用适用于工作负载组件的横向扩展和横向缩减策略，来配置资源的自动扩缩。考虑使用计划的扩展来实现可预测的使用规律。
6. 监控部署来检测故障和回归。在 CI/CD 平台中实施回滚机制，以便在必要时还原更改。

资源

相关文档：

- AWS Auto Scaling: How Scaling Plans Work
- AWS Marketplace: products that can be used with auto scaling
- 使用 DynamoDB Auto Scaling 自动管理吞吐能力
- Using a load balancer with an Auto Scaling group
- What Is AWS Global Accelerator?
- What Is Amazon EC2 Auto Scaling?
- 什么是 AWS Auto Scaling？
- What is Amazon CloudFront?
- What is Amazon Route 53?
- 什么是 Elastic Load Balancing？
- 什么是网络负载均衡器？
- 什么是应用程序负载均衡器？
- Integrating Jenkins with AWS CodeBuild and AWS CodeDeploy
- Creating a four stage pipeline with AWS CodePipeline

相关视频：

- Back to Basics: Deploy Your Code to Amazon EC2
- AWS Supports You | Starting Your Infrastructure as Code Solution Using AWS CloudFormation Templates
- Streamline Your Software Release Process Using AWS CodePipeline
- Monitor AWS Resources Using Amazon CloudWatch Dashboards
- Create Cross Account & Cross Region CloudWatch Dashboards | Amazon Web Services

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。

可以根据工作负载手动调整资源规模（例如，通过 AWS 管理控制台 或 AWS CLI 更改自动扩缩组中 EC2 实例的数量，或者修改 DynamoDB 表的吞吐量）。但是，应尽量采用自动化技术（请参阅在获取或扩展资源时利用自动化）。

期望结果：在检测到故障或客户体验降级时，启动扩缩活动（自动或手动）来恢复可用性。

在未建立这种最佳实践的情况下暴露的风险等级：中

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。定义用于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。
  - 扩缩程序取决于工作负载中不同组件的设计方式。
  - 扩缩程序还因所使用的底层技术而异。
    - 使用 AWS Auto Scaling 的组件可以使用扩缩计划来配置一组用于扩展资源的指令。如果使用 AWS CloudFormation 或向 AWS 资源添加标签，则可以根据应用程序为不同的资源集设置扩缩计划。Auto Scaling 提供了针对每个资源的定制扩展策略建议。创建扩缩计划后，Auto Scaling 结合了动态扩缩和预测性扩缩方法来支持扩缩策略。有关更多详细信息，请参阅 How scaling plans work。
    - Amazon EC2 Auto Scaling 可验证您具有正确数量的 Amazon EC2 实例来处理应用程序负载。您可创建 EC2 实例的集合，称为自动扩缩组。您可以指定每个自动扩缩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 What is Amazon EC2 Auto Scaling?
    - Amazon DynamoDB Auto Scaling 会根据实际的流量模式，使用 Application Auto Scaling 服务代表您动态调整预置的吞吐容量。这样表或全局二级索引可以增加预置读取和写入容量处理突增流量，不会受到限制。有关更多详细信息，请参阅使用 DynamoDB Auto Scaling 自动管理吞吐能力。

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: How Scaling Plans Work
- 使用 DynamoDB Auto Scaling 自动管理吞吐能力
- What Is Amazon EC2 Auto Scaling?

REL07-BP03 在检测到某个工作负载需要更多资源时获取资源

云计算最有价值的功能之一是能够动态预置资源。

在传统的本地计算环境中，您必须提前确定和预置足够的容量来满足峰值需求。这的确是个问题，因为很昂贵，如果您低估了工作负载的峰值容量需求，则会对可用性构成风险。

在云中，您不必这样做。相反，您可以根据需要预置计算、数据库和其它资源容量，以满足当前和预测的需求。Amazon EC2 Auto Scaling 和 Application Auto Scaling 等自动化解决方案可以根据您指定的指标在线为您提供资源。这可以使扩展过程变得更加轻松和可预测，还可以通过确保始终有足够的可用资源来显著提高工作负载的可靠性。

期望结果：您可以配置计算资源和其它资源的自动扩缩来满足需求。您在扩展策略中提供了充足的余量，可以在额外资源上线时应对流量爆增。

常见反模式：

- 您预置固定数量的可扩展资源。
- 您选择的扩展指标与实际需求不相关。
- 您未能在扩展计划中提供足够的余量来应对需求爆增。
- 您的扩展策略添加容量的时间过晚，这会导致容量耗尽和服务降级，同时使额外的资源上线。
- 您未能正确地配置最小和最大资源计数，这会导致扩展失败。

建立此最佳实践的好处：拥有足够的资源来满足当前需求，对于提供工作负载的高可用性和遵守您定义的服务级别目标（SLO）至关重要。自动扩缩可让您提供工作负载所需的适量计算资源、数据库资源和

其它资源，以满足当前和预测的需求。您无需确定峰值容量需求，也无需静态分配资源来满足此需求。相反，随着需求增长，您可以分配更多资源来满足需求，在需求下降后，您可以停用资源以降低成本。

在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

首先，确定工作负载组件是否适合自动扩缩。这些组件之所以称为可水平扩展，是因为它们提供的资源相同，行为也相同。水平可扩展组件的示例包括以类似方式配置的 EC2 实例、Amazon Elastic Container Service（ECS）任务以及在 Amazon Elastic Kubernetes Service（EKS）上运行的容器组（pod）。这些计算资源通常位于负载均衡器后面，称为副本。

其它复制的资源可能包括数据库只读副本、Amazon DynamoDB 表以及 Amazon ElastiCache（Redis OSS）集群。有关支持的资源的完整列表，请参阅 AWS services that you can use with Application Auto Scaling。

对于基于容器的架构，可能需要通过两种不同的方式进行扩展。首先，您可能需要扩展可提供水平可扩展服务的容器。其次，您可能需要扩展计算资源，以便为新容器腾出空间。每个层都有不同的自动扩缩机制。要扩展 ECS 任务，可以使用 Application Auto Scaling。要扩展 Kubernetes 容器组（pod），可以使用 Pod 水平自动扩缩控制器（HPA）或 Kubernetes Event-driven Autoscaling (KEDA)。要扩展计算资源，对于 ECS，可以使用容量提供程序，或者对于 Kubernetes，可以使用 Karpenter 或集群自动扩缩器。

接下来，选择您将如何执行自动扩缩。有三个主要选项：基于指标的扩展、计划扩展和预测式扩展。

基于指标的扩展

基于指标的扩展根据一个或多个扩展指标 的值来预置资源。扩展指标是与工作负载的需求相对应的指标。确定适当扩展指标的一个好方法是在非生产环境中执行负载测试。在负载测试期间，请将可扩展资源的数量保持为固定，并缓慢增加需求（例如，吞吐量、并发性或模拟用户数）。然后，寻找随着需求增长而增加（或减少）的指标，以及相反，即随着需求下降而减少（或增加）的指标。典型的扩展指标包括 CPU 利用率、工作队列深度（例如 Amazon SQS 队列）、活跃用户数量和网络吞吐量。

> **Note**
>
> AWS 观察到，对于大多数应用程序，内存利用率会随着应用程序预热而增加，然后达到稳定的值。当需求减少时，内存利用率通常保持较高水平，而不是并行下降。因为内存利用率与两个方向的需求不符（即随需求而增长和下降），因此在选择此指标进行自动扩缩之前，请慎重考虑。

基于指标的扩展是一种潜在操作。利用率指标可能需要几分钟才能传播到自动扩缩机制，而这些机制通常要等到明确的需求增加信号后才会做出反应。然后，当自动扩缩器创建新资源时，这些资源可能需要更多时间才能全面投入使用。因此，重要的是不要将扩展指标目标设置为过于接近完全利用率（例如，90% 的 CPU 利用率）。这样做有可能在额外容量上线之前耗尽现有资源容量。为了获得最佳可用性，典型的资源利用率目标可介于 50-70% 之间，具体取决于需求规律以及预置额外资源所需的时间。

计划扩展

计划扩展会根据日历或一天中的时间预置或移除资源。它通常用于需求可预测的工作负载，例如工作日工作时间或销售活动期间的峰值利用率。Amazon EC2 Auto Scaling 和 Application Auto Scaling 都支持计划扩展。KEDA 的 cron scaler 支持 Kubernetes 容器组（pod）的计划扩展。

预测式扩展

预测式扩展使用机器学习来根据预期的需求自动扩缩资源。预测式扩展分析您提供的利用率指标的历史值，并持续预测其未来值。然后，使用预测值来纵向扩展或缩减资源。Amazon EC2 Auto Scaling 可以执行预测式扩展。

实施步骤

1. 确定工作负载组件是否适合自动扩缩。
2. 确定哪种扩展机制最适合工作负载：基于指标的扩展、计划扩展或预测式扩展。
3. 为组件选择适当的自动扩缩机制。对于 Amazon EC2 实例，请使用 Amazon EC2 Auto Scaling。对于其它 AWS 服务，请使用 Application Auto Scaling。对于 Kubernetes 容器组（pod）（例如在 Amazon EKS 集群中运行的容器），可以考虑水平容器组（pod）自动扩缩器（HPA）或 Kubernetes 事件驱动的自动扩缩（KEDA）。对于 Kubernetes 或 EKS 节点，可以考虑 Karpenter 和集群自动扩缩器（CAS）。
4. 对于指标或计划扩展，请进行负载测试，以确定工作负载的相应扩展指标和目标值。对于计划扩展，请确定在您选择的日期和时间所需的资源数量。确定为应对预期的峰值流量所需的最大资源数量。
5. 根据上面收集的信息配置自动扩缩器。有关详细信息，请参阅自动扩缩服务的文档。验证最大和最小扩展限制是否配置正确。
6. 验证扩展配置是否按预期发挥作用。在非生产环境中执行负载测试，观察系统的反应，并根据需要进行调整。在生产环境中启用自动扩缩时，请配置适当的警报来向您通知任何意外行为。

资源

相关文档：

- What Is Amazon EC2 Auto Scaling?
- AWS Prescriptive Guidance: Load testing applications
- AWS Marketplace: products that can be used with auto scaling
- 使用 DynamoDB Auto Scaling 自动管理吞吐能力
- Predictive Scaling for EC2, Powered by Machine Learning
- Scheduled Scaling for Amazon EC2 Auto Scaling
- Telling Stories About Little's Law

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 GameDay 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。

在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
- 加载测试应用程序

相关视频：

- AWS Summit ANZ 2023: Accelerate with confidence through AWS Distributed Load Testing

实施变更

要部署新功能并确保工作负载及运行环境运行已知且经过适当修补的软件，就必须进行受控更改。如果这些更改不受控制，那么就很难预测这些更改的影响，也很难解决由此产生的问题。

其他可将风险最小化的部署模式

功能标记（又被称作功能切换）是应用程序上的配置选项。您可以在部署软件时将某个功能关闭，避免客户看到该功能。然后可以像进行金丝雀部署那样启用该功能，也可以将更改速度设置为 100% 来查看效果。如果部署有问题，只需关闭该功能即可，无需回滚。

故障隔离区域部署：AWS 针对自己的部署制定的最重要规则之一，就是避免同时接触一个区域内的多个可用区。这条规则非常重要，可以确保可用区彼此独立，方便计算可用性。我们建议您在自己的部署中，也做同样的考量。

运营准备情况审查（ORR）

AWS 发现，执行运营准备情况审查非常有用。该审查可以评估测试的完整性、监控能力，更重要的是，其可根据应用程序的 SLA 进行性能审计的能力，以及在出现中断或其他操作异常时提供数据的

能力。在初始生产部署之前会进行正式的 ORR。AWS 会定期重复执行 ORR（每年一次或在关键性能阶段之前），确保没有“偏离”运营预期。如需关于运营准备情况的更多信息，请参阅 AWS Well-Architected Framework 中的卓越运营支柱。

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。示例包括部署工作负载，对其进行修补或修改 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，并在适当的情况下调用流程自动运行。

即使是高度自动化的复杂工作负载，运行手册仍可用于开展 GameDay 活动或满足严格的报告和审核要求。

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。
- 为加快部署速度而跳过计划中的步骤，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：有效的更改计划有助于成功执行更改，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。

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

- 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。
- 使用基础设施即代码的原则定义基础设施。通过使用 AWS CloudFormation（或受信任的第三方）来定义基础设施，您可以使用版本控制软件管理并跟踪更改。
  - 使用 AWS CloudFormation（或受信任的第三方提供商）定义基础设施。
    - 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
    - 使用基于 Git 等流行技术的托管源代码管理系统，来存储源代码和基础设施即代码（IaC）配置。

资源

相关文档：

- APN 合作伙伴：可帮助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
- 什么是 AWS CloudFormation？

相关示例：

- 根据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

使用单元测试和集成测试等技术来验证所需功能。

单元测试是测试代码的最小功能单元来验证其行为的过程。集成测试旨在验证每个应用程序功能是否符合软件要求。虽然单元测试侧重于以隔离方式测试应用程序的一部分，但集成测试会考虑副作用（例如，通过变异操作更改数据所带来的影响）。无论是哪种情况，都应将测试集成到部署管道中，若未能满足成功条件，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。

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，开发人员将更改提交到源代码存储库，而 CodePipeline 会自动在其中检测这些更改。构建应用程序，并运行单元测试。在单元测试获得通过之后，将构建的代码部署到暂存服务器来进行测试。CodePipeline 会从暂存服务器运行更多测试，例如集成或负载测试。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。

期望结果：您使用自动化功能来执行单元测试和集成测试，以验证您的代码是否按预期运行。这些测试集成到部署过程中，而测试失败会中止部署。

常见反模式：

- 您在部署过程中忽略或绕过测试失败和测试计划，以加快部署时间表。
- 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，而采用手动应急工作流程。
- 您在与生产环境不太相似的环境中运行自动测试。
- 您构建的测试套件不够灵活，难以随应用程序发展来进行维护、更新或扩展。

建立这种最佳实践的好处：在部署过程中进行自动测试可以及早发现问题，从而降低发布到生产环境时出现错误或意外行为的风险。单元测试可验证代码是否按预期运行，以及 API 合约是否得到遵守。集成测试可验证系统是否按照指定的要求运行。这些类型的测试可验证诸如用户界面、API、数据库和源代码等组件是否按预期正常运行。

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

采用测试驱动型开发（TDD）方法编写软件，从而通过开发测试用例来指定和验证代码。首先，为每个函数创建测试用例。如果测试失败，则编写新的代码来通过测试。这种方法有助于您验证每个函数的预期结果。在将代码提交到源代码存储库之前，请运行单元测试并验证测试获得通过。

执行单元测试和集成测试，作为 CI/CD 管道的构建、测试和部署阶段的一部分。自动执行测试，并在准备好部署新版本的应用程序时自动启动测试。若未满足成功条件，则相关管道会中止或回滚。

如果应用程序是 Web 或移动应用程序，请在多个桌面浏览器或实际设备上执行自动的集成测试。这种方法对于验证移动应用程序在各种设备上的兼容性和功能性特别有用。

实施步骤

1. 在编写功能代码（测试驱动型开发 或 TDD）之前，先编写单元测试。制定代码指南，使编写和运行单元测试成为非功能性编码要求。

2. 创建一套自动化集成测试，其中涵盖已确定的可测试功能。这些测试应模拟用户互动并验证预期结果。
3. 创建运行集成测试所需的测试环境。这可能包括与生产环境非常相似的暂存环境或预生产环境。
4. 使用 AWS CodePipeline 控制台或 AWS Command Line Interface（CLI）设置源代码以及构建、测试和部署各个阶段。
5. 在代码构建和测试完毕后部署应用程序。AWS CodeDeploy 可以将其部署到您的暂存（测试）环境和生产环境。这些环境可能包括 Amazon EC2 实例、AWS Lambda 函数或本地服务器。应使用相同的部署机制将应用程序部署到所有环境。
6. 监控管道的进度以及每个阶段的状态。使用质量检查，根据测试的状态来阻止管道。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。
7. 持续监控测试结果，并查找规律、回归或需要更多关注的领域。使用这些信息来改进测试套件，确定应用程序中需要更稳健测试的领域，并优化部署过程。

资源

相关最佳实践：

- REL07-BP04 对工作负载进行负载测试
- REL08-BP03 将韧性测试作为部署的一部分进行集成
- REL12-BP04 使用混沌工程测试韧性

相关文档：

- AWS Prescriptive Guidance: Test automation
- 持续交付和持续集成
- Indicators for functional testing
- Monitoring pipelines
- Use AWS CodePipeline with AWS CodeBuild to test code and run builds
- AWS Device Farm

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 GameDay 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

建立最佳实践的好处：部署中集成的韧性测试有助于识别系统中的未知问题，以防因忽视这些问题导致生产中断。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

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）的更新。
- 将可靠性测试集成到自动部署管道中。诸如 AWS Resilience Hub 的服务可以集成到 CI/CD 管道中，用于建立持续的韧性评测，这些评测将作为每次部署的一部分自动进行评估。
- 在 AWS Resilience Hub 中定义应用程序。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、热备用和主动-主动模式，这些策略的成本和复杂性各不相同。在执行灾难恢复测试之前，建议先定义恢复时间目标（RTO）和恢复点目标（RPO），简化模拟策略的选择。AWS 提供 AWS Elastic Disaster Recovery 等灾难恢复工具，可帮助您开始规划和测试。

- 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。与其他策略一样，使用诸如 AWS Fault Injection Service 的服务在非生产环境中运行受控故障模拟，以便在生产环境中部署之前增强信心。

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
- 混沌工程原则
- Chaos Engineering 讲习会

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，就禁止为了在工作负载中运行基础设施资源而进行就地修改。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。

常见反模式：

- 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：

- 提高环境间的一致性：由于环境间的基础设施资源没有差异，可以提高一致性并简化测试。
- 减小配置偏差：通过使用已知且版本受控的配置替换基础设施资源，基础设施被设置为已知经过测试的可信状态，避免配置偏差。
- 采用可靠的原子部署：部署要么成功完成，要么没有任何更改，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实现更安全的部署：由于之前运行的版本未发生更改，部署变得更安全。您可以在检测到错误时进行回滚。
- 增强安全态势：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，改善组织的安全态势。

在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

– 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功能来提高可重复性，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实现无需干预的安全部署。

借助基础设施即代码（IaC），基础设施预置、编排和部署步骤以编程、描述性和声明性的方式进行定义，并存储在源代码控制系统中。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：将少量客户引导到新版本的做法，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，您可以将金丝雀中的流量删除，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，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。

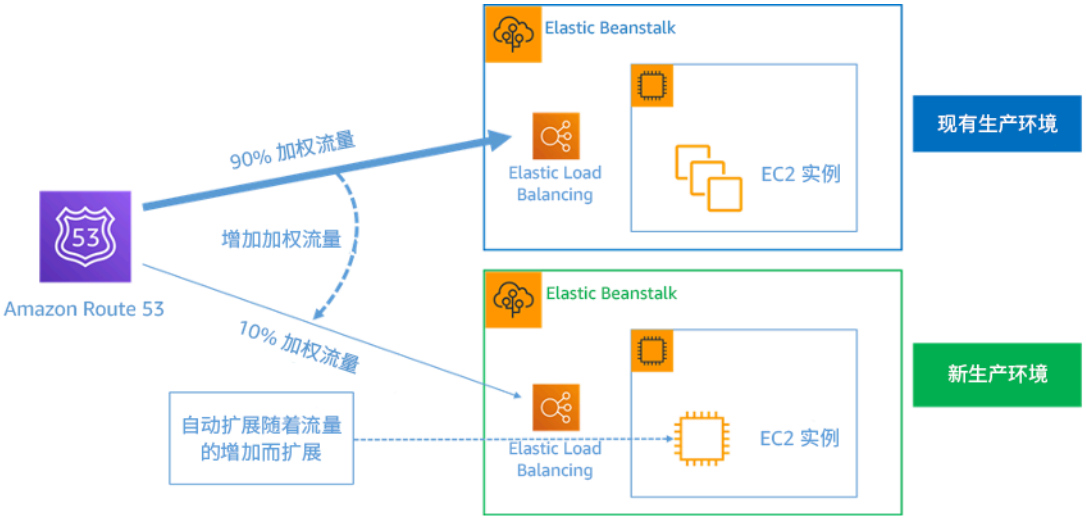


图 8：使用 AWS Elastic Beanstalk 和 Amazon Route 53 进行蓝绿部署

偏差检测

偏差定义为导致基础设施资源的状态或配置与预期不同的任何更改。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）来指定可以访问 AWS 中服务和资源的对象，集中管理精细权限，并分析访问权限来细化跨 AWS 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 DevOps 简介》白皮书中所述，自动化是 AWS 服务的基石，并且在所有服务、功能和产品中均得到内部支持。
  - 预烘焙亚马逊机器映像（AMI）可以加快启动速度。EC2 Image Builder 是一项完全托管的 AWS 服务，可帮助您自动创建、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 是一项服务，用于在熟悉的服务器（例如 Apache、NGINX、Passenger 和 IIS）上部署和扩展使用 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、描述性和声明性的方式创建、部署和维护基础设施。
  - AWS CloudFormation 帮助开发人员以有序且可预测的方式创建 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。可以使用任何代码编辑器以 JSON 或 YAML 格式创作资源，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 CloudFormation 的扩展。
  - AWS Cloud Development Kit (AWS CDK) 是一个开源软件开发框架，可使用熟悉的编程语言对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 CloudFormation，以安全、可重复的方式预置资源。
  - AWS 云端控制 API 引入了一组通用的创建、读取、更新、删除和列出（CRUDL）API，帮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 创建具有金丝雀部署的管道
  - 蓝绿部署：《Blue/Green Deployments on AWS》白皮书介绍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托管配置更改。

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：

- 自动实现无需干预的安全部署
- Leveraging AWS CloudFormation to create an immutable infrastructure at Nubank
- 基础设施即代码
- Implementing an alarm to automatically detect drift in AWS CloudFormation stacks

相关视频：

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

REL08-BP05 使用自动化功能部署更改

自动执行部署与修补来消除负面影响。

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：通过广泛的预生产测试、自动回滚和错开生产部署，将自动化部署安全性融入发布流程。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。

常见反模式：

- 手动执行更改。
- 跳过自动化流程中的步骤，采用手动应急工作流。
- 不遵循既定的计划和流程，急于求成。
- 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：当使用自动化来部署所有更改时，可以消除引入人为错误的可能性，并提供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

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：

- 设置代码存储库以安全地存储您的代码：使用基于 Git 等流行技术的托管源代码管理系统，来存储源代码和基础设施即代码（IaC）配置。
- 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。
- 设置部署服务来自动执行应用程序部署并处理复杂的应用程序更新，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
- APN 合作伙伴：可帮助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
- 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
- 什么是 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。不管使用的是最高质量的硬件还是最低成本的组件，必然会出现故障 – Amazon.com 首席技术官 Werner Vogels

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，在其中被自动复制，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，都有可能遭遇会对工作负载造成影响的故障。因此，如果想让工作负载具有可靠性，您必须采取措施来实施韧性。

要运用此处讨论的最佳实践，必须先确保负责设计实施并运行工作负载的人员了解业务目标以及实现这些目标的可靠性目标。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。

以下各节旨在介绍管理故障的最佳实践，可预防对工作负载造成影响。

主题

- 备份数据
- 使用故障隔离来保护工作负载
- 设计工作负载来承受组件故障
- 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，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 识别并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 识别并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 Glacier 或 Amazon 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，因此对韧性具有不同的要求。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 上创建各种数据来源的备份。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，了解有关如何从不同 AWS 服务创建数据备份的信息。
4. 为没有备份的数据建立数据复制机制。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，若从 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，这就算是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，您就不会因为数据仓库或 MapReduce 集群故障而承受数据丢失风险。其他可从数据来源复制数据的例子包括缓存（如 Amazon ElastiCache）或 RDS 只读副本。
5. 制定备份数据的频率。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：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？
- What is AWS DataSync?
- What is Volume Gateway?
- APN 合作伙伴：可帮助进行备份的合作伙伴
- AWS Marketplace：可用于备份的产品
- Amazon EBS Snapshots
- Backing Up Amazon EFS
- Backing up Amazon FSx for Windows File Server
- Backup and Restore for ElastiCache for Redis
- Creating a DB Cluster Snapshot in Neptune
- 创建数据库快照
- Creating an EventBridge Rule That Triggers on a Schedule
- 使用 Amazon S3 进行跨区域复制
- EFS 到 EFS AWS Backup
- Exporting Log Data to 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
- Working with Amazon OpenSearch Service Index Snapshots
- 什么是 AWS Elastic Disaster Recovery？

相关视频：

- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS
- AWS Backup Demo: Cross-Account and Cross-Region Backup
- AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace (STG341)

REL09-BP02 保护并加密备份

使用身份验证和授权功能来控制并检测对备份的访问。使用加密功能防止备份的数据完整性遭到破坏，以及检测其完整性是否遭到损坏。

实施安全控制措施，以防止未经授权访问备份数据。加密备份以保护数据的机密性和完整性。

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。
- 不加密备份。
- 未实施不可变性来防止删除或篡改。
- 对生产系统和备份系统使用相同的安全域。
- 未通过定期测试来验证备份完整性。

建立此最佳实践的好处：

- 保护备份安全可防止篡改数据，而对数据进行加密可防止数据在意外暴露时遭到访问。
- 增强了对勒索软件及其它针对备份基础设施的网络威胁的防护。
- 通过经过验证的恢复流程，缩短了网络事件后的恢复时间。
- 提高了安全事件期间的业务连续性能力。

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

使用 AWS Identity and Access Management（IAM）等身份验证和授权服务来控制并检测对备份的访问。使用加密功能防止备份的数据完整性遭到破坏，以及检测其完整性是否遭到损坏。

Amazon S3 支持多种对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对象，然后在存储此类数据时进行加密。若采用客户端加密，工作负载应用程序需要负

责在将数据发送到 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、谁不可以访问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，那么备份也会被加密。DynamoDB 备份始终加密。使用 AWS Elastic Disaster Recovery 时，加密所有传输中数据和静态数据。借助弹性灾难恢复，可以使用默认的 Amazon EBS 加密“卷加密密钥”或自定义的客户自主管理型密钥来加密静态数据。

网络弹性注意事项

为了增强针对网络威胁的备份安全性，除了加密之外，还可以考虑实施这些额外的控制措施：

- 使用 AWS Backup 保管库锁或 Amazon S3 对象锁定来实现不可变性，以防止备份数据在其保留期内遭到更改或删除，从而防范勒索软件和恶意删除。
- 使用 AWS Backup 逻辑上受物理隔离的保管库为关键系统建立生产环境与备份环境之间的逻辑隔离，从而实现分离来协助防止这两个环境同时受到危害。
- 定期使用 AWS Backup 还原测试来验证备份的完整性，以验证备份没有受损并且可以在发生网络事件后成功恢复。
- 使用 AWS Backup 多方审批对关键的恢复操作实施多方审批，通过要求由多个指定批准者授权来防止未经授权或恶意的恢复尝试。

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 可加密所有静态数据。您可以使用 AWS 拥有的 AWS KMS 密钥或者 AWS 托管式 KMS 密钥，指定存储在账户中的密钥。
   - 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 中存储的密钥在 Amazon S3 中配置静态加密，但这些密钥是区域特定密钥。您在配置复制时可以指定目标密钥。
   - 选择是为弹性灾难恢复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。

3. 为关键备份配置不可变性。对于关键数据，实施 AWS Backup 保管库锁或 S3 对象锁定，以防止在指定的保留期内遭到删除或更改。有关实施详细信息，请参阅 AWS Backup Vault Lock。
4. 为备份环境创建逻辑分离。为需要增强网络威胁防护的关键系统实施 AWS Backup 逻辑上受物理隔离的保管库。有关实施指南，请参阅 Building cyber resiliency with AWS Backup logically air-gapped vault。
5. 实施备份验证流程。配置 AWS Backup 还原测试，以定期验证备份没有受损并且可以在发生网络事件后成功恢复。有关更多信息，请参阅 Validate recovery readiness with AWS Backup restore testing。
6. 为敏感的恢复操作配置多方审批。对于关键系统，实施 AWS Backup 多方审批，以便要求在继续恢复之前获得多个指定批准者的授权。有关实施详细信息，请参阅 Improve recovery resilience with AWS Backup support for Multi-party approval。

资源

相关文档：

- AWS Marketplace：可用于备份的产品
- Amazon EBS Encryption
- Amazon S3：利用加密来保护数据
- 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
- Encrypting Data and Metadata in Amazon EFS
- Encryption for Backups in AWS
- 管理加密表
- 安全性支柱 - AWS Well-Architected Framework
- 什么是 AWS Elastic Disaster Recovery？
- FSISEC11: How are you protecting against ransomware?
- Ransomware Risk Management on AWS Using the NIST Cyber Security Framework
- Building cyber resiliency with AWS Backup logically air-gapped vault
- Validate recovery readiness with AWS Backup restore testing
- Improve recovery resilience with AWS Backup support for Multi-party approval

相关示例：

- Implementing Bi-Directional Cross-Region Replication (CRR) for Amazon S3

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。具有低数据丢失要求的关键数据集，需要频繁地自动备份；而可以接受一定丢失的较不关键的数据，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 定期执行备份，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

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 版本控制即会自动工作。Amazon Data Lifecycle Manager 可用于自动创建、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 支持的亚马逊机器映像（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 识别并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 Testing Backup and Restore of Data。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 确定的频率来调用它。

实施计划的工作量级别：低

资源

相关文档：

- APN 合作伙伴：可帮助进行备份的合作伙伴
- AWS Marketplace：可用于备份的产品
- Creating an EventBridge Rule That Triggers on a Schedule
- 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: Deep dive on AWS Backup, ft. Rackspace (STG341)

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

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，还原您的备份以评测 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 还允许时间点故障恢复（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。弹性灾难恢复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 识别并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。在确信可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. 设立 RTO 和 RPO，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. 评测恢复能力。检查备份和还原策略，了解是否可以满足 RTO 和 RPO，再根据需要调整策略。使用 AWS 韧性监测中心，可对工作负载运行评估。该评测根据韧性策略评估应用程序配置，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，若使用的是 AWS Backup 等托管服务，则此流程可能就是简单地将备份还原到新的资源。如果使用的是 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，比较还原流程开始时的时间戳以及恢复验证完成时的时间戳，由此计算此流程的用时。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流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验证逻辑来记录。如果使用自动流程，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，则通知利益相关方。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）发送给利益相关方。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 可用于定期调用此自动工作流，如以下架构图所示。了解如何使用 AWS Backup 自动完成数据恢复验证。此外，这个 Well-Architected Lab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。

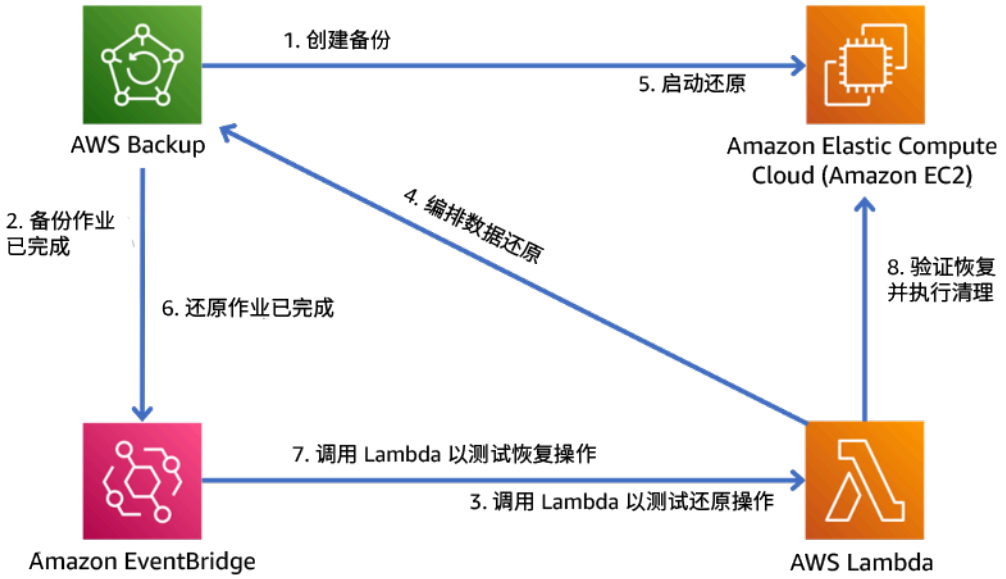


图 9：自动化的备份和还原流程

实施计划的工作量级别：中到高，具体取决于验证标准的复杂性。

资源

相关文档：

- Automate data recovery validation with AWS Backup
- APN 合作伙伴：可帮助进行备份的合作伙伴
- AWS Marketplace：可用于备份的产品
- Creating an EventBridge Rule That Triggers on a Schedule
- 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

使用故障隔离来保护工作负载

故障隔离可将组件或系统故障的影响限制在定义的界限内。通过适当的隔离，界限之外的组件不受故障影响。跨多个故障隔离界限运行工作负载，可以提高工作负载对故障的韧性。

最佳实践

- REL10-BP01 将工作负载部署到多个位置
- REL10-BP02 组件的自动恢复受限于单个位置
- REL10-BP03 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。

AWS 中服务设计的一个基本原则是避免单点故障，包括底层物理基础设施。AWS 在全球多个地理位置（称为 Regions）提供云计算资源和服务。每个区域在物理和逻辑上都是独立的，由三个或更多 Availability Zones (AZs) 组成。可用区在地理上彼此接近，但在物理上是分开和隔离的。当您将工作负载分布于各个可用区和区域之间时，可以降低火灾、洪水、与天气相关的灾难、地震和人为错误等威胁的风险。

制定位置策略，以提供适合您的工作负载的高可用性。

期望结果：生产工作负载分布于多个可用区（AZ）或区域之间，以实现容错和高可用性。

常见反模式：

- 您的生产工作负载只存在于单个可用区中。
- 您在多可用区架构满足业务要求时实施多区域架构。
- 您的部署或数据变得不同步，这会导致配置偏差或数据复制不足。
- 当应用程序组件之间的韧性和多位置要求不同时，您未考虑这些组件之间的依赖关系。

建立此最佳实践的好处：

- 您的工作负载更能抵御意外事件，例如电源或环境控制故障、自然灾害、上游服务故障或影响可用区或整个区域的网络问题。
- 在启动特定 EC2 实例类型时，您可以访问更广泛的 Amazon EC2 实例清单，并降低出现 InsufficientCapacityExceptions（ICE）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

在区域的至少两个可用区（AZ）中部署和运行所有生产工作负载。

使用多个可用区

可用区是资源托管位置，它们在物理上彼此分开，以避免由于火灾、洪水和龙卷风等风险而导致的相关故障。每个可用区都有独立的物理基础设施，包括市电连接、备用电源、机修服务和网络连接。这种安排方式可将其中任何组件的故障限制在受影响的可用区内。例如，如果可用区范围的事件导致 EC2 实例在受影响的可用区中不可用，则您在其它可用区的实例仍保持可用。

尽管同一 AWS 区域中的可用区在物理上是分开的，但它们之间的距离足够近，可以提供高吞吐量、低延迟（个位数毫秒）的联网。您可以在可用区之间同步复制大多数工作负载的数据，而不会显著影响用户体验。这样一来，您便能以主动/主动或主动/备用配置使用区域中的可用区。

与工作负载关联的所有计算均应分布于多个可用区中。这包括 Amazon EC2 实例、AWS Fargate 任务和 VPC 连接的 AWS Lambda 函数。AWS 计算服务，包括 EC2 Auto Scaling、Amazon Elastic Container Service（ECS）和 Amazon Elastic Kubernetes Service（EKS），为您提供了跨可用区启动和管理计算的方法。将它们配置为根据需要在不同的可用区中自动替换计算以保持可用性。要将流量定向到可用的可用区，请在计算前放置一个负载均衡器，例如应用程序负载均衡器或网络负载均衡器。如果某个可用区受到损害，AWS 负载均衡器可以将流量重新路由到可用的实例。

您还应该为工作负载复制数据，并使其在多个可用区中可用。某些 AWS 托管式数据服务，例如 Amazon S3、Amazon Elastic File Service（EFS）、Amazon Aurora、Amazon DynamoDB、Amazon Simple Queue Service（SQS）和 Amazon Kinesis Data Streams，默认情况可在多个可用区中复制数据，并且可以抵御可用区损害。对于其它 AWS 托管式数据服务，例如 Amazon Relational Database Service（RDS）、Amazon Redshift 和 Amazon ElastiCache，您必须启用多可用区复制。启用后，这些服务会自动检测可用区损害，将请求重定向到可用的可用区，并在恢复后根据需要重新复制数据，而无需客户干预。熟悉您使用的每项 AWS 托管式数据服务的用户指南，以了解其多可用区的功能、行为和操作。

如果您使用的是自行管理的存储，例如 Amazon Elastic Block Store（EBS）卷或 Amazon EC2 实例存储，则必须自行管理多可用区复制。

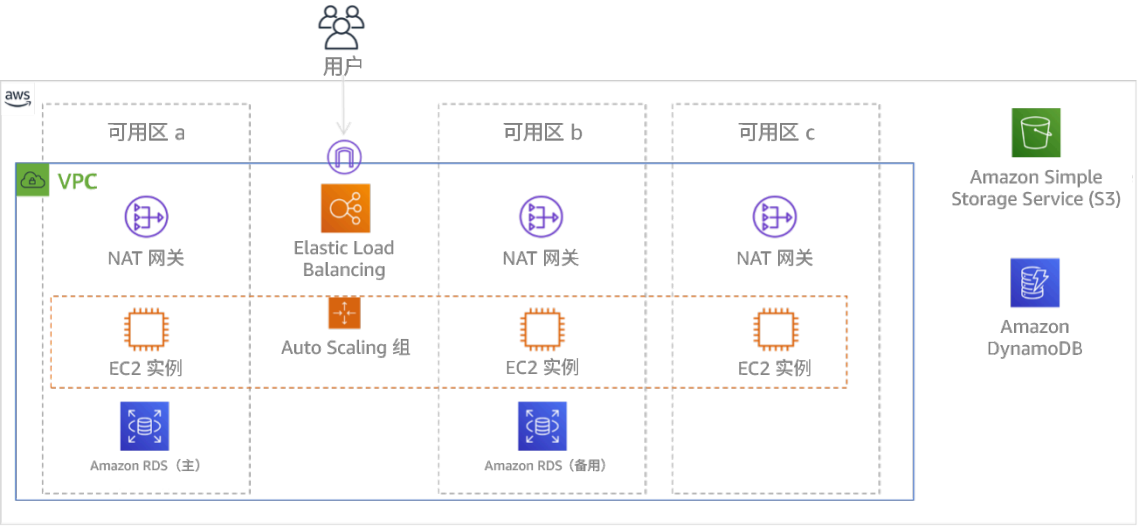


图 9：跨三个可用区部署的多层架构。请注意，Amazon S3 和 Amazon DynamoDB 始终会自动部署到多个可用区。而 ELB 也会被部署到所有三个区。

使用多个 AWS 区域

如果工作负载需要极高的韧性（如关键基础设施、与运行状况相关的应用程序或具有严格的客户或强制可用性要求的服务），您可能需要超出单个 AWS 区域所能提供的额外可用性。在这种情况下，您应该在至少两个 AWS 区域中部署和运行工作负载（假设数据驻留要求支持这么做）。

AWS 区域位于世界各地和多个大洲的不同地理区域。AWS 区域与单独的可用区相比，其物理分离和隔离程度甚至更高。除了少数例外情况，AWS 服务利用这种设计在不同区域之间完全独立运行（也称为区域服务）。AWS 区域服务的故障设计为不影响其它区域中的服务。

当您在多个区域中运行工作负载时，应考虑其它要求。由于不同区域中的资源彼此分开且独立，因此必须在每个区域中复制工作负载的组件。除计算服务和数据服务外，这还包括基本的基础设施，例如 VPC。

注意：在考虑多区域设计时，请验证工作负载能够在单个区域中运行。如果您在区域之间创建依赖关系，其中一个区域中的组件依赖于另一个区域中的服务或组件，则可能会增加故障风险并显著削弱可靠性状况。

为了简化多区域部署并保持一致性，AWS CloudFormation StackSets 可以跨多个区域复制整个 AWS 基础设施。AWS CloudFormation 还可以检测配置偏差，并在某个区域中的 AWS 资源不同步时通知您。许多 AWS 服务为重要的工作负载资产提供多区域复制。例如，例如，EC2 Image Builder 可以在每次构建后将 EC2 机器映像（AMI）发布到您使用的每个区域。Amazon Elastic Container Registry（ECR）可以将容器映像复制到选定的区域。

您还必须跨您所选的每个区域复制数据。许多 AWS 托管式数据服务提供跨区域复制功能，包括 Amazon S3、Amazon DynamoDB、Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon Elasticache 和 Amazon EFS。Amazon DynamoDB 全局表接受在任何受支持的区域中进行写入，并将在所有其它配置的区域间复制数据。对于其它服务，您必须指定一个主区域进行写入，因为其它区域包含只读副本。对于工作负载使用的每项 AWS 托管式数据服务，请参阅其用户指南和开发人员指南，以了解其多区域功能和局限性。请特别注意必须将写入定向到何处、事务处理功能和限制、如何执行复制以及如何监控区域之间的同步。

AWS 还能够非常灵活地将请求流量路由到区域部署。例如，可以使用 Amazon Route 53 配置 DNS 记录，来将流量定向到离用户最近的可用区域。或者，可以在主动/备用配置中配置 DNS 记录，在这种配置中，您将一个区域指定为主区域，仅当主区域变得运行状况不正常时，才回退到区域副本。您可以配置 Route 53 health checks 来检测运行状况不正常的端点并执行自动失效转移，此外，还可以使用 Amazon 应用程序恢复控制器（ARC）来提供高度可用的路由控制，以便根据需要手动重新路由流量。

即使您选择不在多个区域中运行来实现高可用性，也可以将多个区域视为灾难恢复（DR）策略的一部分。如果可能，请在辅助区域中以热备用 或指示灯 配置来复制工作负载的基础设施组件和数据。在此设计中，您从主区域复制基准基础设施，如 VPC、自动扩缩组、容器编排工具和其它组件，但您将备用区域中的可变大小组件（如 EC2 实例和数据库副本的数量）配置为最小可操作大小。您还可以安排从主区域到备用区域的连续数据复制。如果发生事件，则可以横向扩展或增加备用区域中的资源，然后将其提升为主区域。

实施步骤

1. 与业务利益相关者和数据驻留专家合作，来确定哪些 AWS 区域可用来托管您的资源和数据。
2. 与业务和技术利益相关者合作，来评估您的工作负载，并确定其韧性需求是否可以通过多可用区方法（单个 AWS 区域）来满足，或者是否需要多区域方法（如果允许多个区域）。使用多个区域可以提高可用性，但可能会增加复杂性和成本。评估时考虑以下因素：
   a. 业务目标和客户要求：如果在可用区或区域中发生影响工作负载的事件，允许有多少停机时间？评估恢复点目标，如 REL13-BP01 定义停机和数据丢失的恢复目标中所述。
   b. 灾难恢复（DR）要求：您想要确保自行抵御哪种潜在灾难？ 考虑数据丢失或长期不可用的可能性，影响范围涉及单个可用区到整个区域。如果您跨可用区复制数据和资源，而单个可用区持续出现故障，则可以在另一个可用区中恢复服务。如果您跨区域复制数据和资源，则可以在另一个区域中恢复服务。
3. 将计算资源部署到多个可用区中。

a. 在 VPC 中，在不同的可用区中创建多个子网。将每个子网配置为足够大，以容纳处理工作负载所需的资源，即使在发生事件期间也是如此。有关更多详细信息，请参阅 REL02-BP03 确保 IP 子网分配考虑扩展和可用性。

b. 如果您使用的是 Amazon EC2 实例，请使用 EC2 Auto Scaling 来管理实例。在创建自动扩缩组时，指定在上一步中选择的子网。

c. 如果您正在对 Amazon ECS 或 Amazon EKS 使用 AWS Fargate 计算，请在创建 ECS 服务、启动 ECS 任务或为 EKS 创建 Fargate 配置文件时，选择您在第一步中选择的子网。

d. 如果您使用的 AWS Lambda 函数需要在 VPC 中运行，请在创建 Lambda 函数时，选择您在第一步中选择的子网。对于任何没有 VPC 配置的函数，AWS Lambda 自动为您管理可用性。

e. 将流量定向器（例如负载均衡器）放在计算资源之前。如果启用了跨区域负载均衡，AWS 应用程序负载均衡器和网络负载均衡器会检测何时由于可用区受损而无法访问 EC2 实例和容器等目标，并将流量重新路由到正常运行的可用区中的目标。如果您禁用跨区域负载均衡，请使用 Amazon 应用程序恢复控制器（ARC）来提供可用区转移功能。如果您使用的是第三方负载均衡器或已实施了自己的负载均衡器，请为它们配置跨不同可用区的多个前端。

4. 跨多个可用区复制工作负载的数据。

a. 如果您使用的是 AWS 托管式数据服务，例如 Amazon RDS、Amazon ElastiCache 或 Amazon FSx，请研读其用户指南以了解其数据复制和韧性功能。必要时启用跨可用区复制和失效转移。

b. 如果您使用 AWS 托管式存储服务，例如 Amazon S3、Amazon EFS 和 Amazon FSx，请避免对需要高耐久性的数据使用单可用区或单区配置。对这些服务使用多可用区配置。查看相应服务的用户指南，以确定默认情况下是否启用了多可用区复制，或者是否必须启用它。

c. 如果您运行的是自行管理的数据库、队列或其它存储服务，请根据应用程序的说明或最佳实践安排进行多可用区复制。熟悉应用程序的失效转移过程。

5. 配置您的 DNS 服务以检测可用区受损，并将流量重新路由到运行状况正常的可用区。Amazon Route 53 与弹性负载均衡器结合使用时，可以自动执行此操作。还可以为 Route 53 配置失效转移记录，这些记录使用运行状况检查来响应仅具有正常运行的 IP 地址的查询。对于用于失效转移的任何 DNS 记录，请指定较短的生存时间（TTL）值（例如，60 秒或更短），以协助防止记录缓存阻碍恢复（Route 53 别名记录为您提供相应的 TTL）。

使用多个 AWS 区域时的额外步骤

1. 跨所选区域复制工作负载使用的所有操作系统（OS）和应用程序代码。如有必要，可以使用 Amazon EC2 Image Builder 等解决方案复制 EC2 实例使用的亚马逊机器映像（AMI）。使用 Amazon ECR 跨区域复制等解决方案复制存储在注册表中的容器映像。为用于存储应用程序资源的任何 Amazon S3 存储桶启用区域复制。

2. 将您的计算资源和配置元数据（例如，存储在 AWS Systems Manager Parameter Store 中的参数）部署到多个区域。使用前面步骤中介绍的相同过程，但请为您要用于工作负载的每个区域复制配置。使用 AWS CloudFormation 等基础设施即代码解决方案在区域之间统一复制配置。如果您在指示灯配置中使用辅助区域进行灾难恢复，您可以将计算资源的数量减少到最小值以节省成本，同时相应地增加恢复时间。
3. 将数据从主区域复制到辅助区域。
   a. Amazon DynamoDB 全局表提供数据的全局副本，可以从任何支持的区域写入这些副本。对于其它 AWS 托管式数据服务，例如 Amazon RDS、Amazon Aurora 和 Amazon Elasticache，您可以指定主（读/写）区域和副本（只读）区域。有关区域复制的详细信息，请参阅相应服务的用户和开发人员指南。
   b. 如果您运行的是自行管理的数据库，请根据应用程序的说明或最佳实践安排进行多区域复制。熟悉应用程序的失效转移过程。
   c. 如果工作负载使用 AWS EventBridge，则可能需要将选定的事件从主区域转发到辅助区域。为此，请将辅助区域中的事件总线指定为主区域中匹配事件的目标。
4. 考虑是否以及在多大程度上希望跨区域使用相同的加密密钥。平衡安全性和易用性的典型方法是使用区域范围的密钥来处理区域本地数据和身份验证，并使用全球范围的密钥来对在不同区域间复制的数据进行加密。AWS Key Management Service（KMS）支持 multi-region keys，以便安全地分发和保护跨区域共享的密钥。
5. 考虑使用 AWS Global Accelerator，通过将流量定向到包含正常运行的端点的区域，来提高应用程序的可用性。

资源

相关最佳实践：

- REL02-BP03 确保 IP 子网分配考虑扩展和可用性
- REL11-BP05 使用静态稳定性来防止双模态行为
- REL13-BP01 定义停机和数据丢失的恢复目标

相关文档：

- AWS 全球基础设施
- White paper: AWS Fault Isolation Boundaries
- Resilience in Amazon EC2 Auto Scaling
- Amazon EC2 Auto Scaling: Example: Distribute instances across Availability Zones

- How EC2 Image Builder works
- Amazon ECS 如何将任务放置在容器实例上（包括 Fargate）
- AWS Lambda 中的故障恢复能力
- Amazon S3：复制对象概述
- Amazon ECR 中的私有映像复制
- 全局表：使用 DynamoDB 的多区域复制
- Amazon Elasticache for Redis OSS: Replication across AWS 区域 using global datastores
- Amazon RDS 中的弹性
- 使用 Amazon Aurora Global Database
- AWS Global Accelerator Developer Guide
- Multi-Region keys in AWS KMS
- Amazon Route 53: Configuring DNS failover
- Amazon Application Recovery Controller (ARC) Developer Guide
- Sending and receiving Amazon EventBridge events between AWS 区域
- Creating a Multi-Region Application with AWS Services 系列博客文章
- 开启 AWS 灾难恢复（DR）架构，第一部分：云端恢复策略
- Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby

相关视频：

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications
- AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure

REL10-BP02 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数据中心内运行，则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

实施指导

如果由于技术约束无法使用将工作负载部署到多个位置的最佳实践，则必须实施其他的韧性路径。在这种情况下，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。如果这是工作负载韧性所需的必要组件，则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 中的数据可存储在 Amazon S3 中，进而可以跨多个可用区或 AWS 区域进行复制。

同理，对于 Amazon Redshift，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。所有集群节点在同一区域中配置。

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。如果已经在 AWS 托管中，则可以使用弹性灾难恢复将工作负载保护到其他可用区或区域。弹性灾难恢复在轻量级暂存区域中使用持续的块级复制，以便快速可靠地恢复本地应用程序和基于云的应用程序。

实施步骤

1. 实施自我修复。尽可能使用自动扩缩部署实例或容器。如果不能使用自动扩缩，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现自我修复自动化。
   - 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，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 ID 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。
   - 在无法使用自动扩缩或 EC2 恢复的情况下，请使用Amazon EC2 实例生命周期事件或 Amazon ECS 事件实现自我修复自动化。
     - 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
- 恢复实例。

- 服务自动扩缩
- What Is Amazon EC2 Auto Scaling?
- AWS Elastic Disaster Recovery

REL10-BP03 采用隔板架构来限制影响范围

实施隔板架构（也称为基于单元的架构），将工作负载中故障的影响限制于有限数量的组件。

期望结果：基于单元的架构使用多个独立的工作负载实例，其中每个实例称为单元。单元彼此独立，不与其他单元共享状态，并且处理整个工作负载请求的子集。这样减少了故障（例如，错误的软件更新）对单个单元及其正在处理的请求的潜在影响。如果某个工作负载使用 10 个单元来服务 100 个请求，则在发生故障时，总请求中有 90% 不会受故障的影响。

常见反模式：

- 让单元无限制地发展。
- 同时将代码更新或部署到所有单元。
- 在不同单元之间共享状态或组件（路由器层除外）。
- 向路由器层添加复杂的业务或路由逻辑。
- 没有尽量减少跨单元交互。

建立此最佳实践的好处：借助基于单元的架构，许多常见类型的故障控制在单元本身，从而实现了额外的故障隔离。若出现难以控制的故障类型（例如不成功的代码部署，或者是受损或调用特定故障模式的请求，也称为毒丸请求），这些故障边界可以提供韧性。

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

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。边界之外的组件不受故障影响。通过使用多个故障隔离边界，您可以限制对工作负载的影响。在 AWS 中，客户可以使用多个可用区和区域来实现故障隔离，但故障隔离的概念也可以扩展到工作负载的架构。

整个工作负载是分区的单元，按分区键进行划分。这个键需要与服务的粒度，或者与使用最少的跨单元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。单元路由层根据分区键将请求分发到单个单元，并向客户端提供单个端点。

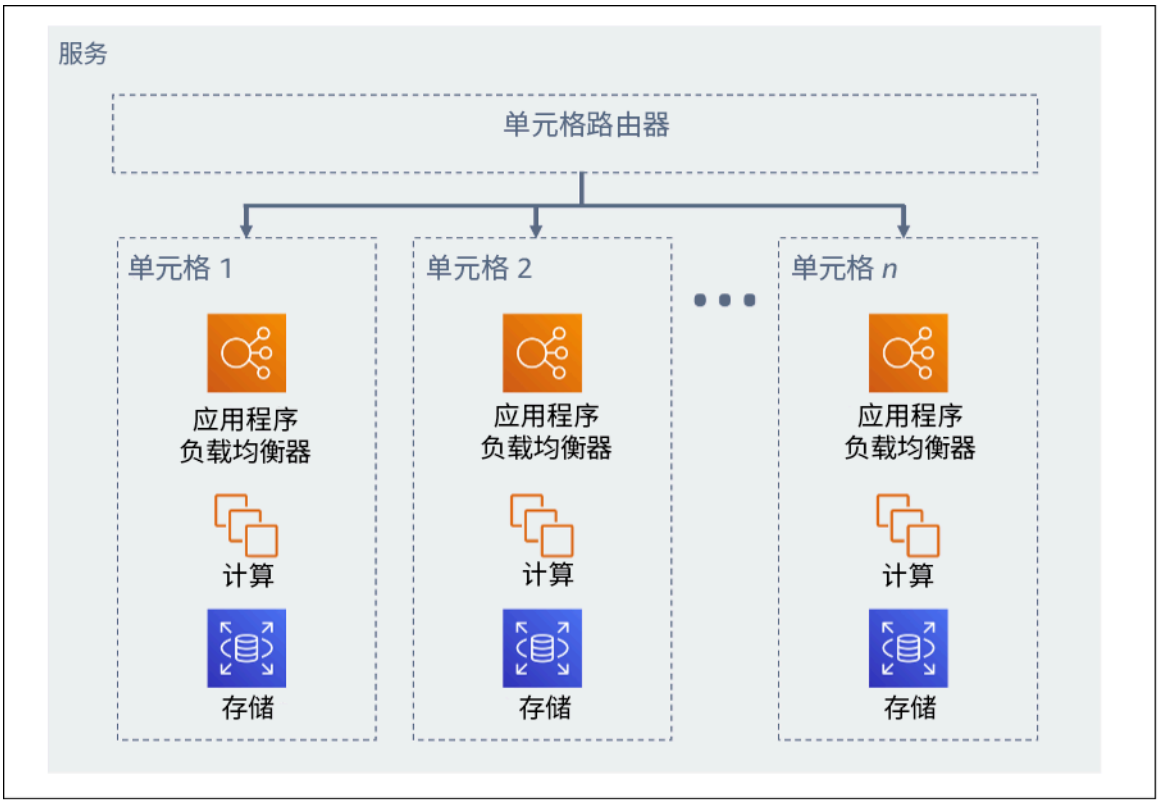


图 11：基于单元的架构

实施步骤

在设计基于单元的架构时，需要考虑几个设计注意事项：

1. 分区键：选择分区键时应考虑一些特别事项。
   - 分区键应与服务的粒度，或者与使用最少的跨单元交互来细分服务工作负载的自然方式保持一致。示例包括 `customer ID` 或 `resource 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. 持久单元映射：上游服务在其资源的生命周期内应只与单个单元交互。
   - 根据工作负载，可能需要使用单元迁移策略将数据从一个单元迁移到另一个单元。可能需要进行单元迁移的一种情况是：工作负载中的一个特定用户或资源变得太大，要求它具有专用的单元。
   - 单元之间不应该共享状态或组件。
   - 因此，应该避免或尽量减少跨单元交互，因为这些交互会在单元之间产生依赖关系，从而削弱故障隔离的改进。

3. 路由器层：路由层是单元之间的共享组件，因此无法遵循与单元相同的分隔策略。
    - 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单元，例如结合加密哈希函数和模块化算法，将分区键映射到单元。
    - 为避免产生多单元影响，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《Reliability, constant work, and a good cup of coffee》一文中所说，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. 单元大小：单元大小应具有上限，不得发展到超过这个值。
    - 应通过执行彻底的测试来确定大小上限，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单元，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：应利用多层韧性来防止出现不同的故障域。
    - 要实现韧性，应使用可构建防御层的方法。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-BP01 将工作负载部署到多个位置。
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单元。
    - 这有助于最大限度地减少因部署不当或人为错误而导致多个单元出现故障。有关更多详细信息，请参阅自动实现无需干预的安全部署。

资源

相关最佳实践：

- REL07-BP04 对工作负载进行负载测试
- REL10-BP01 将工作负载部署到多个位置

相关文档：

- Reliability, constant work, and a good cup of coffee
- AWS and Compartmentalization
- 使用随机分区进行工作负载隔离

- 自动实现无需干预的安全部署

相关视频：

- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small
- AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures (ARC338)
- Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)
- AWS Summit ANZ 2021 - Everything fails, all the time: Designing for resilience

设计工作负载来承受组件故障

在构建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韧性。

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面板而不是控制面板
- 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送通知
- REL11-BP07 构建产品以满足可用性目标和正常运行时间服务水平协议（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）需要成为检测和补救策略的一部分。

期望结果：独立监控工作负载的重要组成部分，对故障发生的时间和位置进行检测，再根据检测结果发出警报。

常见反模式：

- 未配置警报，因此不会在发生中断时进行通知。
- 虽然配置了警报，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）。
- 仅主动监控工作负载中面向客户的接口。
- 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：如果在所有层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。

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

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件后，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。服务列表应广泛而完整。

监控必须覆盖应用程序堆栈的所有层，包括应用程序、平台、基础设施和网络。

监控策略应考虑灰色故障的影响。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

- 监控间隔取决于必须以多快的速度恢复。恢复时间取决于恢复所需的时间，因此在确定收集频率时，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 实例和自动扩缩来说是否有必要。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。
  - 确定是否需要为 RDS 设置增强监控。增强监控使用 RDS 实例上的代理，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。

- 创建自定义指标来测量业务关键性能指标（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。可运行并模拟客户行为的综合事务测试（又称为“金丝雀测试”，但与金丝雀部署不同）是一项重要的测试流程。从不同的远程位置针对工作负载端点持续地运行此类测试。
- 创建跟踪用户体验的自定义指标。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，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 或电子邮件发送警报，并与自动扩缩功能结合使用来纵向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调查的问题的指示。
- 为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。
- 随时了解 AWS Health 的服务降级情况。通过 AWS 用户通知服务 创建要发送到电子邮件和聊天渠道且契合目标的 AWS Health 事件通知，并以编程方式通过 Amazon EventBridge 与监控和警报工具集成。

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送通知

相关文档：

- 使用 Amazon CloudWatch Synthetics 创建用户金丝雀
- 为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

- 使用跨区域跨账户的 CloudWatch 控制面板
- 使用跨区域跨账户的 X-Ray 跟踪
- Understanding availability

相关视频：

- Mitigating gray failures

相关示例：

- One Observability Workshop: Explore 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 受损），确保拥有适当的系统，可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。它们经构建为使用基于单元的隔离，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（如 Lambda 或 API Gateway）本质上是跨多个可用区部署的服务。其他 AWS 服务（如 EC2 和 EKS）需要特定的最佳实践设计，才能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，检查失效转移资源的运行状况，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故障。
- 未设立 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：可以构建更具韧性的系统，在遇到故障时，通过优雅降级和快速恢复来保持可靠性。

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

AWS 服务（例如弹性负载均衡和 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端点。Amazon Route 53、Amazon 应用程序恢复控制器 (ARC)、Amazon 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运行状况良好的位置。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状况良好的实例。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 容器组，您要选择部署到哪些可用区。对于某些设计，弹性负载均衡会提供解决方案来检测运行状况不佳的可用区内的实例，并将流量路由至运行状况良好的可用区。弹性负载均衡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、Amazon 应用程序恢复控制器、AWS Global Accelerator、Route 53 Private DNS for VPC 或 CloudFront 提供一种方法，来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状况良好的区域。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，由此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，确保可重复性。
- 配置恢复站点（例如第二个区域）来实施并测试失效转移设计。如有必要，可以临时配置测试资源来限制额外成本。
- 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）
- 制定计划来启动行动手册并加以演练。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 受损），确保拥有适当的系统，可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工作负载

相关文档：

- 设立 RO 和 RPO 目标
- 使用 Route 53 加权路由进行失效转移
- Disaster Recovery with Amazon Application Recovery Controller
- 带自动扩缩功能的 EC2
- EC2 部署 – 多可用区
- ECS 部署 – 多可用区
- Switch traffic using Amazon Application Recovery Controller
- 使用应用程序负载均衡器和失效转移的 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
- Guidance for Cross Region Failover and Graceful Failback on AWS
- 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

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。

期望结果：执行自动操作来修复检测到的故障。

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。
- 手动修复自动扩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。

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

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 控制面板自动修复任何故障。

使用计算集群通过负载均衡器访问的设计模式应利用自动扩缩组。弹性负载均衡（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带有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。该工作流程可以使用 AWS Systems Manager 自动执行，从而快速修复问题。

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，其可根据事件信息调用 AWS Lambda、Systems Manager Automation 或其他目标，对工作负载运行自定义修复逻辑。可以配置 Amazon EC2 Auto Scaling 来检查 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 会认为实例的运行状况不佳，继而启动替换实例。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

- 使用自动扩缩组在工作负载中部署层。自动扩缩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 EC2 实例上实施自动恢复，这些实例中部署的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。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，都会得到保留。使用 AWS OpsWorks 时，您可以在层级别配置 EC2 实例的自动修复。
- 当无法使用自动扩缩或自动恢复，或者自动恢复出故障时，可以使用 AWS Step Functions 和 AWS Lambda 实施自动恢复。当无法使用自动扩缩，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 服务中的状态更改。根据事件信息，该服务可以调用 AWS Lambda（或其他目标），在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)
- What is Amazon FSx for Lustre?
- What is Amazon FSx for Windows File Server?
- AWS OpsWorks: Using Auto Healing to Replace Failed Instances
- 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 Automation
- 韧性架构最佳实践

相关视频：

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

相关示例：

- Amazon RDS 失效转移讲习会

相关工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP04 恢复期间依赖于数据面板而不是控制面板

控制面板提供用于创建、读取、描述、更新、删除和列出（CRUDL）资源的管理 API，而数据面板则处理日常服务流量。在对可能影响韧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面板

操作，实现对服务的恢复、重新扩缩、恢复、修复或失效转移。在这些降级事件期间，数据面板操作应凌驾于任何活动之上。

例如，以下是所有控制面板操作：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，控制面板必须执行多项任务，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。控制面板往往是复杂的编排。

期望结果：当资源进入受损状态时，将流量从受损资源转移到运行状况良好的资源，能够自动或手动恢复系统。

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，依赖控制面板扩展操作来替换受损组件。
- 依靠广泛、多服务、多 API 控制面板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。

在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，控制面板会受到影响。依赖于大量使用控制面板进行修复，可能会增加恢复时间（RTO）和平均恢复时间（MTTR）。

实施指导

要限制数据面板操作，请评测每项服务，了解恢复服务所需的操作。

利用 Amazon 应用程序恢复控制器来转移 DNS 流量。这些功能会持续监控应用程序从故障中恢复的能力，让您能够跨多个 AWS 区域、可用区和本地部署控制应用程序恢复。

Route 53 路由策略使用控制面板，因此不要依赖控制面板进行恢复。Route 53 数据面板会回复 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% 可用性的服务水平协议（SLA）而设计。

用于创建、更新和删除 Route 53 资源的 Route 53 管理 API 和控制台是在控制面板上运行，而这些控制面板设计用于优先考虑在管理 DNS 时所需的强一致性和持久性。为了实现这一点，控制面板位于单个区域“美国东部（弗吉尼亚州北部）”中。虽然这两个系统都非常可靠，但控制面板不包含在 SLA 中。在极少数情况下，数据面板的韧性设计允许自身保持可用性，而控制面板做不到。对于灾难恢复和失效转移机制，使用数据面板功能可提供尽可能高的可靠性。

将计算基础设施设计为静态稳定，以避免在事故发生期间使用控制面板。例如，如果您使用的是 Amazon EC2 实例，请避免手动配置新实例或指示自动扩缩组添加实例作为响应。要获得最高级别的

韧性，请在集群中预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，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面板。但是，创建新函数、负载均衡器、API Gateway 或 DynamoDB 表是一项控制面板操作，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，数据面板操作允许访问数据。

有关数据面板、控制面板以及 AWS 如何构建服务来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面板，哪些位于控制面板。

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、自动修复设计或高可用性（HA）资源恢复计划。确定可能被视为控制面板操作的每个操作。

考虑将控制面板操作更改为数据面板操作：

- 自动扩缩（控制面板）到预扩展 Amazon EC2 资源（数据面板）
- Amazon EC2 实例扩展（控制面板）到 AWS Lambda 扩展（数据面板）
- 评测任何使用 Kubernetes 的设计以及控制面板操作的性质。在 Kubernetes 中，添加容器组是一项数据面板操作。操作应仅限于添加容器组而不是添加节点。使用预置过度的节点是限制控制面板操作的首选方法

考虑允许数据面板操作影响相同修复措施的其他方法。

- Route 53 记录更改（控制面板）或 Amazon 应用程序恢复控制器（数据面板）
- Route 53 运行状况检查，可获取更多自动更新

如果是任务关键型服务，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面板和数据面板操作。

- 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 相比，以及将流量路由到辅助区域（控制面板操作）
- 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（控制面板操作）

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：可帮助实现容错自动化的合作伙伴
- AWS Marketplace：可支持容错的产品
- Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control
- Amazon DynamoDB API（控制面板和数据面板）
- AWS Lambda 执行（拆分为控制面板和数据面板）
- AWS Elemental MediaStore Data Plane
- Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack
- Building highly resilient applications using Amazon Application Recovery Controller, Part 2: Multi-Region stack
- 使用 Amazon Route 53 创建灾难恢复机制
- What is Amazon Application Recovery Controller
- Kubernetes 控制面板和数据面板

相关视频：

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

相关示例：

- Introducing Amazon Application Recovery Controller
- Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control

- Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack
- Building highly resilient applications using Amazon Application Recovery Controller, Part 2: Multi-Region stack
- 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：

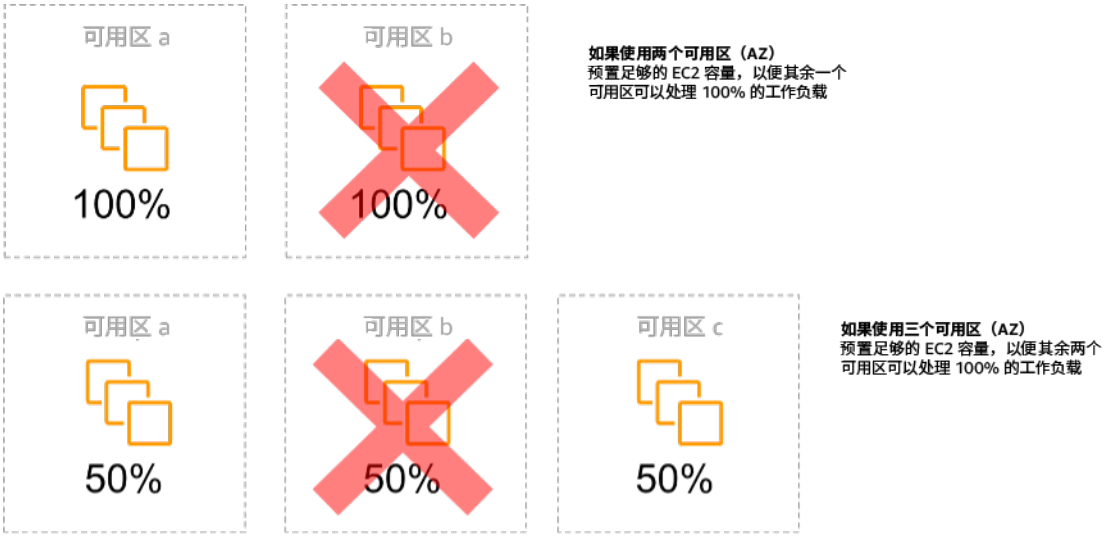
- 假设无论故障范围如何，始终可以预置资源。
- 尝试在故障期间动态获取资源。
- 在出现故障之前，没有跨可用区或跨区域预置足够的资源。
- 仅为计算资源考虑了静态稳定设计。

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。

在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，稳定的 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。弹性负载均衡或 Amazon Route 53 运行状况将进行检查，将负载从受损实例上移开。在流量转移以后，使用 AWS Auto Scaling 异步替换故障可用区的实例，并在运行状况良好的可用区中启动。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韧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，但它会明显改变工作负载的需求，而且很有可能导致故障。

评测关键工作负载，确定哪些工作负载需要这种韧性设计。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运行状况良好的资源，例如：
  - 跨区域 DNS 路由
  - MRAP Amazon S3 多区域路由
  - AWS Global Accelerator
  - Amazon Application Recovery Controller
- 配置数据库只读副本，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面板而不是控制面板

相关文档：

- Minimizing Dependencies in a Disaster Recovery Plan
- Amazon Builders' Library：使用可用区的静态稳定性
- Fault Isolation Boundaries
- 使用可用区的静态稳定性
- 多可用区 RDS
- Minimizing Dependencies in a Disaster Recovery Plan
- 跨区域 DNS 路由
- MRAP Amazon S3 多区域路由
- AWS Global Accelerator
- Amazon Application Recovery Controller
- 单区 Amazon S3
- Cross Zone Load Balancing

相关视频：

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

REL11-BP06 当事件影响可用性时发送通知

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，以便从根本上解决问题。

韧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，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不考虑灰色故障。
- 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）和平均修复时间（MTTR）。恢复事件通知还可确保您不会忽略不经常发生的问题。

在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 或电子邮件发送警报，并与 Auto Scaling 结合使用来横向扩展或缩减工作负载资源。

- 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，并关联任何自动化功能进行修复或通信。
- 随时了解 AWS Health 的服务降级情况。通过 AWS 用户通知服务 创建要发送到电子邮件和聊天渠道且契合目标的 AWS Health 事件通知，并以编程方式通过 Amazon EventBridge 与监控和警报工具集成。

资源

相关的 Well-Architected 最佳实践：

- 可用性定义

相关文档：

- 根据静态阈值创建 CloudWatch 警报
- 什么是 Amazon EventBridge？
- What is Amazon Simple Notification Service?
- 发布自定义指标
- 使用 Amazon CloudWatch 警报
- 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 构建产品以满足可用性目标和正常运行时间服务水平协议（SLA）

构建产品以满足可用性目标和正常运行时间服务水平协议（SLA）。如果您公开或私下同意可用性目标或正常运行时间 SLA，请确保架构和运营流程的设计支持它们。

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，实现业务成果。

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。
- 设计应用程序时不考虑韧性的责任共担模式。

建立此最佳实践的好处：根据关键韧性目标设计应用程序，有助于实现业务目标并满足客户期望。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。

在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

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韧性指标目标，以便进行监控并予以支持。不得在部署工作负载之后才设置或引出韧性指标。这些指标应该在设计阶段得到定义，有助于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韧性指标。这些指标可能与其他业务应用程序的指标不同。
- 减少依赖项可以对可用性产生积极影响。每个工作负载都应考虑自己的依赖项及其 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。
- 若可行，尽量考虑采用松耦合设计，让工作负载可以在依赖关系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。评估可确保任务关键型工作负载静态稳定性的设计。使用资源节约来提高工作负载中这些依赖项的可用性。
- 可观测性和检测能力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。
- 降低故障频率（提高 MTBF）、缩短故障检测时间（缩短 MTTD）和缩短修复时间（缩短 MTTR）是提高分布式系统可用性的三项因素。
- 设立并满足工作负载的韧性指标，这是所有有效设计的基础。这些设计必须在设计复杂性、服务依赖项、性能、扩展和成本之间作出权衡。

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？
  - 扩展、自动扩展、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？
  - 有哪些服务依赖项？
- 与利益相关方合作时，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。
- 设立了 SLA 目标后，优化架构来满足 SLA。
- 确定了满足 SLA 的设计后，实施侧重于减少 MTTD 和 MTTR 的运营更改、流程自动化和运行手册。
- 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-BP04 使用混沌工程测试韧性
- 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：

- Availability with redundancy
- 可靠性支柱 – 可用性
- 衡量可用性
- AWS Fault Isolation Boundaries
- 韧性的责任共担模式
- 使用可用区的静态稳定性

- AWS 服务水平协议（SLA）
- Guidance for Cell-based Architecture on AWS
- AWS infrastructure
- Advanced Multi-AZ Resilience Patterns whitepaper

相关服务：

- Amazon CloudWatch
- AWS Config
- AWS Trusted Advisor

测试可靠性

在为工作负载采用韧性设计来应对生产压力之后，测试是确保工作负载按设计运行并且提供预期韧性的唯一方式。

通过测试验证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对工作负载的可靠性造成影响。测试工作负载的韧性，有助于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 测试可扩展性和性能要求
- REL12-BP04 使用混沌工程测试韧性
- REL12-BP05 定期进行 GameDay 活动

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，对并不十分了解的故障场景实现一致且及时的响应。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，以便有效采取被动措施。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流程。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。

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

- 使用行动手册来发现问题。行动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（即执行事后分析）。
  - 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，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调用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 自动执行运营行动手册
    - AWS Systems Manager Run Command
    - AWS Systems Manager Automation
    - 什么是 AWS Lambda ？
    - 什么是 Amazon EventBridge？
    - 使用 Amazon CloudWatch 警报

资源

相关文档：

- AWS Systems Manager Automation

- AWS Systems Manager Run Command
- 使用 AWS Systems Manager 自动执行运营行动手册
- 使用 Amazon CloudWatch 警报
- 使用金丝雀（Amazon CloudWatch Synthetics）
- 什么是 Amazon EventBridge？
- 什么是 AWS Lambda ？

相关示例：

- 根据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，限制或防止再次发生同类事件。制定程序，以便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。

评测为什么现有测试找不到问题。如果还没有，为此案例增设测试。

期望结果：您的团队采用一致且一致的方法来处理事后分析。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，导致根本原因相同或相似的事件反复发生

建立此最佳实践的好处：如果其他工作负载实施了相同的成因，执行事后分析并共享分析结果可帮助缓解这些工作负载的故障风险，让它们能够在事件发生之前实施缓解或自动恢复措施。

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成因。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审查会议，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，您需要基于业务环境在各个支柱之间做出权衡。这些业务决策可以推动您的工程优先事务。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？
    - 操作项
    - 相关术语
- 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：

- Why you should develop a correction of error (COE)

相关视频：

- Amazon's approach to failing successfully
- AWS re:Invent 2021 - Amazon Builders' Library: Operational Excellence at Amazon

REL12-BP03 测试可扩展性和性能要求

使用负载测试等技术来验证工作负载是否满足扩展和性能要求。

在云中，可以按需为工作负载创建生产规模测试环境。可以使用云来预置一个与预期生产环境非常接近的测试环境，而不是依赖于缩减的测试环境（这可能会导致对生产行为的预测不准确）。此环境有助于您更准确地模拟应用程序面临的现实世界条件来进行测试。

除了性能测试工作外，还务必验证基础资源、扩展设置、服务配额和韧性设计在负载之下是否按预期运行。这种整体方法验证应用程序可根据需要可靠地扩展和执行，即使在最苛刻的条件下也不例外。

期望结果：即使在峰值负载下，工作负载也会保持其预期的行为。您可以主动解决随着应用程序发展和演变而可能出现的任何与性能有关的问题。

常见反模式：

- 您使用的测试环境与生产环境不太匹配。
- 您将负载测试视为单独的一次性活动，而不是部署持续集成（CI）管道不可或缺的部分。

- 您没有定义明确且可衡量的性能要求，例如响应时间、吞吐量和可扩展性目标。
- 您在不切实际或负载不足的情况下执行测试，并且无法针对峰值负载、突然激增和持续高负载进行测试。
- 您没有通过超出预期的负载限制来对工作负载进行压力测试。
- 您使用了不充分或不适当的负载测试和性能分析工具。
- 您缺乏全面的监控和警报系统来跟踪性能指标和检测异常情况。

建立此最佳实践的好处：

- 负载测试有助于您在系统投入生产之前识别其潜在的性能瓶颈。在模拟生产级流量和工作负载时，您可以确定系统可能难以处理负载的领域，例如响应时间慢、资源限制或系统故障。
- 当您在各种负载条件下测试系统时，可以更好地了解支持工作负载所需的资源需求。这些信息有助于您在资源分配方面做出明智的决策，并防止资源过度配置或配置不足。
- 要识别潜在的故障点，您可以观察工作负载在高负载条件下的性能。这些信息有助于您通过酌情实施容错机制、失效转移策略和冗余措施，来提高工作负载的可靠性和韧性。
- 您可以尽早发现并解决性能问题，这有助于避免系统中断、响应时间缓慢和用户不满意所带来的代价高昂的后果。
- 在测试期间收集的详细性能数据和分析信息有助于您排查生产环境中可能出现的与性能相关的问题。这可以加快事件响应和解决速度，从而减少对用户和组织运营的影响。
- 在某些行业，主动性能测试有助于工作负载达到合规标准，从而降低受处罚或出现法律问题的风险。

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

第一步是定义全面的测试策略，该策略涵盖扩展和性能要求的各个方面。首先，根据业务需求（例如吞吐量、延迟直方图和错误率），明确定义工作负载的服务级别目标（SLO）。接下来，设计一套测试来模拟各种负载场景，范围涵盖从平均使用量到突然激增和持续的峰值负载，并验证工作负载的行为是否符合 SLO。这些测试应自动执行，并集成到持续集成和部署管道中，以便在开发过程的早期阶段发现性能回归情况。

要有效地测试扩展和性能，请投资购买正确的工具和基础设施。这包括可以生成真实用户流量的负载测试工具、用于识别瓶颈的性能分析工具以及用于跟踪关键指标的监控解决方案。重要的是，您应该验证测试环境在基础设施和环境条件方面是否与生产环境紧密匹配，以使您的测试结果尽可能准确。为了更轻松且可靠地复制和扩展类似于生产环境的设置，请使用基础设施即代码和基于容器的应用程序。

扩展和性能测试是一个持续的过程，而不是一次性活动。实施全面的监控和警报来跟踪应用程序在生产环境中的性能，并使用这些数据来不断完善测试策略和优化工作。定期分析性能数据来识别新出现的问题，测试新的扩展策略，并实施优化以提高应用程序的效率和可靠性。当您采用迭代方法并不断从生产数据中学习时，可以验证应用程序是否能够适应不断变化的用户需求，并随着时间的推移保持韧性和最佳性能。

实施步骤

1. 制定明确且可衡量的性能要求，例如响应时间、吞吐量和可扩展性目标。这些要求应基于工作负载的使用规律、用户预期和业务需求。
2. 选择并配置负载测试工具，该工具可以准确地模仿生产环境中的负载规律和用户行为。
3. 设置与生产环境（包括基础设施和环境条件）紧密匹配的测试环境，来提高测试结果的准确性。
4. 创建涵盖各种场景的测试套件，范围从平均使用规律到峰值负载、快速激增和持续的高负载。将测试集成到持续集成和部署管道中，以便在开发过程的早期阶段发现性能回归情况。
5. 开展负载测试来模拟真实的用户流量，并了解应用程序在不同负载条件下的行为。要对应用程序进行压力测试，请超出预期负载并观察其行为，例如响应时间降级、资源耗尽或系统故障，这有助于确定应用程序的突破点并为扩展策略提供信息。通过逐步增加负载来评估工作负载的可扩展性，并衡量性能影响，来确定扩展限制并规划未来的容量需求。
6. 实施全面的监控和警报，来跟踪性能指标，检测异常，并在超过阈值时启动扩展操作或通知。
7. 持续监控和分析性能数据，来确定需要改进的领域。对测试策略和优化工作进行迭代。

资源

相关最佳实践：

- REL01-BP04 监控和管理配额
- REL06-BP01 为工作负载监控全部组件（生成）
- REL06-BP03 发送通知（实时处理和报警）

相关文档：

- 加载测试应用程序
- AWS 上的分布式负载测试
- 应用程序性能监控
- Amazon EC2 Testing Policy

相关示例：

- Distributed Load Testing on AWS (GitHub)

相关工具：

- Amazon CodeGuru Profiler
- Amazon CloudWatch RUM
- Apache JMeter
- K6
- Vegeta
- Hey
- ab
- wrk
- AWS 上的分布式负载测试

REL12-BP04 使用混沌工程测试韧性

在生产环境中或尽可能接近生产的环境中定期运行混沌试验，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韧性测试之外，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韧性。将混沌工程和韧性测试结合起来，这可以让您提升信心，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：

- 进行韧性设计，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实验。
- 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韧性，这可以让您提升信心，相信韧性设计的恢复程序会在真正发生故障的情况下发挥作用。

在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。其可让团队从故障中学习，观察、测量和提高工作负载的韧性，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，混沌工程可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**
>
> 混沌工程是在系统上进行实验的学科，目的是建立对系统抵御生产环境中失控条件的能力以及信心。– 混沌工程原则

如果系统能够经受住这些干扰，则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。

为了确保工作负载能够经受住组件故障，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。

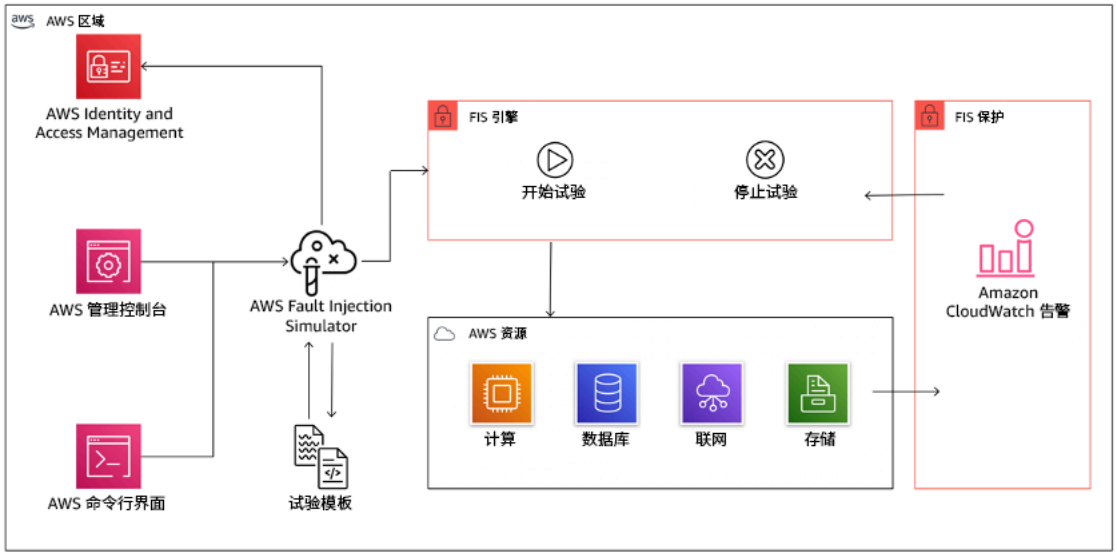
为了验证因间歇性网络中断而引发的外部依赖项的回退或失效转移机制，组件应通过在指定时间段（从几秒到几小时不等）内阻止对第三方提供商的访问来模拟此类事件。

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 GameDay 活动期间使用的一个不错选择。该服务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。由于该服务已与 Amazon CloudWatch 警报集成，您可以设置停止条件作为防护机制，在实验导致意外影响时回滚。



AWS Fault Injection Service 与 AWS 资源集成，让您能够为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），以及商用工具（例如 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，让您能够在多个工具之间协调故障注入工作流程。例如，您可以使用 Chaos Mesh 或 Litmus 故障对容器组的 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

1. 确定哪些故障要用于实验。

   评测工作负载的设计是否具有韧性。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项、以往事件、已知问题以及合规性要求的风险。列出每个旨在保持韧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《Operational Readiness Review》白皮书，了解如何创建流程来防止以往事件再次发生。故障模式与影响分析（FMEA）流程提供了一个框架，可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。
2. 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。要评测优先级，请考虑故障的频率和故障对整体工作负载的影响。

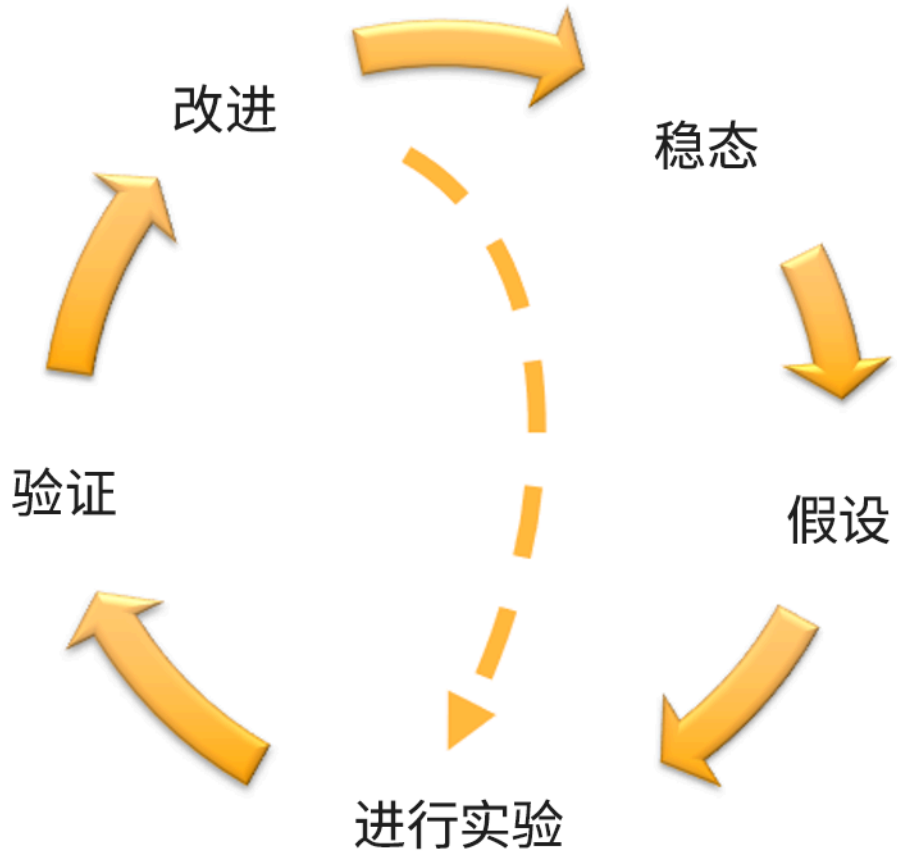
考虑给定故障的频率时，请分析此工作负载的以往数据（如有）。如果没有以往数据，则使用在类似环境中运行的其他工作负载的数据。

考虑给定故障的影响时，故障的范围越大，影响通常也越大。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，您要确定访问故障以及节流访问和延迟插入等实验的优先级。

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对哪些故障进行实验，以及开发新的故障注入实验的顺序。

3. 对于执行的每项实验，请遵循下图中的混沌工程和持续韧性飞轮。



Adrian Hornsby 采用科学方法制作的混沌工程和持续韧性飞轮。

a. 将稳定状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，则显示为稳定状态。因此，定义稳定状态之前，请验证工作负载是否运行状况良好。稳定状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。稳定状态是您将在实验期间观察到的基线，如果下一步中定义的假设结果不符合预期，则会突出显示异常。

例如，可以将某个支付系统的稳定状态定义为处理 300 TPS，成功率为 99%，且往返时间为 500 毫秒。

b. 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来保持稳定状态。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定状态，因为该工作负载在设计时就有特定缓解措施。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########，以此维持#########。

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 100 毫秒的时间内继续处理 99% 的请求（稳定状态）。Amazon EKS 节点将在五分钟内恢复，容器组将在实验开始后八分钟内得到调度并处理流量。警报将在三分钟内发出。
- 如果单个 Amazon EC2 实例发生故障，订单系统的弹性负载均衡运行状况检查会让弹性负载均衡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%（稳定状态）。
- 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（稳定状态）。

c. 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。对已知发生了故障的工作负载进行实验，并不会为带来新的见解。您应该仔细规划实验，明确影响范围，并提供一种可在出现意外动荡时应用的回滚机制。如果尽职调查表明工作负载应该能经受住

实验，才继续这项实验。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，可在实验引入了意外动荡的情况下回滚实验。要了解更多使用 AWS FIS 的实验，另请参阅 Resilient and Well-Architected Apps with Chaos Engineering 实验室。此外，AWS Resilience Hub 将分析工作负载，并创建您可以选择在 AWS FIS 中实施和执行的实验。

> **Note**
>
> 对于每项实验，要清楚地了解其范围及影响。建议首先在非生产环境中模拟故障，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定状态指标，以及针对您要注入故障的组件的指标。综合监控（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，确认停止条件的阈值是否在实验期间按预期激活，以及是否具有可观测性来捕获异常，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，工作负载的韧性设计会自我修复。但一些故障设计或失败实验可能会让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）。后置操作可将目标返回到操作运行之前的状态。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。

d. 验证假设。

混沌工程原则为如何验证工作负载的稳定状态提供了以下指导：

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，因为此类错误是工作负载客户端会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 上包含一个综合监控（也称为用户金丝雀）。

e. 改进工作负载设计，提高韧性。

如果未保持稳定状态，则调查如何改进工作负载设计来缓解故障，并应用 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），确定更改的效果。如果验证步骤表明假设成立，则工作负载将处于稳定状态，循环将继续。

4. 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 实验的实验室让您能够动手实践。

故障注入实验也是 GameDay 活动的一部分（请参阅 REL12-BP05 定期进行 GameDay 活动）。GameDay 活动会模拟故障或事件，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。

5. 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？
- 混沌工程原则
- Chaos Engineering: Planning your first experiment
- Resilience Engineering: Learning to Embrace Failure
- 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: Testing resiliency using chaos engineering (ARC316)
- AWS re:Invent 2019: Improving resiliency with chaos engineering (DOP309-R1)
- AWS re:Invent 2019: Performing chaos engineering in a serverless world (CMY301)

相关工具：

- AWS Fault Injection Service
- AWS Marketplace：Gremlin 混沌工程平台
- Chaos Toolkit
- Chaos Mesh
- Litmus

REL12-BP05 定期进行 GameDay 活动

安排 GameDay 来定期练习旨在应对影响工作负载的事件和损害的过程。让负责处理生产场景的团队参与进来。这些练习有助于强制实施相关措施，来防止生产事件对用户造成影响。当您在现实条件下实践响应过程时，可以在实际事件发生之前发现并解决任何差距或弱点。

GameDay 活动会模拟类似于生产的环境中的事件，以便测试系统、流程和团队的响应。其目的是执行团队在实际发生事件时会执行的相同操作。这些练习有助于您了解可以从哪些方面作出改进，并有助于培养组织在处理各种事件和损害方面的经验。这些练习应该定期开展，这样，团队就知道如何建立根深蒂固的应对习惯。

GameDay 可让团队做好准备，以便更充满信心地处理生产事件。经过良好练习的团队更有能力快速检测和应对各种场景。这可以显著改善就绪状态和韧性态势。

期望结果：您在一致、有计划的基础上运行韧性 GameDay。这些 GameDay 被视为业务运营中正常和预期的组成部分。您的组织已经建立了备灾文化，当出现生产问题时，团队已经做好了充分的准备，可以有效地做出响应，高效地解决问题并减轻对客户的影响。

常见反模式：

- 您记录过程，但从不练习这些过程。
- 您不让业务决策者参与测试练习。
- 您开展了 GameDay，但没有通知所有相关的利益相关者。
- 您只关注技术故障，但不涉及业务利益相关者。
- 您未将从 GameDay 中吸取的经验教训纳入恢复过程。
- 您将失败或错误归咎于团队。

建立此最佳实践的好处：

- 增强响应技能：在 GameDay，团队在模拟的事件中练习其职责并测试其沟通机制，从而在生产环境中做出更加协调和高效的响应。
- 识别和解决依赖关系：复杂的环境通常涉及各种系统、服务和组件之间错综复杂的依赖关系。GameDay 有助于您识别和解决这些依赖关系，并验证运行手册过程是否正确涵盖了关键系统和服务，以及是否可以及时纵向扩展或恢复这类系统和服务。
- 培养韧性文化：GameDay 有助于培养组织内部的韧性思维。当您让跨职能团队和利益相关者参与时，这些练习可以提高整个组织对韧性重要性的认识、协作和共同理解。
- 持续改进和适应：定期的 GameDay 有助于您不断评测和调整韧性策略，从而使这些策略在不断变化的环境中保持相关性和有效性。
- 增强对系统的信心：成功的 GameDay 有助于您树立信心，确信系统能够承受中断并从中恢复。

在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

设计并实施了必要的韧性措施后，请开展 GameDay 来验证生产中的一切是否按计划进行。GameDay，尤其是第一个 GameDay，应让所有团队成员都参与，并应事先向所有利益相关者和参与者告知日期、时间和模拟场景。

在 GameDay 期间，参与的团队会根据规定的过程模拟各种事件和潜在的场景。参与者密切监控和评测这些模拟事件的影响。如果系统按设计运行，则应激活自动检测、扩展和自我修复机制，且对用户几乎没有影响。如果团队观察到任何负面影响，他们就会回滚测试，并通过相应运行手册中记载的自动手段或手动干预来纠正已发现的问题。

要持续提高韧性，记录和吸取经验教训至关重要。该过程是一个反馈循环，它系统化地从 GameDay 捕获见解，并使用这些见解来增强系统、流程和团队能力。

为协助您重现系统组件或服务可能意外出现故障的现实场景，请将模拟故障作为 GameDay 练习注入。团队可以在受控的环境中测试其系统的韧性和容错能力，并模拟其事件响应和恢复流程。

借助 AWS，可以使用基础设施即代码，通过生产环境的副本来开展 GameDay。通过此过程，可以在与生产环境非常相似的安全环境中进行测试。考虑使用 AWS Fault Injection Service来创建不同的故障场景。使用诸如 Amazon CloudWatch 和 AWS X-Ray 之类的服务来监控 GameDay 期间的系统行为。使用 AWS Systems Manager 来管理和运行行动手册，并使用 AWS Step Functions 来编排重复出现的 GameDay 工作流程。

实施步骤

- 制定 GameDay 计划：制定结构化计划来定义 GameDay 的频率、范围和目标。让关键利益相关者和主题专家参与规划和实施这些练习。
- 为 GameDay 做好准备：
  1. 确定主要的业务关键服务，这些服务是 GameDay 的重点。对支持这些服务的人员、流程和技术进行编目和映射。
  2. 制定 GameDay 的日程，让相关团队做好参与事件的准备。准备好自动化服务来模拟计划的场景并运行相应的恢复流程。诸如 AWS Fault Injection Service、AWS Step Functions 和 AWS Systems Manager 等 AWS 服务有助于您自动实施 GameDay 的各个方面，例如注入故障和启动恢复操作。
- 运行模拟：在 GameDay，运行计划的场景。观察并记录人员、流程和技术对模拟事件的反应。
- 开展练习后回顾：GameDay 结束后，召开回顾会议来回顾所吸取的教训。确定需要改进的领域以及改善运营韧性所需的任何措施。记录您的调查发现，并跟踪任何必要的更改，来增强韧性策略和完成准备工作。

资源

相关最佳实践：

- REL12-BP01 使用行动手册调查故障
- REL12-BP04 使用混沌工程测试韧性
- OPS04-BP01 确定关键绩效指标
- OPS07-BP03 使用运行手册执行程序
- OPS10-BP01 使用流程来管理事件、意外事件和问题

相关文档：

- 什么是 AWS GameDay？

相关视频：

- AWS re:Invent 2023 - Practice like you play: How Amazon scales resilience to new heights

相关示例：

- AWS Workshop - Navigate the storm: Unleashing controlled chaos for resilient systems
- Build Your Own Game Day to Support Operational Resilience

灾难恢复（DR）计划

拥有适当的备份和冗余工作负载组件是灾难恢复策略的开始。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业务价值。

可用性和灾难恢复都仰赖于相同的最佳实践，例如监控故障、部署到多个位置以及自动失效转移。不过，可用性侧重的是工作负载的组成部分，灾难恢复侧重的是整个工作负载的离散副本。灾难恢复的目标侧重于灾难发生后的恢复时间，与可用性的目标不同。

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 管理灾难恢复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

故障可通过多种方式影响您的业务。首先，故障可能导致服务中断（停机时间）。其次，故障可能导致数据丢失、不一致或过时。为了指导您如何应对故障和从故障中恢复，请为每个工作负载定义恢复时间目标（RTO）和恢复点目标（RPO）。恢复时间目标（RTO）是指服务中断和恢复服务之间可接受的最大延迟。恢复点目标（RPO）是指在上一个数据恢复点之后可接受的最长时间。

期望结果：根据技术考虑和业务影响，每个工作负载都有指定的 RTO 和 RPO。

常见反模式：

- 您尚未指定恢复目标。
- 您选择任意的恢复目标。
- 您选择的恢复目标过于宽松并且不符合业务目标。
- 您尚未评估停机时间和数据丢失的影响。

- 您选择不切实际的恢复目标，如零恢复时间或零数据丢失，这对工作负载配置而言可能无法实现。
- 您选择的恢复目标比实际业务目标更苛刻。这会迫使实施恢复的成本和复杂度超出工作负载所需。
- 您选择的恢复目标与相关工作负载的恢复目标不兼容。
- 您没有考虑监管和合规要求。

建立此最佳实践的好处：在为工作负载设置 RTO 和 RPO 时，可以根据业务需求制定明确且可衡量的恢复目标。一旦设定了这些目标，就可以创建专为实现这些目标而量身定制的灾难恢复（DR）计划。

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

构造一个矩阵或工作表，来协助指导灾难恢复规划。在矩阵中，创建不同的工作负载类别或层级，所依据的是这些类别或层级的业务影响（例如严重、高、中和低），以及每个工作负载类别或层级关联的目标 RTO 和 RPO。以下矩阵提供了您可以遵循的示例（请注意，RTO 和 RPO 值可能有所不同）：

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

灾难恢复矩阵示例

对于每个工作负载，调查和了解停机时间和数据丢失对业务的影响。影响通常会随着停机时间和数据丢失而增加，但影响的形式可能会因工作负载类型而异。例如，长达一小时的停机时间可能影响很小，但在一小时之后，影响可能会迅速加剧。影响可能以多种形式呈现，包括财务影响（如收入损失）、声誉影响（包括失去客户信任）、运营影响（如错过工资发放或生产率下降）和监管风险。完成后，将工作负载分配到相应的层级。

分析故障所产生的影响时，请考虑以下问题：

1. 在对业务产生不可接受的影响之前，工作负载可以保持不可用的最长时间是多少？

2. 工作负载中断将对业务产生多大影响以及会产生什么样的影响？ 考虑各种影响，包括财务、声誉、运营和监管。
3. 在对业务造成不可接受的影响之前，可以丢失或无法恢复的最大数据量是多少？
4. 能否从其它来源重新创建丢失的数据（也称为派生的 数据）？ 如果是，还要考虑用于重新创建工作负载数据的所有源数据的 RPO。
5. 此工作负载所依赖的工作负载（下游）的恢复目标和可用性期望是什么？ 根据工作负载下游依赖关系的恢复能力，工作负载的目标必须是可实现的。考虑可能的下游依赖关系解决办法或缓解措施，以提高此工作负载的恢复能力。
6. 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？ 上游工作负载目标可能要求此工作负载具有比最初看起来更严格的恢复能力。
7. 根据事件的类型，是否有不同的恢复目标？ 例如，根据事件影响的是一个可用区还是整个区域，您可能有不同的 RTO 和 RPO。
8. 恢复目标在一年中的某些事件或时期是否会发生变化？ 例如，在假日购物季、体育赛事、特别促销和新产品发布期间，您可能有不同的 RTO 和 RPO。
9. 恢复目标如何与您可能拥有的任何业务线和组织灾难恢复策略保持一致？
10. 是否需要考虑法律或合同后果？ 例如，根据合同，您是否有义务提供具有给定 RTO 或 RPO 的服务？ 不履行这些义务会受到什么处罚？
11. 您是否需要维护数据完整性来满足监管或合规性要求？

以下工作表有助于您评估每项工作负载。您可以修改此工作表来满足具体的需求，例如添加附加问题。

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

实施步骤

1. 确定业务利益相关方和负责每个工作负载的技术团队，并与他们互动。
2. 对组织中的工作负载影响创建严重性类别或层级。类别示例包括：严重、高、中和低。对于每个类别，请选择反映您的业务目标和要求的 RTO 和 RPO。
3. 将您在上一步创建的影响类别之一分配给每个工作负载。要决定工作负载如何映射到某个类别，请考虑工作负载对业务的重要性，以及中断或数据丢失的影响，并使用上述问题来提供指导。这将为每个工作负载生成一个 RTO 和 RPO。
4. 考虑上一步中确定的每个工作负载的 RTO 和 RPO。让工作负载的业务和技术团队参与进来，以确定是否应调整目标。例如，业务利益相关者可能确定需要更严格的目标。或者，技术团队可能决定应修改目标，以使这些目标能够在可用的资源和技术限制下得以实现。

资源

相关最佳实践：

- REL09-BP04 定期执行数据恢复以验证备份完整性和流程

- REL12-BP01 使用行动手册调查故障
- 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

相关文档：

- AWS Architecture Blog: Disaster Recovery 系列博客文章
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- Managing resiliency policies with AWS Resilience Hub
- APN 合作伙伴：可帮助进行灾难恢复的合作伙伴
- AWS Marketplace：可用于灾难恢复的产品

相关视频：

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications
- 上工作负载的灾难恢复AWS

REL13-BP02 使用定义的恢复策略来实现恢复目标

定义可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：每个工作负载都有已定义且实施的灾难恢复策略，可让该工作负载实现灾难恢复目标。工作负载之间的灾难恢复策略利用可重用模式（例如前面所述的策略）。

常见反模式：

- 为具有类似灾难恢复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灾难恢复策略。
- 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可改善团队之间的知识共享情况，并在他们自己的工作负载上实施灾难恢复。

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、实施和测试的灾难恢复策略，发生灾难时就不太可能实现恢复目标。

实施指导

灾难恢复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，如 REL13-BP01 定义停机和数据丢失的恢复目标中所述。

跨单个 AWS 区域内的多个可用区（AZ）的灾难恢复策略可以缓解火灾、洪水和重大停电等灾难事件造成的影响。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灾难恢复策略。

在跨多个区域构建灾难恢复策略时，您应该选择以下某个策略。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。

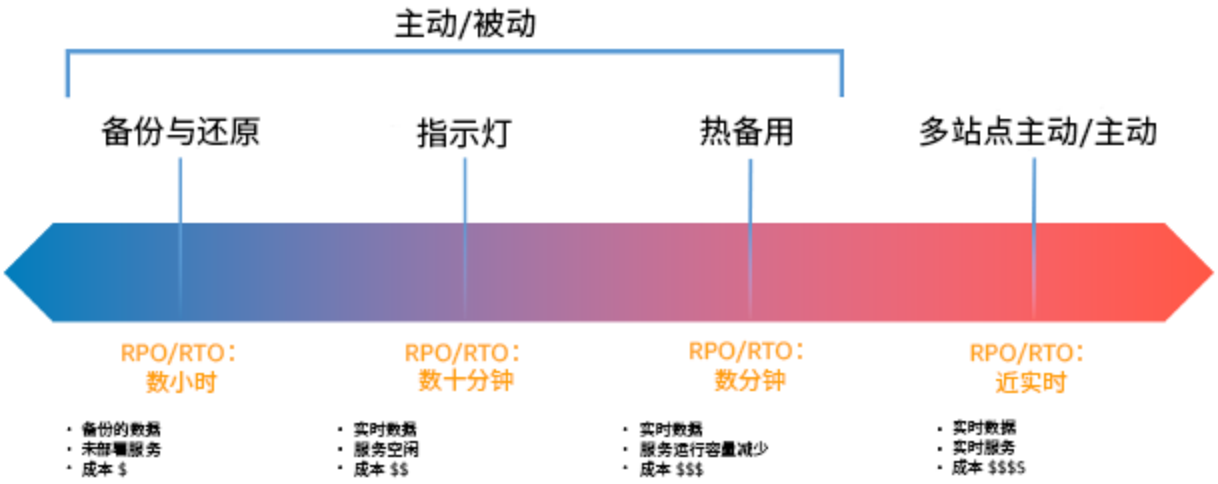


图 17：灾难恢复（DR）策略

- 备份和还原（RPO 以小时为单位，RTO 为 24 小时或更短时间为单位）：将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（PITR），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 以数十分钟为单位）：在恢复区域中预置核心工作负载基础设施的副本。将数据复制到恢复区域并在其中创建数据备份。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。

- 温备用（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，系统会快速纵向扩展来处理生产负载。温备用的规模越大，RTO 和控制面板依赖度就越低。当完全扩展时，这称为热备用。
- 多区域（多站点）主动-主动（RPO 接近于零，RTO 可能为零）：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，因为这种情况很复杂。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**
>
> 指示灯和温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，而温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，而温备用只需要您纵向扩展（所有内容都已部署并运行）。根据 RTO 和 RPO 需求在两者之间进行选择。
>
> 如果需要考虑成本，并且希望实现与温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），此类解决方案会采用指示灯方法并提供改进的 RPO 和 RTO 目标。

实施步骤

1. 确定可满足此工作负载恢复要求的灾难恢复策略。

   选择灾难恢复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。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，企业已经确定了自己允许的最大 RTO 以及可在服务恢复策略上花费的费用限额。考虑到企业目标，指示灯或温备用这样的灾难恢复策略将同时满足 RTO 和成本标准。

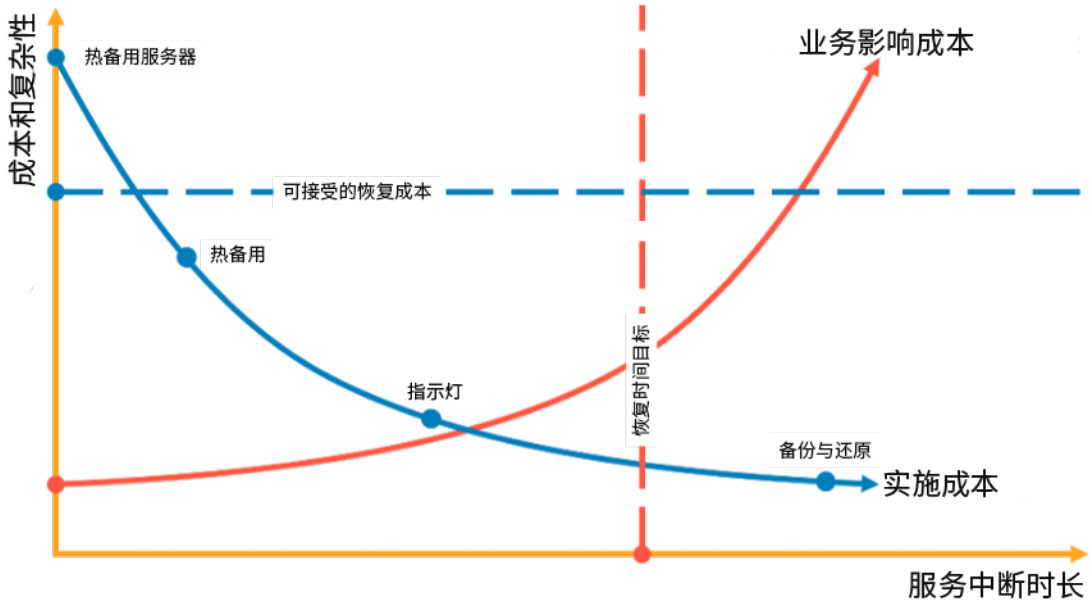


图 18：根据 RTO 和成本选择灾难恢复策略

如需了解更多信息，请参阅业务连续性计划（BCP）。

2. 查看如何实施所选灾难恢复策略的模式。

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灾难恢复策略，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，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（例如另一个 AWS 区域）。

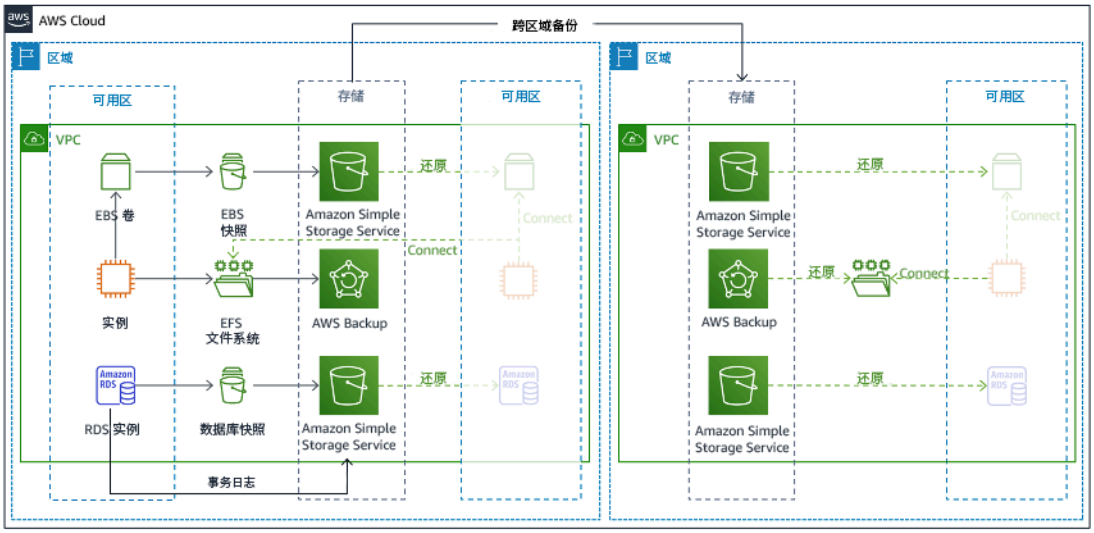


图 19：备份和还原架构

有关此策略的更多详细信息，请参阅《Disaster Recovery (DR) Architecture on AWS, Part II: Backup and Restore with Rapid Recovery》。

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，才能让此恢复区域成为功能堆栈。例如，图 20 中就没有部署计算实例。

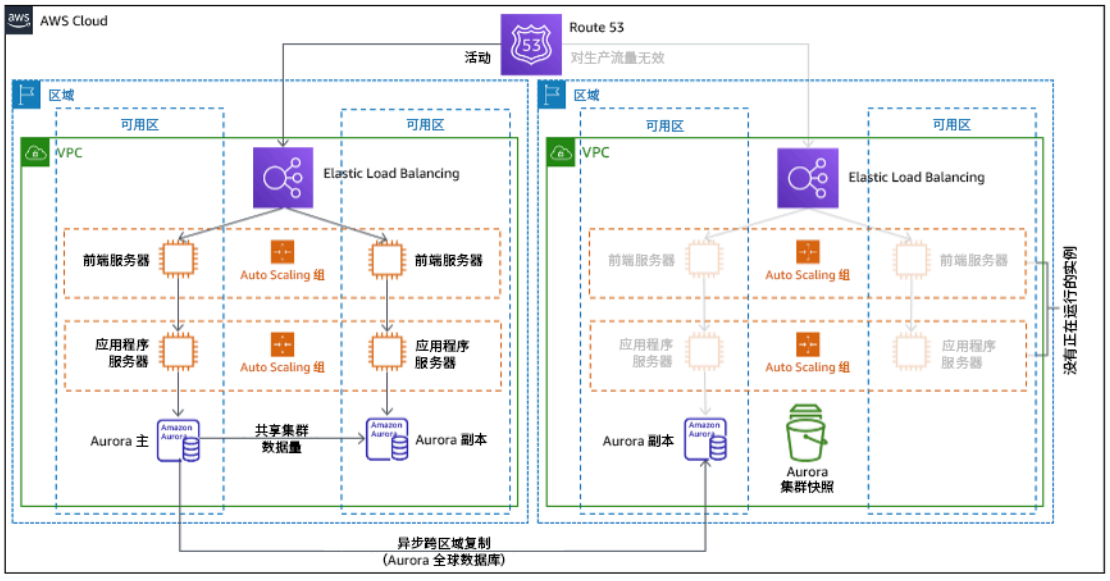


图 20：指示灯架构

有关此策略的更多详细信息，请参阅《Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby》。

温备用

温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，这种方式就称为热备用。

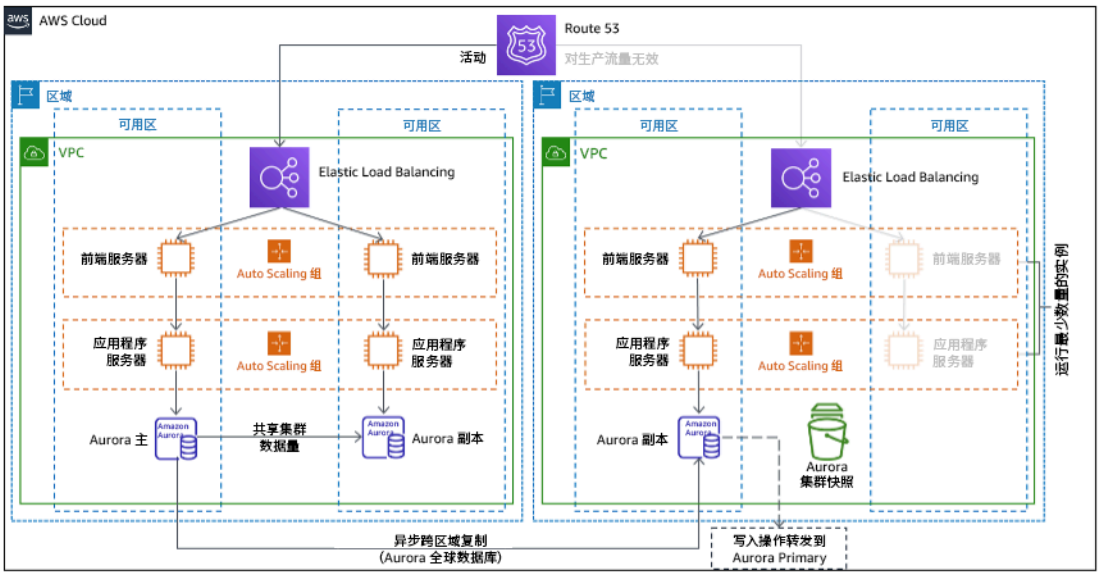


图 21：温备用架构

使用温备用或指示灯需要纵向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运行时环境，以便这些环境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《Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby》。

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。客户可能会出于灾难恢复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（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）时使用此策略。作为一种灾难恢复策略，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/主动策略是灾难恢复策略中操作最复杂的策略，只有在业务要求需要时才应选择它。

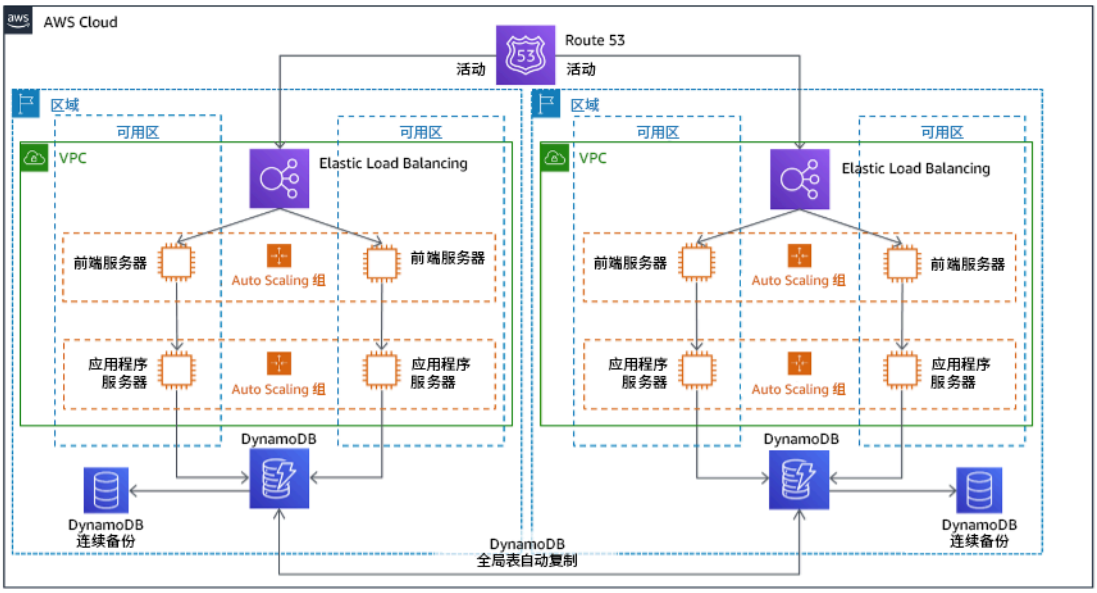


图 22：多站点主动/主动架构

有关此策略的更多详细信息，请参阅《Disaster Recovery (DR) Architecture on AWS, Part IV: Multi-site Active/Active》。

AWS Elastic Disaster Recovery

如果您正考虑为灾难恢复使用指示灯或温备用策略，AWS Elastic Disaster Recovery 可以提供一种带来更多好处的替代方法。弹性灾难恢复可以提供类似于温备用方法的 RPO 和 RTO 目标，同时保持指示灯方法的低成本。弹性灾难恢复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。使用弹性灾难恢复时，如果在失效转移或演练过程中启动，则服务会协调并编排计算资源的恢复。

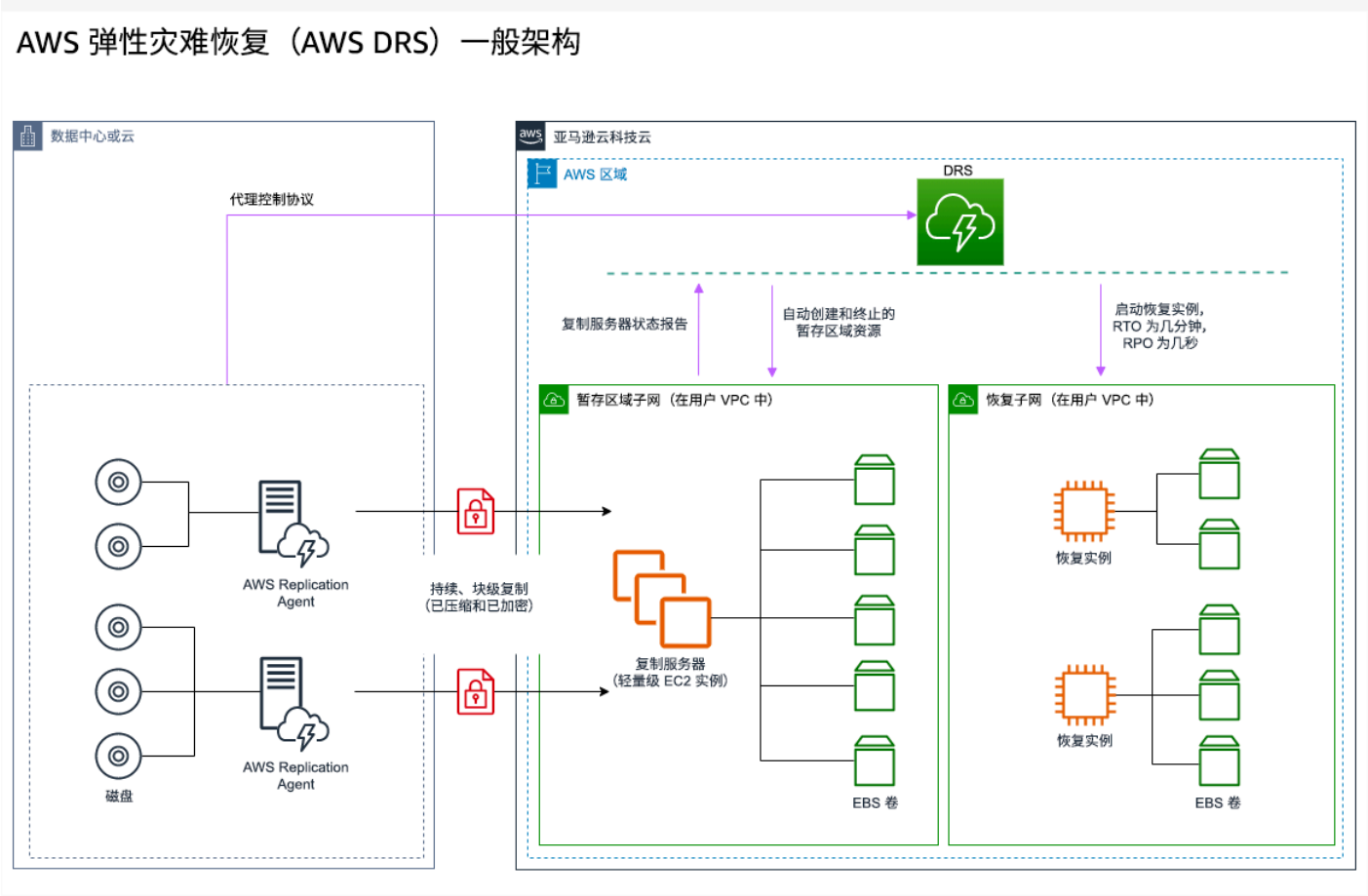


图 23：AWS Elastic Disaster Recovery 架构

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，实施灾难恢复会用到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 实例和弹性负载均衡器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。

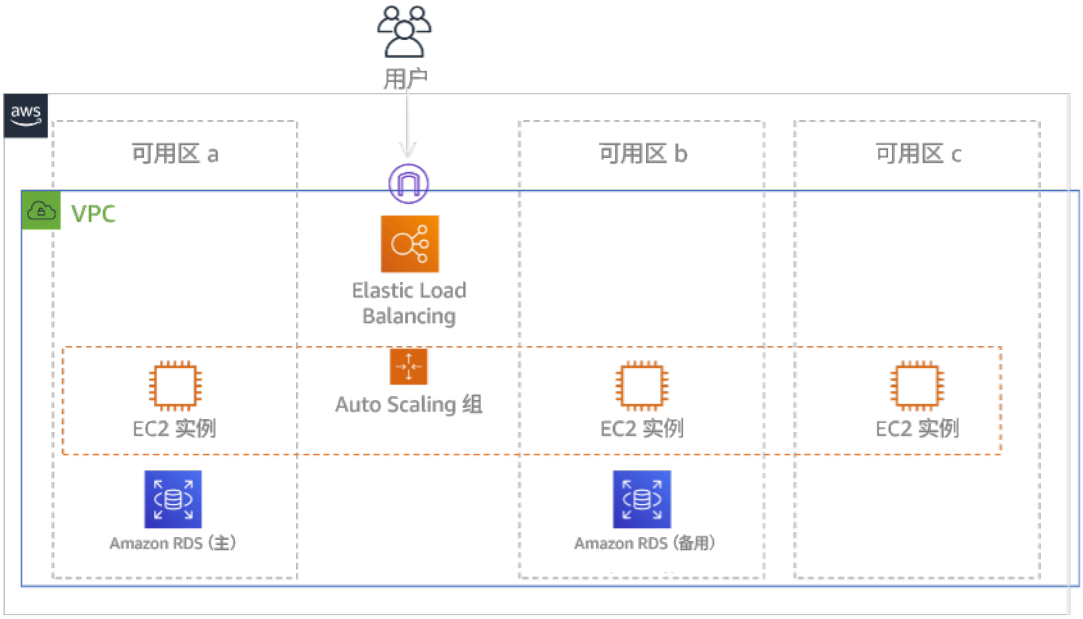


图 24：多可用区架构

除了这种高可用性架构，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灾难恢复的替代方法：Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack。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. 评测工作负载的资源，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。对于指示灯和温备用策略，部署的基础设施需要采取额外操作，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处于活动状态还是备用状态。使用弹性灾难恢复时，服务会复制并编排应用程序配置和计算资源的还原。

所有灾难恢复策略都要求在 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，可供您在其中配置、调度和监控这些资源的备份。对于指示灯、温备用和多站点主动/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，请参阅博客系列文章《Creating a Multi-Region Application with AWS Services》。

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。对于指示灯和温备用策略，恢复操作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。有关更多信息，请参阅 REL09-BP01 识别并备份需要备份的所有数据或从源复制数据。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，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 应用程序恢复控制器，利用其提供的高可用性数据面板 API 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，如此部分所述：REL11-BP04 恢复期间依赖于数据面板而不是控制面板。

   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中的此部分。

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，确保主区域处于最新状态。

   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，只要重新联机，DynamoDB 就会继续传播任何挂起的写操作。如果使用 Amazon Aurora Global Database 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，则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。

   失效转移后，如果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中且可供所有团队成员使用，并定期接受审查。

   使用弹性灾难恢复时，服务会协助编排并自动执行失效自动恢复流程。有关更多详细信息，请参阅 Performing a failback。

实施计划的工作量级别：高

资源

相关最佳实践：

- the section called “REL09-BP01 识别并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面板而不是控制面板”
- the section called “REL13-BP01 定义停机和数据丢失的恢复目标”

相关文档：

- AWS Architecture Blog: Disaster Recovery 系列博客文章
- 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
- Build a serverless multi-region, active-active backend solution in an hour
- Multi-region serverless backend — reloaded
- RDS：跨区域复制只读副本
- Route 53: Configuring DNS Failover
- S3：跨区域复制
- 什么是 AWS Backup？
- What is Amazon Application Recovery Controller?
- AWS 弹性灾难恢复
- HashiCorp Terraform: Get Started - AWS
- APN 合作伙伴：可帮助进行灾难恢复的合作伙伴
- AWS Marketplace：可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: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- Get Started with AWS Elastic Disaster Recovery | Amazon Web Services

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，验证是否在正常运作，以及是否满足 RTO 和 RPO。

常见反模式：

- 从不在生产环境中进行失效转移演练。

建立此最佳实践的好处：定期测试灾难恢复计划，验证计划在需要时能否正常发挥作用，以及团队是否知道如何执行策略。

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。在写入某个数据存储，却发现主存储故障时，您可能希望失效转移到辅助数据存储。如果不经常测试此失效转移，您可能会发现自己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。根据我们的经验，唯一有效的错误恢复路径是您经常测试的路径。因此，最好只制定几条恢复路径。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。对恢复路径进行演练，确认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 为灾难恢复策略实施和启动演练实例。AWS Elastic Disaster Recovery 可以高效地运行演练，帮助您为失效转移事件做好准备。您还可以使用弹性灾难恢复频繁地启动实例进行测试和演练，无需重定向流量。

资源

相关文档：

- APN 合作伙伴：可帮助进行灾难恢复的合作伙伴
- AWS Architecture Blog: Disaster Recovery 系列博客文章
- AWS Marketplace：可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
- The Berkeley/Stanford recovery-oriented computing project
- What is AWS Fault Injection Simulator?

相关视频：

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications

- AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS

REL13-BP04 管理灾难恢复站点或区域的配置偏差

为了成功执行灾难恢复（DR）过程，一旦灾难恢复环境上线，工作负载就必须能够及时恢复正常操作，而不会丢失相关的功能或数据。要实现这一目标，务必在灾难恢复环境和主环境之间保持一致的基础设施、数据和配置。

期望结果：灾难恢复站点的配置和数据与主站点相当，这有助于在需要时进行快速而完整的恢复。

常见反模式：

- 当对主位置进行更改时，您未能更新恢复位置，这导致配置过时，从而阻碍恢复工作。
- 您未考虑潜在的限制，例如主位置和恢复位置之间的服务差异，这些限制可能会在失效转移期间导致意外故障。
- 您依赖手动流程来更新和同步灾难恢复环境，这会增加人为错误和不一致的风险。
- 您未能检测到配置偏差，这会导致在事件发生之前错误地感知灾难恢复站点就绪状态。

建立此最佳实践的好处：灾难恢复环境和主环境之间的一致性可显著提高事件发生后成功恢复的可能性，并降低恢复过程失败的风险。

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

全面的配置管理和失效转移就绪方法有助于您验证灾难恢复站点是否持续更新，并准备好在主站点出现故障时进行接管。

要实现主环境和灾难恢复（DR）环境之间的一致性，请验证您的交付管道是否同时将应用程序分发到主站点和灾难恢复站点。在适当的评估期后推出对灾难恢复站点的更改（也称为错开部署），以检测主站点的问题，并在问题蔓延之前停止部署。实施监控以检测配置偏差，并跟踪环境中的更改和合规性。在灾难恢复站点中执行自动修复，以使其保持完全一致，并做好准备在发生事件时立即接管。

实施步骤

1. 验证灾难恢复区域包含成功执行灾难恢复计划所需的 AWS 服务和功能。
2. 使用基础设施即代码（IaC）。保持生产基础设施和应用程序配置模板的准确性，并定期将其应用于灾难恢复环境。AWS CloudFormation 可以检测 CloudFormation 模板指定的内容与实际部署内容之间的偏差。

3. 配置 CI/CD 管道来将应用程序和基础设施更新部署到所有环境，包括主站点和灾难恢复站点。诸如 AWS CodePipeline 等 CI/CD 解决方案可以自动执行部署过程，从而降低配置偏差的风险。
4. 在主环境和灾难恢复环境之间错开部署。这种方法支持在主环境中对更新进行初始部署和测试，这样可以在问题传播到灾难恢复站点之前，将其隔离在主站点中。这种方法可以防止同时将缺陷推送到生产和灾难恢复站点，并保持灾难恢复环境的完整性。
5. 持续监控主环境和灾难恢复环境中的资源配置。诸如 AWS Config 之类的解决方案有助于强制实施配置合规性并检测偏差，这有助于在不同环境中保持一致的配置。
6. 实施警报机制，以跟踪和通知任何配置偏差或数据复制中断或滞后。
7. 自动修复检测到的配置偏差。
8. 安排定期审计和合规性检查，以验证主配置和灾难恢复配置之间的持续一致性。定期审核可帮助您保持对既定规则的遵守，并确定需要解决的任何差异。
9. 检查 AWS 预置容量、服务配额、节流限制以及配置和版本差异是否存在不匹配。

资源

相关最佳实践：

- REL01-BP01 了解服务配额和约束
- REL01-BP02 跨多个账户和区域管理服务配额
- REL01-BP04 监控和管理配额
- REL13-BP03 测试灾难恢复实施以验证实施效果

相关文档：

- 按照 AWS Config 规则修正不合规 AWS 资源
- AWS Systems Manager Automation
- AWS CloudFormation：检测堆栈和资源的非托管配置更改
- AWS CloudFormation：在整个 CloudFormation 堆栈上检测偏差
- 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: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)

相关示例：

- CloudFormation 注册表
- 适用于 AWS 的配额监控程序
- Implement automatic drift remediation for AWS CloudFormation using Amazon CloudWatch and AWS Lambda
- AWS Architecture Blog: Disaster Recovery 系列博客文章
- AWS Marketplace：可用于灾难恢复的产品
- 自动实现无需干预的安全部署

REL13-BP05 自动执行恢复

实施可靠、可观测、可重复且经过测试的自动化恢复机制，来降低故障的风险和业务影响。

期望结果：您已经为恢复流程实施了有据可查、标准化且经过全面测试的自动化工作流程。恢复自动化功能会自动纠正导致数据丢失或不可用（低风险）的小问题。您可以针对严重事件快速调用恢复流程，在恢复流程运行时观察补救行为，并在观测到危险情况或故障时结束这些流程。

常见反模式：

- 您依赖于处于故障或已降级状态的组件或机制，作为恢复计划的一部分。
- 恢复流程需要手动干预，例如控制台访问（也称为单击操作）。
- 您在存在数据丢失或不可用高风险的情况下自动启动恢复过程。
- 您没有包含一个机制来中止不起作用或带来额外风险的恢复过程（如暗灯 或红色的大型停止按钮）。

建立此最佳实践的好处：

- 提高了恢复操作的可靠性、可预测性和一致性。
- 能够实现要求更高的恢复目标，包括恢复时间目标（RTO）和恢复点目标（RPO）。

- 降低了事件期间恢复失败的可能性。
- 降低了与容易出现人为错误的手动恢复过程相关的故障风险。

在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

要实施自动恢复，您需要一种使用 AWS 服务和最佳实践的全面方法。首先，请确定工作负载中的关键组件和潜在故障点。开发自动化流程，无需人工干预即可从故障中恢复工作负载和数据。

使用基础设施即代码（IaC）原则开发恢复自动化功能。这使您的恢复环境与源环境保持一致，并支持对恢复过程进行版本控制。要编排复杂的恢复工作流程，可以考虑诸如 AWS Systems Manager 自动化或 AWS Step Functions 等解决方案。

恢复过程自动化可带来显著的好处，有助于您更轻松地实现恢复时间目标（RTO）和恢复点目标（RPO）。但是，此类功能可能会遇到意外情况，而可能会导致失败或造成新的风险，例如额外的停机时间和数据丢失。要降低这种风险，请提供快速停止正在进行的恢复自动化的功能。一旦停止，您就可以进行调查并采取纠正措施。

对于支持的工作负载，可以考虑使用 AWS 弹性灾难恢复（AWS DRS）等解决方案来提供自动失效转移。AWSDRS 会持续将计算机（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暂存区。如果发生事件，AWS DRS 会自动将复制的服务器转换为 AWS 上恢复区域中完全预置的工作负载。

维护和改进自动恢复是一个持续的过程。根据经验教训不断测试和完善恢复过程，并随时了解可以增强恢复能力的新 AWS 服务和功能。

实施步骤

1. 规划自动恢复
    a. 对您的工作负载架构、组件和依赖关系进行全面审核，来确定和规划自动恢复机制。将工作负载的依赖关系分类为硬 依赖关系和软 依赖关系。硬依赖关系是指工作负载运行所必须具备且无法替代的依赖关系。软依赖关系是工作负载通常使用的依赖关系，但此类依赖关系可以用临时替代系统或流程替换，或者可以通过优雅降级进行处理。
    b. 建立识别和恢复丢失或已损坏数据的流程。
    c. 定义在完成恢复操作后确认已恢复的稳定状态的步骤。
    d. 考虑为使恢复的系统准备好提供全面服务所需的任何操作，例如预热和填充缓存。
    e. 考虑在恢复过程中可能遇到的问题以及如何检测和修复这些问题。

   f. 考虑无法访问主站点及其控制面板的场景。验证可以在不依赖主站点的情况下独立执行恢复操作。考虑使用诸如 Amazon 应用程序恢复控制器（ARC）之类的解决方案来重定向流量，而无需手动更改 DNS 记录。

2. 开发自动恢复流程
   a. 实施自动化的故障检测和失效转移机制，来实现免手动恢复。构建控制面板（例如使用 Amazon CloudWatch）来报告自动恢复过程的进度和运行状况。包括验证成功恢复的过程。提供一种机制来中止正在进行的恢复。
   b. 对于无法自动恢复的故障，编写行动手册作为后备流程，并考虑制定灾难恢复计划。
   c. 测试恢复过程，如 REL13-BP03 中所述。
3. 为恢复做好准备
   a. 评估恢复站点的状态并提前为其部署关键组件。有关更多详细信息，请参阅 REL13-BP04。
   b. 为恢复操作定义明确的角色、职责和决策过程，涉及整个组织的有关利益相关者和团队。
   c. 定义启动恢复过程的条件。
   d. 制定计划来还原恢复过程，需要时或在认为安全之后回退到主站点。

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
- 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 管理灾难恢复站点或区域的配置偏差

相关文档：

- AWS Architecture Blog: Disaster Recovery 系列博客文章
- AWS 上工作负载的灾难恢复：云中的恢复（AWS 白皮书）
- Orchestrate Disaster Recovery Automation using Amazon Route 53 ARC and AWS Step Functions
- Build AWS Systems Manager Automation runbooks using AWS CDK
- AWS Marketplace: Products That Can Be Used for Disaster Recovery
- AWS Systems Manager Automation

- AWS 弹性灾难恢复
- 使用弹性灾难恢复进行失效转移和失效自动恢复
- AWS 弹性灾难恢复资源
- APN 合作伙伴：有助于进行灾难恢复的合作伙伴

相关视频：

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- AWS re:Invent 2022: AWS On Air ft. AWS Failback for AWS Elastic Disaster Recovery

结论

无论刚开始接触可用性和可靠性主题的新手，还是希望寻求见解来尽力提高关键任务型工作负载可用性的经验丰富的老手，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。我们希望这能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面向设计、运营和恢复的建议，以及 AWS 解决方案架构师的知识和经验。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：

- Michael Fischer，Amazon Web Services 首席解决方案架构师
- Seth Eliot，Amazon Web Services 首席开发技术推广工程师
- Mahanth Jayadeva，Amazon Web Services Well-Architected 业务首席解决方案架构师
- James Devine，Amazon Web Services 首席解决方案架构师
- Jason DiDomenico，Amazon Web Services Cloud Foundations 业务高级解决方案架构师
- Marcin Bednarz，Amazon Web Services 首席解决方案架构师
- Tyler Applebaum，Amazon Web Services 高级解决方案架构师
- Rodney Lester，亚马逊云科技首席解决方案架构师
- Joe Chapman，Amazon Web Services 高级解决方案架构师
- Adrian Hornsby，Amazon Web Services 首席系统开发工程师
- Kevin Miller，Amazon Web Services S3 业务副总裁
- Shannon Richards，Amazon Web Services 技术项目经理
- Laurent Domb，Amazon Web Services Fed Fin 业务首席技术专家
- Kevin Schwarz，Amazon Web Services 高级解决方案架构师
- Rob Martell，Amazon Web Services 首席云韧性架构师
- Priyam Reddy，Amazon Web Services 灾难恢复业务高级解决方案架构师
- Jeff Ferris，Amazon Web Services 首席技术专家
- Matias Battaglia，Amazon Web Services 高级解决方案架构师

延伸阅读

有关更多信息，请参阅：

- AWS Well-Architected Framework
- AWS 架构中心

文档修订

如需获取有关该白皮书更新的通知，请订阅 RSS 信息源。

变更	说明	日期
更新了最佳实践指南	根据以下领域的新指导更新了最佳实践：REL 1、REL 2、REL 4、REL 6、REL 7、REL 8、REL 10、REL 12 和 REL 13 整个支柱的指导都得到了扩展和阐述。REL10-BP02 和 REL12-BP03 已将其指导合并到其它最佳实践中。整个支柱的资源已更新。	2024 年 11 月 6 日
更新了最佳实践指南	对 REL 2、4、5、6、7 和 8 中的最佳实践做了小幅更新。	2024 年 6 月 27 日
更新了最佳实践指南	根据以下领域的新指导更新了最佳实践：在分布式系统中设计交互来预防发生故障、在分布式系统中设计交互来减少或承受故障、监控工作负载资源、设计工作负载来适应需求变化、实施变更和测试可靠性。	2023 年 12 月 6 日
更新了最佳实践指南	根据以下领域的新指导更新了最佳实践监控工作负载资源和设计工作负载来承受组件故障。	2023 年 10 月 3 日
更新了最佳实践指南	根据以下领域的新指导更新了最佳实践：设计工作负载服务架构、在分布式系统中设计交	2023 年 7 月 13 日

	互以减少或承受故障和监控工作负载资源。	
次要更新	删除非包容性用语。	2023 年 4 月 13 日
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。	2023 年 4 月 10 日
已更新白皮书	为最佳实践更新了新的实施指导。	2022 年 12 月 15 日
次要更新	全文更正了图号并进行了细微改动。	2022 年 11 月 17 日
已更新白皮书	扩展了最佳实践并增加了改进计划。	2022 年 10 月 20 日
已更新白皮书	在使用故障隔离来保护工作负载和设计工作负载来承受组件故障两节中，为可靠性支柱添加了两个新的最佳实践。	2022 年 5 月 5 日
已更新白皮书	更新灾难恢复指导，纳入 Route 53 应用程序恢复控制器的相关信息。添加对 DevOps Guru 的引用。更新了多个资源链接，并做了其他编辑方面的细微改动。	2021 年 10 月 26 日
次要更新	添加了有关 AWS Fault Injection Service（AWS FIS）的信息。	2021 年 3 月 15 日
次要更新	对文本进行了细微更新。	2021 年 1 月 4 日

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；更清晰地解释可用性和灾难恢复之间的差异，以及二者如何促进实现韧性；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；改进对演练日活动的描述	2020 年 12 月 7 日
次要更新	更新了附录 A 以更新 AWS Lambda 的可用性设计目标	2020 年 10 月 27 日
次要更新	更新了附录 A 以添加 AWS Global Accelerator 的可用性设计目标	2020 年 7 月 24 日

针对新框架进行了更新	大量更新和全新/修订内容，包括：新增了“工作负载架构”最佳实践小节；将最佳实践整合到“变更管理”和“故障管理”小节；更新了“资源”，将最新的 AWS 资源和服务纳入其中，如 AWS Global Accelerator、AWS Service Quotas 和 AWS Transit Gateway；新增了/更新了可靠性、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）对白皮书进行了调整；对设计原则进行了重新排列；将自动从故障中恢复移到了测试恢复程序前面；更新了公式的图表和格式；删除了“关键服务”小节，转而将对关键的 AWS 服务的引用整合到最佳实践中。	2020 年 7 月 8 日
次要更新	修复了失效链接	2019 年 10 月 1 日
已更新白皮书	更新了附录 A	2019 年 4 月 1 日
已更新白皮书	新增了具体的 AWS Direct Connect 联网建议和额外的服务设计目标	2018 年 9 月 1 日
已更新白皮书	新增了“设计原则”和“限制管理”小节。更新了链接，删除了上游/下游术语的歧义，并为可用性场景中的其余“可靠性支柱”主题新增了详细的参考。	2018 年 6 月 1 日

已更新白皮书	更改了针对 DynamoDB 全局表的 DynamoDB 跨区域解决方案。新增了服务设计目标	2018 年 3 月 1 日
次要更新	对可用性计算进行了细微更正，将应用程序可用性纳入其中	2017 年 12 月 1 日
已更新白皮书	内容经过更新，提供了关于高可用性设计的指导，包括概念、最佳实践和实施示例指导。	2017 年 11 月 1 日
初次发布	发布了可靠性支柱 – AWS Well-Architected Framework。	2016 年 11 月 1 日

版权声明

AWS 术语表

有关最新的 AWS 术语，请参阅 AWS 词汇表 参考中的 AWS 词汇表。