可靠性支柱

AWS 架构完善的框架

可靠性支柱: AWS 架构完善的框架

Table of Contents

可靠性支柱 – AWS 架构完善的框架

发布日期： 2020 年 7 月 (文档修订 (p. 49))

摘要

本白皮书主要介绍以下架构的可靠性支柱： AWS 架构完善的框架。文中提供了指导，可帮助客户在 Amazon Web Services (AWS) 环境的设计、交付和维护过程中应用最佳实践。

引言

如示例所示，AWS 架构完善的框架 能够帮助您认识到您在 AWS 上构建工作负载时所做决策的优缺点。通过使用此框架，您将了解在云中设计和运行可靠、安全、高效且经济实惠的工作负载的架构的最佳实践。它提供了一种统一的方法，使您能够根据最佳实践衡量架构，并确定需要改进的方面。我们相信，拥有架构完善的工作负载能够大大提高实现业务成功的可能性。

AWS 良好架构框架基于五个支柱：

- 卓越运营
- 安全性
- 可靠性
- 性能效率
- 成本优化

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

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

可靠性

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

主题

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绩效指标 (KPI)，您可以在指标超过阈值时触发自动化功能。这些 KPI 应该是对业务价值（而不是服务运营的技术方面）的一种度量。这包括自动发送故障通知和跟踪故障，以及启动解决或修复故障的自动恢复流程。借助更高级的自动化功能，您可以在故障发生之前预测和修复故障。
- 测试恢复过程： 在本地环境中，经常会通过执行测试来证明工作负载能够在特定场景中正常运作。通常不会利用测试来验证恢复策略。在云中，您可以测试工作负载的故障情况，并验证您的恢复程序。您可以采用自动化方式来模拟不同的故障，也可以重新建立之前导致故障的场景。此方式可以在实际的故障发生 移动到 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： 使用多个小型资源替换一个大型资源，以降低单个故障对整个工作负载的影响。跨多个较小的资源分配请求，以确保它们不共用常见故障点。
- 无需再预估容量： 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 (p. 7)”）。
- 管理自动化变更：应利用自动化功能对基础设施进行更改。需要管理的变更包括，对自动化的变更，可对其进行跟踪与审查。

定义

本白皮书涵盖了云中的可靠性，对以下四个领域的最佳实践进行描述：

- 基础
- 工作负载架构
- 变更管理
- 故障管理

要实现可靠性，您必须从基础入手，而基础是服务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

- 恢复时间目标 (RTO) 和恢复点目标 (RPO) (p. 5)

弹性，以及可靠性的组件

云中的工作负载的可靠性取决于多个因素，其中最主要的要属 弹性：

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

会对工作负载可靠性产生影响的其他因素还有：

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

弹性是本白皮书的主要关注点。

其他四个因素也很重要，我们将在讨论以下架构的对应支柱时加以介绍： AWS 架构完善的框架。在最佳实践部分可能会提到它们，但本白皮书的重点还是放在弹性。

可用性

可用性 （也称为 服务可用性）是对弹性进行定量测量的常用指标。

- 可用性 是工作负载可供使用的时间百分比。

可供使用 指的是它会在有需要时履行其约定的功能。

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

- 一段时间内（通常为一年）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
- 在第一项中的公式中，有些客户选择在总时间中排除计划的维护停机时间（例如，计划的维护）。但是，这通常是错误的选择，因为在此期间，您的用户实际上可能需要使用您的服务。

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

可用性	最大不可用性（每年）	应用程序类别
99% (p. 35)	3 天 15 小时	批处理、数据提取、传输和负载作业
99.9% (p. 37)	8 小时 45 分钟	内部工具，如知识管理、项目跟踪

可用性	最大不可用性（每年）	应用程序类别
99.95% (p. 40)	4 小时 22 分钟	网上商务、销售点
99.99% (p. 38)	52 分钟	视频传输、广播 工作负载
99.999% (p. 42)	5 分钟	ATM 交易、电信 工作负载

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

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

99.99% × 99.99% × 99.99% = 99.97%

因此，在计算您自己的可用性时，一定要了解您的依赖项及其可用性设计目标。

针对冗余组件计算可用性。 当系统涉及到使用独立的冗余组件（例如，冗余的可用区）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，得到的系统可用性为 99.9999%：

$$可用_{工作负载} = 可用_{最大值} - ((100\% - 可用_{依赖项}) \times (100\% - 可用_{依赖项}))$$

100% − (0.1%×0.1%) = 99.9999%

但是，如果我不知道依赖项的可用性，该怎么办？

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（请参阅“ 附录 A：旨在展示所选 AWS 服务的可用性 (p. 50)” ）。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) and 平均修复时间 (MTTR)。可以通过以下公式来确定可用性估算值：

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

有关其他详细信息，请参阅 此文档（计算总体系统可用性），它可帮助您计算您的可用性。

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

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

恢复时间目标 (RTO) 和恢复点目标 (RPO)

此类术语最常与灾难恢复 (DR) 相关联，指的是在灾难中恢复工作负载的一系列目标与策略。

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

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

RPO（恢复点目标）、RTO（恢复时间目标）和灾难事件之间的关系。

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

了解可用性需求

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

推荐
在适当时对您的应用程序的独特方面进行批判性评估，区分可用性设计目标以反映您的业务需求。

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

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 拥有丰富的应用程序工程经验以及一系列可用性设计目标，包括可实现 99.999% 或更高可用性的服务。AWS 解决方案架构师 (SA) 可帮助您根据可用性目标进行合理设计。在设计过程中尽早让 AWS 参与有助于我们更好地帮助您实现可用性目标。并不是只有在启动工作负载前才要针对可用性进行规划。还应该持续不断地进行规划，从而在获得运营经验的过程中细化设计，从实际事件中吸取经验教训，并能承受不同类型的故障。然后，您可以投入适当的工作来改进实施。

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

基础

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

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

下文将针对此类可靠性注意事项的最佳实践进行说明。

主题

管理 Service Quotas 和约束

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

如果您使用的是 AWS Marketplace 应用程序，则必须了解应用程序的限制。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。

了解服务配额和限制：您要知道您的工作负载架构的默认配额和配额提高请求。您还要了解哪些资源限制（如磁盘或网络）可能会对您产生影响。

Service Quotas 是一项 AWS 服务，可帮助您在一个地方对超过 100 项 AWS 服务的配额进行管理。除了查找配额值，您还可以在 Service Quotas 控制台或通过 AWS 开发工具包请求和跟踪配额的提高。AWS Trusted Advisor 提供服务配额检查，显示您的服务使用情况，以及某些服务在某些方面的配额。不同服务的默认服务配额也见于对应服务的 AWS 文档，例如，请参阅 Amazon VPC 配额。通过配置使用计划，可在 API Gateway 内设置限流 API 的速率限制。可通过配置对应的服务进行设置的其他限制包括预置 IOPS、已分配的 RDS 存储，以及 EBS 卷分配等。Amazon Elastic Compute Cloud (Amazon EC2) 有自己的服务限制控制面板，可帮助您管理您的实例、Amazon Elastic Block Store (Amazon EBS) 和弹性 IP 地址限制。如果在某用例中，服务配额会对您的应用程序的性能造成影响，而且您无法根据自身需求对其进行调整，请联系 AWS Support 了解是否有解决的办法。

跨多个账户和区域管理配额：如果您目前使用多个 AWS 账户或 AWS 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。

每个账户的服务配额都可被跟踪。除非另外说明，否则每个配额都针对特定的 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，以避免妨碍测试与开发。

通过架构适应固定服务配额和限制：请注意不可更改的服务配额和物理资源，并且在设计架构时要防止这些因素影响可靠性。

其中的示例包括网络带宽、AWS Lambda 有效负载大小、限制 API Gateway 的突发速率，以及并发用户连接至 Amazon Redshift 集群。

监控和管理配额：评估您的可能使用情况，并适当提高您的配额，支持使用量按计划增长。

对于支持的服务，您可以通过配置 CloudWatch 警报来监控使用情况，并在接近配额时发出提醒，从而管理您的配额。这些警报可以从 Service Quotas 或 Trusted Advisor 触发。您还可以使用 CloudWatch Logs 上的指标筛选条件来搜索与提取日志中的模式，确定使用量是否快达到配额阈值。

自动管理配额：实施工具以便在接近阈值时向您发送提醒。借助 Service Quotas API，您可以对提高配额请求进行自动化。

如果将您的配置管理数据库 (CMDB) 或票证系统与 Service Quotas 集成，您可以自动跟踪配额提高请求和当前配额。除了 AWS 开发工具包，Service Quotas 还支持通过 AWS 命令行工具进行自动化。

确保在当前配额与最大使用量之间存在足够的差距，以便应对故障转移：当资源出现故障时，它可能仍会被计入配额，直到被成功终止。在出现故障的资源被终止之前，请确保您的配额涵盖所有出现故障的资源与其替换资源的叠加。在计算此差距时，应将可用区故障考虑在内。

资源

视频

- AWS Live re:Inforce 2019 - Service Quotas

文档

- 什么是 Service Quotas？
- AWS Service Quotas （以前称为服务限制）
- Amazon EC2 服务限制
- AWS Trusted Advisor 最佳实践检查 （请参阅“服务限制”部分）
- AWS Answers 上的 AWS Limit Monitor
- AWS Marketplace：可帮助跟踪限制的 CMDB 产品
- APN 合作伙伴：可帮助进行配置管理的合作伙伴

计划网络拓扑

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

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

Amazon Virtual Private Cloud (Amazon VPC) 让您在 AWS 云中部署一个私有隔离的部分，并在这个虚拟网络中启动 AWS 资源

为工作负载公有终端节点使用高度可用的网络连接：这些终端节点及其路由必须高度可用。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。

Amazon Route 53、AWS Global Accelerator、Amazon CloudFront、Amazon API Gateway 和 Elastic Load Balancing (ELB) 都可以提供高度可用的公有终端节点。您还可以选择评估 AWS Marketplace 软件设备是否适用于负载均衡和代理。

您的工作负载所提供的服务的用户，无论其是最终用户或其他服务，都要在这些服务终端节点上发起请求。您可以使用多个 AWS 资源以提供高度可用的终端节点。

Elastic Load Balancing 在多个可用区提供负载均衡，执行第 4 层 (TCP) 或第 7 层 (http/https) 路由，与 AWS WAF 以及 AWS Auto Scaling 集成，从而帮助构建自我修复基础设施，吸收流量增长，并同时在流量减少时释放资源。

Amazon Route 53 是一项可扩展且高度可用的域名系统 (DNS) 服务，可将用户请求连接至在 AWS 中运行的基础设施，如 Amazon EC2 实例、Elastic Load Balancing 负载均衡器或 Amazon S3 存储桶，此外还可用于将用户路由至 AWS 以外的基础设施。

AWS Global Accelerator 是一项网络层服务，您可以用它将流量引导至 AWS 全球网络中的最佳终端节点。

分布式拒绝服务 (DDoS) 攻击会引发使您的用户的合法流量无法进入并降低可用性的风险。AWS Shield 提供自动防护，无需额外成本即可避免您的工作负载上的 AWS 服务终端节点遭受此类攻击。您可以使用 APN 合作伙伴和 AWS Marketplace 提供的虚拟设备来增强这些功能，以满足您的需求。

为云环境和本地环境之间的私有网络预置冗余连接： 在单独部署的私有网络之间使用多个 AWS Direct Connect (DX) 连接或 VPN 隧道。使用多个 DX 位置以实现高可用性。如果使用多个 AWS 区域，请确保其中至少有两个区域存在冗余。您可能想要评估终止 VPN 的 AWS Marketplace 设备。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。

AWS Direct Connect 是一项云服务，可以在本地环境和 AWS 之间轻松建立专用网络连接。使用 Direct Connect Gateway，您的本地数据中心可以连接至跨多个 AWS 区域的多个 AWS VPC。

此类冗余可解决会对连接弹性造成影响的潜在故障：

- 您将如何灵活应对拓扑中的故障？
- 如果您配置错了某些内容并删除了连接，会发生什么情况？
- 您是否能应对服务流量/使用量的意外增加？
- 您是否能够吸收未遂的分布式拒绝服务 (DDoS) 攻击？

若通过 VPN 将您的 VPC 连接至本地数据中心，您应该考虑在选择运行该设备所需的供应商和实例大小时所需要的弹性和带宽要求。如果您使用的 VPN 设备在其实施中没有弹性，则您应该通过第二个设备获取冗余连接。对于所有这些场景，您需要定义可接受的恢复时间并进行测试，以确保您可以满足这些要求。

如果选择通过 Direct Connect 连接将您的 VPC 连接至您的数据中心，而且您需要它们是高度可用的连接，从每个数据中心获得冗余 DX 连接。冗余连接应使用与第一个来自不同位置的其他 DX 连接。如果您有多个数据中心，则确保连接在不同的位置终止。使用 Direct Connect 弹性工具包 以帮助您完成设置。

如果您选择使用 AWS VPN 并通过互联网将故障转移到 VPN，一定要了解，它支持每个 VPN 隧道高达 1.25-Gbps 的吞吐量，但在多个 AWS 托管 VPN 隧道终止于同一 VGW 的情况下，不支持将等价多路径 (ECMP) 用于出站流量。我们不建议您使用 AWS 托管 VPN 作为 Direct Connect 连接的备份，除非您可以接受故障转移期间的速度低于 1 Gbps。

您还可以使用 VPC 终端节点将您的 VPC 私下连接至受支持的 AWS 服务和 VPC 终端节点服务，它们得到 AWS PrivateLink 的支持而无需遍历公有互联网。终端节点为虚拟设备。它们是水平扩展、冗余，而且高度可用的 VPC 组件。它们支持在您的 VPC 和服务实例之间进行通信，而不会对您的网络流量施加可用性风险或带宽限制。

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

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

- 在每个区域中为多个 VPC 留出 IP 地址空间。
- 在 VPC 内，为跨多个可用区的多个子网留出空间。
- 始终在 VPC 内保留未使用的 CIDR 块空间以用于未来扩展。
- 确保有 IP 地址空间以满足您可能使用的任何 EC2 实例临时性队列的需求，如适用于机器学习的 Spot 队列、Amazon EMR 集群或 Amazon Redshift 集群。
- 注意，每个子网 CIDR 块中的前四个 IP 地址和最后一个 IP 地址将被预留而无法供您使用。

  您应计划部署大型 VPC CIDR 块。注意，最初被分配到您的 VPC 的 VPC CIDR 块无法被更改或删除，但您可以向 VPC 添加额外的非重叠的 CIDR 块。虽然无法更改 IPv4 CIDR，但可以对 IPv6 CIDR 进行更

改。请记住，部署最大的 VPC (/16) 会产生超过 65000 个 IP 地址。单单在基础 10.x.x.x IP 地址空间内，您可以预置 255 个这样的 VPC。因此，您可以趋向于过大数量而不是过小数量，这样更容易管理 VPC。

轴辐式拓扑优先于多对多网格： 如果通过 VPC 对等连接、AWS Direct Connect 或 VPN 连接的网络地址空间超过两个（例如，VPC 和本地网络），则使用与 AWS Transit Gateway 所提供的模型类似的轴辐式模型。

如果只有两个此类网络，您可以简单地使其相互连接，但随着网络数量的增加，这种网格连接的复杂性将变得无法接受。AWS Transit Gateway 提供易于维持的轴辐式模型，允许在您的多个网络中对流量进行路由。

图 1：若无 AWS Transit Gateway：您需要通过 VPN 连接将每个 Amazon VPC 对等连接至其他 VPC 和每个现场位置，其复杂程度会随着它的扩展而递增。

图 2：若有 AWS Transit Gateway：您只需将每个 Amazon VPC 或 VPN 连接至 AWS Transit Gateway，而且它会在每个 VPC 或 VPN 之间路由流量。

在互相连接的所有私有地址空间中必须采用非重叠的私有 IP 地址范围： 多个 VPC 通过对等连接或 VPN 连接时，各个 VPC 的 IP 地址范围不得重叠。与之类似，您必须避免 VPC 和本地环境或与其他您使用的云提供商之间出现 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) 系统可以帮助解决这个问题。AWS Marketplace 提供多个 IPAM。

资源

视频

- AWS re:Invent 2018：Amazon VPC 的高级 VPC 设计和新功能 (NET303)
- AWS re:Invent 2019：适用于众多 VPC 的 AWS Transit Gateway 参考架构 (NET406-R1)

文档

- 什么是中转网关？
- 什么是 Amazon VPC？
- 使用 Direct Connect 网关
- 使用 Direct Connect 弹性工具包开始操作
- 多数据中心 HA 网络连接
- 什么是 AWS Global Accelerator？
- 使用冗余站点到站点 VPN 连接以提供故障转移
- VPC 终端节点和 VPC 终端节点服务 (AWS PrivateLink)
- Amazon Virtual Private Cloud 连接选项白皮书
- 适用于网络基础设施的 AWS Marketplace
- APN 合作伙伴：可帮助您规划联网的合作伙伴

工作负载架构

可靠的工作负载始于前期的软件和基础设施设计决策。您的架构选择将影响所有五个架构完善支柱的工作负载行为。针对可靠性，您必须遵循特定的模式。

以下各节将介绍使用这些保证可靠性的模式时要遵循的最佳实践。

主题

设计您的工作负载服务架构

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

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

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。

选择如何划分工作负载：应避免使用整体式架构。您应该在 SOA 和微服务之中做选择。在做选择时，权衡优点和复杂性 – 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。使用更小型的区段的好处包括，更高的敏捷性、组织灵活性和可扩展性。而复杂性则包括可能会增加延迟，调试变得更复杂，而且加重运营负担。

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。一个影响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。AWS X-Ray 可用于帮助您解决此问题。需要考虑的另一个影响是，您管理的应用程序数量激增时，运营复杂性也会随之增加，这要求部署多个相互独立组件。

图 3：整体架构和微服务架构

构建专注于特定业务领域和功能的服务：SOA 采用由业务需求定义的划分明确的功能来构建服务。微服务则使用领域模型和有界上下文对此进行进一步限制，使每项服务都 只用于一种用途。专注于特定功能让您可以区分不同服务的可靠性要求，并且更有针对性地锁定投资目标。简化的业务问题和与每项服务关联的小型团队同时简化了组织扩展。

在设计微服务架构时，借助于领域驱动设计 (DDD) 对使用实体的业务问题进行建模十分有帮助。以 Amazon.com 实体为例，它可能包括包装、配送、预定、价格、折扣和货币。然后，该模型会使用 边界上下文进一步细分为更小的模型，具有相似功能和属性的实体在边界上下文中被分到一组。因此，在 Amazon 例子中，包装、配送和预定是装运上下文的一部分，而价格、折扣和货币是定价上下文的一部分。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。

根据 API 提供服务合同：服务合同是团队之间关于服务集成的成文协议，它包括机器可读的 API 定义、速率限制和性能预期。版本控制策略让客户能够继续使用现有的 API，并在更新的 API 准备就绪时将他们的应用程序迁移到此类 API。只要遵守合同，即可随时进行部署。服务提供商团队可以使用自己选择的技术堆栈来满足 API 合同要求。同样，服务使用者可以使用自己的技术。

微服务将 SOA 的概念提升到创建具有最小功能集的服务。每项服务都会发布一个 API，以及使用相应服务的设计目标、限制和其他注意事项。这会通过调用应用程序建立“合同”。这可以实现三个主要优势：

- 服务具有一个需要解决的简明的业务问题，以及出现该业务问题的小型团队。这有助于更好地扩展组织。
- 只要满足 API 和其他“合同”要求，团队就可以随时进行部署。
- 只要满足 API 和其他“合同”要求，团队就可以使用他们想用的任何技术堆栈。

Amazon API Gateway 是一种完全托管的服务，可以帮助开发人员轻松创建、发布、维护、监控和保护任意规模的 API。它负责处理多达数十万个并发 API 调用的接受和处理过程中涉及的所有任务，包括流量管理、授权和访问控制、监控以及 API 版本管理。采用 OpenAPI 规范 (OAS)，亦即之前的 Swagger 规范，您可以定义 API 合同并将其导入到 API Gateway。然后，您便可以通过 API Gateway 对 API 进行版本控制与部署。

资源

文档

- Amazon API Gateway： 使用 OpenAPI 配置 REST API
- 在 AWS 上实施微服务
- AWS 上的微服务

外部链接

- 微服务 – 一个全新架构术语的定义
- 微服务利弊权衡
- 边界上下文 （领域驱动设计的中心模式）

在分布式系统中设计交互以预防发生故障

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

确定需要哪种类型的分布式系统：硬性实时分布式系统需要同步并快速地做出响应，而软性实时系统有更宽松的以分钟计的时间窗口，或更多响应。离线系统会对响应进行批处理或异步处理。硬性实时分布式系统具有最严格的可靠性要求。

硬性实时分布式系统 要面对分布式系统中的 最艰巨的挑战，又被称作请求/回复服务。因为收到请求的时间不可预测，而响应必须迅速（例如，客户正急切地等待响应）。此类示例包括，前端 Web 服务器、指令管道、信用卡交易，以及每个 AWS API 和语音通话。

实施松散耦合的依赖关系：队列系统、流系统、工作流和负载均衡器等依赖关系是松散耦合的。松散耦合有助于隔离来自其他依赖于它的组件的行为，从而提升弹性和敏捷性。

如果对一个组件的更改会强迫其他依赖于它的组件也发生更改，则它们之间的关系为 紧密 耦合。松散 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散耦合将隔离一个组件中的故障，防止对其他组件造成影响。

松散耦合让您可以为组件增加额外的代码或功能，同时在最大程度上降低依赖于它的组件的风险。而且，随着您可以进行扩展，或甚至更改依赖项的底层实施，可扩展性也得到改善。

要通过松散耦合进一步提升弹性，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，SQS 队列或诸如 Amazon Kinesis 或 AWS Step Functions 的流数据平台。

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 Elastic Load Balancer 只是为松散耦合增加中间层的两种方式。您还可以使用 Amazon EventBridge 在 AWS 云中构建事件驱动型架构，而 Amazon EventBridge 可从其依赖的服务（事件使用者）中提取客户端（事件生成者）。当您需要进行高吞吐量、基于推送的多对多消息收发时，Amazon Simple Notification Service 是可供选择的高效解决方案。通过 Amazon SNS 主题，您的发布者系统可以呈扇形将消息分发到大量订阅者终端节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。

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

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

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

例如，如果运行状况检查系统正在监控 100000 个服务器，它的标称负载低于通常而言较低的服务器故障率。但如果发生重大事件使一半的服务器运行不正常，则运行状况检查系统会因为尝试更新通知系统以及向其客户端传送状态而变得不堪重负。因此，运行状况检查系统每次应发送当前状态的完整快照。100000 个服务器的运行状况，若每个以一个比特代表，则仅需要 12.5-KB 有效负载。无论是没有服务器发生故障还是所有服务器都发生故障，运行状况检查系统都会持续工作，而大幅度骤变也不会威胁到系统的稳定性。这就是实际为 Amazon Route 53 运行状况检查量身定制的控制层面。

资源

视频

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

文档

- 发布 CloudWatch 指标的 AWS 服务
- 什么是 Amazon Simple Queue Service？
- Amazon EC2: 确保幂等性

- Amazon Builders' Library： 分布式系统的挑战
- 集中式日志记录解决方案
- AWS Marketplace：可用于监控和提醒的产品
- APN 合作伙伴：可以帮助您进行监控和日志记录的合作伙伴

在分布式系统中设计交互以缓解或经受住故障的影响

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

这些最佳实践能够预防故障，并改善平均故障间隔时间 (MTBF)。

实现轻松降级以将适用的硬依赖关系转换为软依赖关系：某个组件的依赖关系运行不正常时，该组件仍可在性能降低的条件下运行。例如，当依赖项调用失败时，则使用预先确定的静态响应。

假设被服务 A 调用的服务 B 反过来调用服务 C。

图 5：若服务 B 调用服务 C 失败。服务 B 向服务 A 返回降级响应。

当服务 B 调用服务 C 时，它会收到错误或超时消息。而服务 B，因为缺少来自服务 C（及其所包含数据）的响应，则会返回它能够做出的响应。它可以是最后缓存的正确值，或服务 B 可以使用预先确定的静态响应取代它收到的来自服务 C 的响应。然后，向调用方（即服务 A）返回降级响应。若无此静态响应，服务 C 的故障将级联传递至服务 B 和服务 A，因此而丧失可用性。

按照硬依赖关系可用性等式中的倍乘系数（见 使用硬依赖关系计算可用性 (p. 5)），C 的可用性的任何降低将严重影响 B 的有效可用性。通过返回静态响应，服务 B 能够缓解 C 中的故障的影响，而且，虽然被降级，可使服务 C 看起来似乎 100% 可用（假设它在错误的情况下可靠地返回静态响应）。注意，静态响应是返回错误的简单替代，而不是使用其他方式对响应进行重新计算的尝试。此类采用完全不同的机制试图达到相同结果的尝试被称作回退行为，是一种要被避免的反模式。

优雅降级的另一个例子是 断路器模式。当故障为临时性时，应采用重试策略。若情况并非如此，而且操作很有可能失败，则断路器模式会防止客户端执行可能失败的请求。系统照常处理请求时，断路器会处于关闭状态，让请求正常通过。当远程系统开始返回错误或出现高延迟，断路器就会打开，依赖项被忽略，或者结果会被更轻松获取但较不全面的响应（可能只是响应缓存）所取代。系统将定期尝试调用依赖项，以确定它是否已恢复。出现这种情况时，断路器将处于关闭状态。

图 6：断路器显示关闭或开启状态。

除了图表中显示的关闭和开启状态，在开启状态内的可配置时间段以后，断路器可能会变为半开启状态。在此状态中，它会以远低于正常水平的速率定期尝试调用服务。此探测器用于检查服务的运行状况。在半开启状态中多次成功以后，断路器会转为关闭，并恢复正常请求。

限制请求： 这是对按需求意外增加做出响应的缓解模式。部分请求会得到执行，但超出定义限制的请求会被拒绝，并返回说明它们已被限制的消息。客户端预期将会回退，并且放弃请求或以较低速率进行重试。

您的服务应设计为具有每个节点或单元格可以处理的已知请求容量。这可以通过负载测试建立。然后，您需要跟踪请求到达速率，如果临时到达速率超过此限制，则相应的响应是发出信号表明请求已被限制。这允许用户进行重试，或许会向可能具有可用容量的另一个节点/单元格发出请求。Amazon API Gateway 提供一些限制请求的方法。Amazon SQS 和 Amazon Kinesis 可对请求进行缓冲，平滑请求速率并降低对可异步处理的请求进行限制的需求。

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。引入抖动使此类重试间隔随机化，并限制重试的最大次数。

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。在操作中，受故障影响，任何此类组件都可能开始生成错误。处理错误的默认方式为，在客户端实施重试。此方法可提高应用程序的可靠性和可用性。不过，如果规模较大，而且客户端在错误发生时立即重试失败的操作，网络中的新请求和重试请求可能快速饱和，并相互争抢网络带宽。这可能导致 重试风暴， 从而降低服务的可用性。此模式可能会继续，直到发生全系统故障。

为避免出现此情况，应使用 回退算法， 如常用的 指数回退 。指数回退算法会逐渐降低执行重试的速率，从而避免网络阻塞。

很多开发工具包和软件库（包括 AWS 的开发工具包和软件库）都采用此类算法的一种版本。但是， 别心存侥幸地认为已采用回退算法，始终要进行测试和验证。

仅依靠回退还不够，因为在分布式系统中，所有客户端都可能同步回退，形成重试调用集群。Marc Brooker 在他的博文 指数回退和抖动中解释了如何修改指数回退中的 `wait()` 函数以防止形成重试调用集群。他给出的解决办法 是 在 `wait()` 函数中增加抖动。要避免过长时间的重试，实施应为回退设置一个最大值限制。

最后，务必要配置 最大重试次数 或已用时间，在此以后，重试将失败。AWS SDK 将默认实施此操作，而且也可以对它进行配置。针对堆栈较低的服务，最大重试限制为 0 或 1 将限制风险，但却在重试被委派给堆栈较高的服务时依然有效。

快速试错和限制队列： 如果工作负载无法成功响应请求，则快速试错。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恢复。如果工作负载能够成功响应，但请求速率过高，则使用队列来对请求进行缓冲。不过，不要允许使用长队列，它可能导致处理已被客户端放弃的过时请求。

此最佳实践适用于请求的服务器端，或接收方。

请注意，可在系统的多个级别创建队列，它们可能严重妨碍快速恢复的能力，因为在较新请求需要响应以前，较旧的过时请求会被处理（虽然不再需要响应）。另外还要注意队列所在的位置。它们通常会隐藏在工作流或记录到数据库的工作中。

设置客户端超时：适当设置超时，对它们进行系统性验证，而且不要依赖默认值，因为它们通常设置得过高

此最佳实践适用于请求的客户端，或发送方。

为任何远程调用和大体上任何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，因为许多默认值为无限值或过高。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。过低的值可能因为要重试过多请求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。

要了解关于 Amazon 如何利用超时、重试和抖动回退的更多信息，请参阅 构建者库：超时、重试和抖动回退。

尽可能使服务为无状态：服务应该不需要状态或在不同的客户端请求之间卸载状态，在磁盘上或内存中本地存储的数据不存在依赖关系。这使任意替换服务器成为可能，而且不会对可用性产生影响。Amazon ElastiCache 或 Amazon DynamoDB 是卸载状态的理想目标位置。

图 7：在此无状态 Web 应用程序中，会话状态会被卸载到 Amazon ElastiCache。

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，您则可以使用无服务器计算平台，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。

实施紧急杠杆：这些是可帮助您的工作负载减轻可用性影响的快速流程。即使未找到根本原因，它们也可以运行。理想的紧急杠杆可通过提供完全确定的激活与停用标准，将解析器的认知负担降低到零。杠杆示例包括，阻止所有的机器人流量或静态响应处理。杠杆通常需要手动操作，但也可实现自动化。

关于实施和使用紧急杠杆的提示：

- 当杠杆被激活时，求少不求多
- 保持简单，避免双模态行为
- 定期测试您的杠杆

以下为非紧急杠杆的操作示例：

- 添加容量
- 号召依赖您的服务的客户端服务所有者，要求他们降低调用
- 更改代码并将其释放

资源

视频

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

文档

- AWS 中的错误重试和指数退避
- Amazon API Gateway： 对 API 请求限流以提高吞吐量
- Amazon Builders' Library： 超时、重试和抖动回退
- 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

实验室

- 架构完善的实验室 第 300 级：实施运行状况检查和管理依赖项以提高可靠性

外部链接

- CircuitBreaker（在著作《发布！》中对断路器进行了总结）

图书

- Michael Nygard“发布！ 设计和部署生产就绪的软件”

变更管理

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

以下各节将介绍变更管理的最佳实践。

主题

监控工作负载资源

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

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

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

AWS 的监控包含四个不同的阶段：

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

生成 — 为工作负载监控全部组件： 使用 Amazon CloudWatch 或第三方工具监控工作负载组件。使用 AWS Personal Health Dashboard 监控 AWS 服务。

应监控您的工作负载的全部组件，包括前端、业务逻辑和存储层。定义关键指标以及如何将其从日志中提取出来（如有必要），并且为对应的警报事件设置阈值

云中监控创造新的机会。大多数云提供商都已经开发出可自定义的挂钩，而且对您的工作负载的多个层有深入的了解。

AWS 提供了大量监控和日志信息以供使用，可用来定义按需更改流程。下面是生成日志和指标数据的部分服务和功能列表。

- Amazon ECS、Amazon EC2、Elastic Load Balancing、AWS Auto Scaling 和 Amazon EMR 发布 CPU、网络 I/O 和磁盘 I/O 平均值指标。
- 可以启动 Amazon Simple Storage Service (Amazon S3)、Classic Load Balancers 和 Application Load Balancers 的 Amazon CloudWatch Logs。

- 可启用 VPC 流日志以便对传入与传出 VPC 的网络流量进行分析。
- AWS CloudTrail 会记录 AWS 账户活动，包括通过 AWS Management Console、AWS 开发工具包和命令行工具执行的操作。
- Amazon EventBridge 可提供一个实时的系统事件流，描述 AWS 服务中的更改。
- AWS 提供了工具，用于收集操作系统级别的日志并将其流式传输到 CloudWatch Logs 中。
- 自定义 Amazon CloudWatch 指标可用于任何维度的指标。
- Amazon ECS 和 AWS Lambda 将日志数据传输至 CloudWatch Logs。
- Amazon Machine Learning (Amazon ML)、Amazon Rekognition、Amazon Lex 和 Amazon Polly 提供成功请求和失败请求的指标。
- AWS IoT 提供了有关规则执行数量的指标，以及围绕规则的具体成功和失败指标。
- Amazon API Gateway 提供了关于请求数量、错误请求以及 API 延迟的指标。
- AWS Personal Health Dashboard 可提供 AWS 资源底层的 AWS 服务性能和可用性的个性化视图。

此外，从远程位置监控所有外部终端节点，确保它们独立于基本实施。这种主动监控可通过合成事务实现（有时被称为“用户金丝雀”，但不要与 Canary 部署相混淆），它们会定期执行应用程序使用者的部分常见任务。确保较短的持续时间，以及在测试期间不要使您的工作流过载。Amazon CloudWatch Synthetics 使您能够 创建 Canary，以便监控您的终端节点和 API。您还可以整合 Synthetic Canary 客户端节点和 AWS X-Ray 控制台，精确定位哪些 Synthetic Canary 遇到错误、故障，或对指定时段的速率进行限制的问题。

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

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

实时处理和报警 — 发送通知： 当重大事件发生时，需要知晓的组织会收到通知。

提醒还可以发送到 Amazon Simple Notification Service (Amazon SNS) 主题中，然后推送给任意数量的订阅者。例如，Amazon SNS 可以将提醒转发给某个电子邮件别名，以便技术人员可以回复。

实时处理和报警 — 自动响应： 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。

提醒可以触发 AWS Auto Scaling 事件，以便集群对需求的变化做出反应。提醒还可以发送到 Amazon Simple Queue Service (Amazon SQS)，后者可充当第三方票证系统的集成点。AWS Lambda 还可以订阅提醒，为用户提供一种无服务器的异步模式，以动态方式应对更改。AWS Config 会持续监视和记录您的 AWS 资源配置，并且可以触发 AWS Systems Manager Automation 以修正问题。

存储与分析： 收集日志文件和指标历史，并对其进行分析以获得更广泛的趋势和工作负载见解。

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

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

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

定期进行审核：经常审核工作负载监控的实施情况，并根据重大事件和变更加以更新。

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

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

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

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

对通过系统的请求的端到端跟踪进行监控：利用 AWS X-Ray 或第三方工具，使开发人员可以更轻松地分析与调试分布式系统，理解他们的应用程序及其底层服务的表现。

资源

文档

- 使用 Amazon CloudWatch 指标
- 使用 Canary (Amazon CloudWatch Synthetics)
- Amazon CloudWatch Logs Insights 查询示例
- AWS Systems Manager Automation
- 什么是 AWS X-Ray？
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 进行调试
- Amazon Builders' Library： 检测分布式系统的运营可见性

设计工作负载，以适应需求的变化

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

在获取或扩展资源时利用自动化： 在替换被破坏的资源或扩展您的工作负载时，通过采用托管 AWS 服务（如 Amazon S3 和 AWS Auto Scaling）对流程进行自动化。您还可以使用第三方工具和 AWS 开发工具包自动扩展。

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

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

在对 EC2 实例或托管于 EC2 实例上的 Amazon ECS 容器进行扩展时，确保您使用多个可用区（最好至少三个）并增加或减少容量以保持这些可用区之间的平衡。

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

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

配置和使用 Amazon CloudFront 或受信任的内容分发网络。内容分发网络 (CDN) 可以缩短最终用户的响应时间，还可以对可能导致工作负载进行不必要扩展的内容请求做出响应。

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，或触发自动化以对其进行自动扩展。

可为您的工作负载手动调整扩展，例如，通过控制台或 AWS CLI 更改 Auto Scaling 组中 EC2 实例的数量，或者修改 DynamoDB 表的吞吐量来实现。不过，应在可能的情况下尽量使用自动化（见 在扩展或缩减工作负载时使用自动化）。

当检测到某个工作负载需要更多资源时，就会获取资源：主动扩展资源以满足需求并避免影响可用性。

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

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

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

L = λW

L = 实例数量（或系统中的平均并发值）

λ = 收到请求的平均速率（请求数量/秒）

W = 每个请求在系统中所花的平均时间（秒）

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

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是否能够满足工作负载要求。

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与测试工作负载的性能。AWS 对临时测试环境的设置进行简化，该环境将为您的生产工作负载的扩展建立模型。在云中，您可以根据需要创建一套生产规模等级的测试环境，完成测试，然后停用资源。由于测试环境只需在运行时付费，您模拟真实环境的成本仅为本地测试成本的一小部分。

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

资源

文档

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

- 什么是 Amazon CloudFront？
- AWS 上的分布式负载测试：模拟数千个连接的用户
- AWS Marketplace：可以与 Auto Scaling 一起使用的产品
- APN 合作伙伴：可以帮您制定自动计算解决方案的合作伙伴

外部链接

- 讲述与利特尔法则有关的故事

实施变更

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

对部署等标准活动使用运行手册： 运行手册是用于实现特定结果的预定义步骤。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。例如部署工作负载，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，触发自动执行。

即使是高度自动化的复杂工作负载，运行手册同样适用于 运行实际试用 (p. 31) 或用于满足严格的报告和审计要求。

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

将功能测试作为部署的一部分进行集成： 功能测试作为自动化部署的一部分运行。若未满足成功条件，则相关管道会中止或回滚。

这些测试在预生产环境中运行，该环境会在管道中的生产开始前被暂存。在理想情况下，此操作是部署管道的一部分。

将弹性测试作为部署的一部分进行集成： 将弹性测试（混沌工程的一部分）作为预生产环境中自动化部署管道的一部分执行。

在生产开始前，这些测试会在管道中暂存与运行。它们应在生产中运行，但作为 实际试用 (p. 31)的一部分。

使用不可变基础设施部署：这种模式要求在生产系统上不会 就地 出现更新、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。

最常被实施的不可变基础设施范式为 不可变服务器。这意味着，若服务器需要更新或修复，将部署新的服务器，而不是对使用中的服务器进行更新。因此，相对于通过 SSH 登录到服务器并更新软件版本，应用程序的每次更改都会在开始时将软件推送到代码库，如 git 推送。由于在不可变基础设施中不允许更改，您可以确定已部署系统的状态。不可变基础设施在本质上具有更稳定、可靠和可预测的特性，它们对软件开发和运行的多个方面进行了简化。

当您在不可变基础设施中部署应用程序时，使用 Canary 或蓝绿部署。

金丝雀部署 是将您的少量客户引导到新版本的做法，它通常在单个服务实例 (Canary) 上运行。然后，您可以深入检查生成的任何行为更改或错误。如果遇到了严重问题，您可以将 Canary 中的流量删除，并将用户发回到以前的版本。如果部署成功，您可以继续以期望的速度进行部署，同时监控更改以便发现错误，直到所有部署完成。AWS CodeDeploy 的部署配置可以配置为启用 Canary 部署。

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

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

不可变基础设施的优点：

- 减小配置偏差： 通过从基本、已知，而且版本受控的配置频繁替换服务器，基础设施会被 重置 为已知状态，以避免配置偏差。
- 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。
- 可靠的原子部署： 成功完成部署，或没有任何更改。它让您更信任部署流程。
- 采用快速回滚和恢复流程的更安全部署： 由于之前运行的版本未发生更改，因此部署变得更安全。您可以在检测到错误时进行回滚。
- 一致的测试和调试环境： 由于所有服务器都使用相同的映像，因此环境之间没有任何差异。同一个版本被部署到多个环境。它还防止出现不一致的环境，并且简化测试与调试。
- 增强可扩展性： 服务器都使用一个基础映像，它们是一致、可重复的，自动扩展并不重要。
- 简化工具链：您无需采用配置管理工具对生产软件升级进行管理，因此工具链也得到简化。也不需要在服务器上安装其他工具或代理。对基础映像进行更改，然后在经过测试后实施。
- 提高安全性： 通过拒绝对服务器的所有更改，您可以在实例上禁用 SSH 并移除密钥。这样做可以减少攻击载体，改善您的组织的安全状况。

使用自动化功能部署更改：自动部署与修补以消除负面影响。

对许多组织来说，对生产系统进行变更是风险最大的工作之一。除了软件解决的业务问题外，我们认为部署也是亟待解决的首要问题。如今，这意味着根据实际情况在操作中使用自动化，包括测试和部署更改、添加或删除容量以及迁移数据。AWS CodePipeline 让您可以管理释放您的工作负载所需的步骤。其中包括，采用 AWS CodeDeploy 将应用程序代码自动部署到 Amazon EC2 实例、本地实例、无服务器 Lambda 函数或 Amazon ECS 服务的部署状态。

推荐
虽然传统智慧告诉我们，循环中最困难的操作程序应该由人来负责，但出于相同的原因，我们建议您将最困难的程序自动化。

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，这样您的客户就看不到这个功能了。然后可以像处理 Canary 部署那样启用该功能，也可以将更改速度设置为 100% 以查看效果。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。

运维准备情况审查 (ORR)

AWS 发现，执行运维准备情况审查非常有用，它可以评估测试的完整性、监控能力，更重要的是，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。正式的 ORR 会在初

始生产部署之前实施。AWS 会定期重复 ORR（每年一次或在关键性能阶段之前），以确保没有“偏离”运营预期。如需关于运维准备情况的更多信息，请参阅 卓越运营支柱 – AWS 架构完善的框架。

推荐
在首次投入生产使用前为应用程序执行运维准备情况审查 (ORR)，并在之后定期开展。

资源

视频

- 2019 年 AWS 峰会：AWS 上的 CI/CD

文档

- 什么是 AWS CodePipeline？
- 什么是 CodeDeploy？
- 蓝绿部署概览
- 逐步部署无服务器应用程序
- 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
- AWS Marketplace：可用于自动实施部署的产品
- APN 合作伙伴：可以帮助您创建自动化部署解决方案的合作伙伴

实验室

- 架构完善的实验室 第 300 级：测试 EC2 RDS 和 S3 的弹性

外部链接

- CanaryRelease

故障管理

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

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

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

以下各节将介绍管理故障以预防对您的工作负载产生影响的最佳实践：

主题

备份数据

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

识别和备份需要备份所有数据，或从源复制数据：Amazon S3 可用作多个数据源的备份目标位置。Amazon EBS、Amazon RDS 和 Amazon DynamoDB 等 AWS 服务具有创建备份的内置功能。或者，您也可以使用第三方备份软件。另外，如果可以从源中复制数据以满足 RPO 要求，您可能不需要进行备份。

使用 Amazon S3 存储桶和 AWS Storage Gateway 可将本地数据备份到 AWS 云。您还可以通过 Amazon S3 Glacier 或 S3 Glacier Deep Archive 对备份数据进行存档，以获得经济实惠，而且非时效性的云储存。

如果您从 Amazon S3 将数据加载到数据仓库（如 Amazon Redshift），或 MapReduce 集群（如 Amazon EMR），以便对此类数据进行分析，这可能是可从其他数据源复制数据的例子。只要此类分析的结果被存储在某位置，或者可重现，您不会因为数据仓库或 MapReduce 集群故障而遭遇数据丢失的情况。其他可从数据源复制数据的例子包括，缓存（如 Amazon ElastiCache）或 RDS 只读副本。

保护并加密备份：通过身份验证和授权（如 AWS Identity and Access Management (IAM)）检测访问，并使用加密检测数据完整性是否受损。

Amazon S3 支持多种对您的静态数据进行加密的方式。借助服务器端加密，Amazon S3 接受您的未加密数据对象，然后在保留此类数据前对其进行加密。若采用客户端加密，您的工作负载应用程序要负责在将其发送到 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) 创建与存储数据密钥，或者您也可以提供自己的密钥（您要对其负责）。使用 AWS KMS，您可以通过 IAM 设置策略，决定谁可以以及谁不可以访问您的数据密钥与解密数据。

针对 Amazon RDS，如果您已选择对数据库进行加密，那么您的备份也会被加密。DynamoDB 备份则始终被加密。

自动执行数据备份： 将备份配置为根据定期计划自动备份，或在数据集发生更改时自动备份。RDS 实例、EBS 卷、DynamoDB 表和 S3 对象均可配置为自动备份。您还可以使用 AWS Marketplace 解决方案或第三方解决方案。

Amazon Data Lifecycle Manager 可用于自动拍摄 EBS 快照。Amazon RDS 和 Amazon DynamoDB 提供支持时间点恢复的持续备份。一旦启用，Amazon S3 版本控制即是自动的。

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 将云中和本地的多项 AWS 服务的数据备份集中在一起并进行自动化。

除了版本控制，Amazon S3 还具有复制功能。整个 S3 存储桶都可被自动复制到不同 AWS 区域的其他存储桶。

定期执行数据恢复以验证备份完整性和流程：通过执行恢复测试，验证您的备份流程实施是否满足恢复时间目标 (RTO) 和恢复点目标 (RPO) 要求。

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

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

资源

视频

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

文档

- 什么是 AWS Backup？
- Amazon S3：利用加密保护数据
- AWS 中的备份的加密
- DynamoDB 的按需备份和还原
- EFS 到 EFS 备份
- AWS Marketplace：可以用于备份的产品
- APN 合作伙伴：可以帮助进行备份的合作伙伴

实验室

- 架构完善的实验室 第 200 级：测试数据备份与还原
- 架构完善的实验室 第 200 级：为 Amazon Simple Storage Service (Amazon S3) 实施双向跨区域复制 (CRR)

使用故障隔离来保护您的工作负载

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

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

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

可用区

AWS 区域由在设计上相互独立的两个或更多可用区组成。每个可用区之间的物理距离相隔很远，以避免因环境公害（如火灾、洪水和龙卷风）导致的相互关联的故障情况。每个可用区拥有独立的物理基础设施：专用的公用电源连接、独立备份电源、独立机械服务以及可用区内外的独立网络连接。尽管可用区在地理位置上相互分离，但它们位于相同的区域中。这可以在不过度影响应用程序延迟的情况下实现同步数据复制（例如数据库之间的复制）。这样一来，客户便能在主动/主动或主动/备用配置中使用可用区。可用区是独立的，因此使用多个区可以提高应用程序可用性。某些 AWS 服务（包括 EC2 实例数据平面）会被部署为严格区域服务，与可用区一起作为一个整体共存亡。这些服务用于在特定可用区内独立运行资源（实例、数据库和其他基础设施）。AWS 在我们的区域中具有多个长期提供的可用区。

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

虽然 AWS 控制平面通常提供在整个区域（多个可用区）内管理资源的功能，但某些控制平面（包括 Amazon EC2 和 Amazon EBS）能够将结果筛选到单个可用区。完成筛选后，请求仅在指定可用区中进行处理，从而降低其他可用区的中断风险。另一方面，主动/主动配置的区域 AWS 服务在内部使用多个可用区，以实现我们设定的可用性设计目标。

推荐
当您的应用程序在某个 可用区中断期间依赖于控制平面的可用性，使用 API 筛选条件请求每个 API 请求的单个可用区结果（如 DescribeInstances）。

AWS Local Zones

AWS Local Zones 在其各自的 AWS 区域内的作用和可用区相似，它们可被选择作为区域 AWS 资源（如子网和 EC2 实例）的置放位置。特别之处在于，它们并不位于相关的 AWS 区域内，而是位于目前还未设置 AWS 区域的人口密集的工业和 IT 中心。但是，您还是可以享有高带宽，而且在本地扩展区的本地工作负载之间以及在 AWS 区域内运行的本地工作负载之间进行安全连接。您应该利用 AWS 本地扩展区将工作负载尽量部署在接近用户的地方，以满足低延迟的要求。

Amazon 全球边缘网络

Amazon 全球边缘网络由全球各大城市的边缘站点组成。Amazon CloudFront 使用此网络以较低的延迟向最终用户分发内容。您可以通过 AWS Global Accelerator 在这些边缘站点创建您的工作负载终端节点，以便接入与您的用户接近的 AWS 全球网络。Amazon API Gateway 让使用 CloudFront 分配的边缘优化 API 终端节点可以通过最近的边缘站点方便客户端访问。

AWS 区域

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

推荐
工作负载的大多数可靠性目标都可通过在单个 AWS 区域内采用多可用区策略来实现。只有当工作负载有多区域的要求时，您才应该考虑多区域架构。

AWS 让客户能够跨区域运行服务。例如，Amazon Aurora Global Database、Amazon DynamoDB 全局表、Amazon S3 的跨区域复制、采用 Amazon RDS 的跨区域只读副本，以及将多个快照和 Amazon 系统

映像 (AMI) 复制到其他区域的能力。不过，我们采用的方法会保护区域的自主权。这种方法的例外很少，包括我们提供全球边缘交付的服务（例如 Amazon CloudFront 和 Amazon Route 53），以及的 AWS Identity and Access Management (IAM) 服务的控制平面。绝大多数服务都完全在单个区域中运行。

本地数据中心

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

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

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，您必须利用相关功能在定义的恢复目标内彻底重建工作负载。

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

例如，Amazon EMR 会为相同可用区内的特定集群启动全部节点，因为在相同区内运行集群可以改善作业流的性能，提高数据访问速率。如果是工作负载弹性所需的必要组件，则您必须有办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 多个主节点。使用 EMR 文件系统 (EMRFS)，EMR 中的数据可存储在 Amazon S3 内，进而可以实现跨多个可用区或 AWS 区域复制。

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

采用隔板架构类似于船上的隔板，此模式确保将故障限制在较小的请求/用户子集，受损的请求数量有限，因此大部分可以继续执行而不会受错误影响。数据的隔板经常被称作 分区 或者 分片，而服务的隔板称为 单元格。

在 基于单元格的架构中，每个单元格都是完整、独立的服务实例，而且都有固定的最大大小。当负载增加时，工作负载会通过添加更多单元格而变大。分区键用于传入流量，以确定哪个单元格将处理请求。任何故障都会被限制在它所发生的单个单元格内，因此受损请求的数量有限，而其他单元格将继续执行而不受错误影响。确定适当的分区键，最大限度地减少跨单元格交互，以便使每个请求无需使用复杂的映射服务，这一点非常重要。需要复杂映射的服务最终只是把问题转移到映射服务上，而需要跨单元格交互的服务会在单元格之间创建依赖关系（因此这样做会减少假定的可用性改进）。

图 10：基于单元格的架构

Colm MacCarthaigh 在他的 AWS 博客中说明了 Amazon Route 53 如何利用 随机分区 的概念来隔离客户请求以避免影响其他分区。在此情况下，一个分区由两个或更多单元格组成。根据分区键，来自客户（或资源，或其他您想要隔离的对象）的流量会被路由至其指定的分区。若有八个单元格（每个分区中有两个单元格），而且在四个分区中划分客户，25% 的客户将在出现问题时受到影响。

图 11：在四个传统分区内划分服务，每个分区有两个单元格

通过随机分区，您可以创建由两个单元格组成的虚拟分区，然后将您的客户指定给其中的一个虚拟分区。当问题发生时，您还是会失去完整服务的四分之一，但分配客户或资源的方式意味着若采用随机分区，影响的范围会在很大程度上小于 25%。在八个单元格中，存在着 28 种由两个单元格组成的独特组合，亦即有 28 种可能的随机分区（虚拟分区）。如果您有数百或数千个客户，并将每个客户指定给一个随机分区，那么问题的影响范围仅为 1/28。这比正常分区的情况好七倍。

图 12：服务被划分到 28 个随机分区（虚拟分区），每个分区由两个单元格组成（仅显示 28 种可能中的两个随机分区）

除了单元格，分区还可用于服务器、队列或其他资源。

资源

视频

- AWS re:Invent 2018：适用于多区域主动-主动应用程序的架构模式 (ARC209-R2)
- 随机分区：AWS re: Invent 2019：Amazon Builders' Library 简介 (DOP328)
- AWS re:Invent 2018：AWS 如何将故障的影响范围最小化 (ARC338)
- AWS re: Invent 2019：AWS 全球网络基础设施的创新与运营 (NET339)

文档

- 什么是 AWS Outposts？
- 全局表：使用 DynamoDB 的多区域复制
- AWS Local Zones 常见问题
- AWS 全球基础设施
- 区域、可用区和本地区
- Amazon Builders' Library： 采用随机分区进行工作负载隔离

将工作负载设计为能够承受组件故障的影响

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

监控工作负载的所有组件以检测故障： 持续监控您的工作负载的运行状况，以便您和您的自动化系统在性能下降或发生全面故障时立即察觉。监控基于商业价值的关键性能指标 (KPI)。

所有恢复和修复机制必须从快速检测问题的能力入手。首先，应该检测技术故障并加以解决。不过，可用性基于您的工作负载创造业务价值的能力，因此它需要成为您的检测和补救策略的关键指标。

故障转移到运行状况良好的资源：确保如果某个资源发生故障，该运行状况良好的资源可以继续为请求提供服务。对于位置故障（如可用区或 AWS 区域），确保您拥有适当的系统以故障转移到未受损位置内运行状况良好的资源。

对于单个资源故障（如 EC2 实例）或多可用区工作负载中的可用区受损，这更简单，因为 AWS 服务（如 Elastic Load Balancing 和 AWS Auto Scaling）可帮助在资源和可用区之间分配负载。对于多区域工作负载就比较复杂。例如，跨区域只读副本让您可以将数据部署到多个 AWS 区域，但您仍必须提升只读副本至主节点，并在主要位置发生故障时将您的流量指向该节点。Amazon Route 53 和 AWS Global Accelerator 还可以帮助跨 AWS 区域路由流量。

如果您的工作负载使用 Amazon S3 或 Amazon DynamoDB 等 AWS 服务，则它们会自动部署到多个可用区。当发生故障时，AWS 控制平面会自动为您将流量路由至运行正常的位置。针对 Amazon RDS，您必须选择多可用区作为配置选项，然后在发生故障时，AWS 会自动将流量定向至运行正常的实例。对于 Amazon EC2 实例或 Amazon ECS 任务，您要选择部署到哪些可用区。然后，Elastic Load Balancing 会提供解决方案以检测运行不正常区内的实例，并将流量路由至运行正常的区。Elastic Load Balancing 甚至可以将流量路由至本地数据中心内的组件。

针对多区域方案（也可能包括本地数据中心），Amazon Route 53 会提供定义互联网域并指定路由策略的方式，而此类策略可能包含运行状态检查，以确保流量被路由至运行正常的区域。此外，AWS Global Accelerator 也可以提供静态 IP 地址作为您的应用程序的固定接入点，然后通过 AWS 全球网络而不是互联网路由至您选择的 AWS 区域内的终端节点，以提高性能和可靠性。

AWS 在设计服务时始终会考虑恢复功能。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会在请求持续存储在多个副本后再确认请求。这些服务和资源包括 Amazon Aurora、Amazon Relational Database Service (Amazon RDS) 多可用区数据库实例、Amazon S3、Amazon DynamoDB、Amazon Simple Queue Service (Amazon SQS) 和 Amazon Elastic File System (Amazon EFS)。它们被构建为使用基于单元格的隔离，并使用可用区提供的故障隔离功能。我们在自己的运营过程中广泛使用自动化。我们还将替换和重新启动功能优化为可从中断快速恢复。

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。

重启功能 是用于修复故障的重要工具。正如我们之前在分布式系统部分讨论过那样，最佳实践是尽可能使服务为无状态。它可以防止重启时数据丢失或可用性受损。您可以（而且在一般情况下也应该）在云中替换完整的资源（如 EC2 实例或 Lambda 函数），并将其作为重启的一部分。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。将众多不同类别的故障映射到相同的恢复策略上，而不是构建新颖的机制来捕获、确定和纠正各个不同类型的故障。实例可能会因为硬件故障、操作系统错误、内存泄漏或其他原因而出现故障。系统不会针对每种情况构建自定义修复，而是会将它们全部视为实例故障。终止实例，并且允许使用 AWS Auto Scaling 进行取代。然后，在带外对故障资源进行分析。

另一个例子是重启网络请求功能。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，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 (ROC) 和高可用性集群架构的特色恢复机制。

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，它可以触发 AWS Lambda（或其他目标）在您的工作负载上执行自定义修复逻辑。

Amazon EC2 Auto Scaling 可以配置为对 EC2 实例的运行状况进行检查。若实例处于正在运行以外的任何状态，或系统状态受损，Amazon EC2 Auto Scaling 会认为实例的运行不正常，并且启动替换实例。如果使用 AWS OpsWorks，您可以在层级别配置 EC2 实例的自动修复。

针对大规模替换（如整个可用区受损），静态稳定性更适合高可用性，而不是立即尝试获取多个新的资源。

使用静态稳定性来防止双模态行为： 双模态行为是指您的工作负载在正常和故障模式下展现出不同的行为，例如，若可用区发生故障时依赖于启动新的实例。您应该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，如果删除了一个可用区，要在每个可用区内预置足够的实例来处理工作负载，然后再使用 Elastic Load Balancing 或 Amazon Route 53 运行状况检查将负载从受损实例中转出。

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。您必须在稳定性水平和成本之间认真权衡。预置较小的计算容量，并在发生故障时依赖启动新实例，其成本较低。但对于大规模故障（如可用区故障）来说，此方法的效果较差，因为它依赖于对发生的损坏做出反应，而不会在损坏发生前做好准备。您选择的解决方案应在工作负载的可用性和成本需求之间做出取舍。若使用更多可用区，静态稳定性所需的额外计算量就会减少。

图 13：在流量转移以后，使用 AWS Auto Scaling 异步替换故障区的实例，并且在运行正常的区内启动。

双模态行为的另一个例子是，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，触发其他意外后果。此负面的反馈环路会影响您的工作负载的可用性。您应该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 (p. 13)，并且始终定期刷新配置状态。当调用失败时，工作负载会使用之前的缓存值，并触发警报。

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

当事件影响可用性时发出通知：在检测到重大事件时发送通知，即使由事件引发的问题已经自动解决。

自动修复使您的工作负载变得可靠。不过，它也可能会掩盖需要处理的潜在问题。实施适当的监控和措施，以便检测问题的模式，包括那些被自动修复的问题，从而从根本上解决问题。Amazon CloudWatch 警报会基于发生的故障触发。它们还可能由于执行自动修复操作而被触发。CloudWatch 警报可被配置为发送电子邮件，或使用 Amazon SNS 集成将事件记录到第三方事件跟踪系统。

资源

视频

- AWS 中的静态稳定性：AWS re: Invent 2019：Amazon Builders' Library 简介 (DOP328)

文档

- AWS OpsWorks： 使用自动修复来更换失败的实例
- 什么是 Amazon EventBridge？
- Amazon Route 53：选择一个路由策略
- 什么是 AWS Global Accelerator？
- Amazon Builders' Library： 使用可用区的静态稳定性
- Amazon Builders' Library： 实施运行状况检查
- AWS Marketplace：可以支持容错的产品
- APN 合作伙伴：可以帮助您实现容错自动化的合作伙伴

实验室

- 架构完善的实验室 第 300 级：实施运行状况检查和管理依赖项以提高可靠性

外部链接

- 加州大学伯克利分校/斯坦福大学的面向恢复的计算 (ROC) 项目

测试可靠性

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

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

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

行动手册是您必须要执行的主动计划，以便有效采取被动措施。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，而行动手册则被用于对非例行事件做出响应。

在意外事件发生后执行分析： 审核影响客户的事件，确定这些事件的成因和预防措施。利用这些信息来制定缓解措施，以限制或防止再次发生同类事件。制定程序以迅速有效地做出响应。针对目标受众适当地传达导致因素和纠正措施。

评估为什么现有测试找不到问题。如果还没有，增设测试。

测试功能要求：这包括用于验证所需功能的单元测试和集成测试。

如果这些测试作为构建和部署措施的一部分自动运行，则您可以获得最佳的结果。例如，使用 AWS CodePipeline，开发人员会在 CodePipeline 自动检测到变更时提交对源存储库的更改。执行更改，然后加以测试。在测试完成以后，构建的代码会被部署到用于测试的暂存服务器。CodePipeline 会从暂存服务器运行更多测试，如集成或负载测试等。在成功完成此类测试以后，CodePipeline 会将经过测试并获得批准的代码部署到生产实例。

此外，过去的经验告诉我们，可运行与模拟用户行为的合成事务测试（又被称作“Canary 测试”，但不要和 Canary 部署相混淆）是最重要的测试流程之一。从不同的远程位置针对您的工作负载终端节点持续地运行此类测试。Amazon CloudWatch Synthetics 使您能够 创建 Canary 以便监控您的终端节点和 API。

测试扩展和性能要求：这包括负载测试以验证工作负载是否满足扩展和性能要求。

在云中，您可以按需为您的工作负载创建生产规模环境。如果在缩减的基础设施上运行这些测试，您必须根据您认为在生产中将会发生的情况扩展您观察到的结果。如果不想影响实际用户，您可以在生产中开展负载和性能测试，并且对您的测试数据进行标记，以避免它与真实的用户数据、损坏的使用情况统计或生产报告混在一起。

通过测试确保您的基础资源、扩展设置、服务配额和弹性设计能够在负载之下如预期运行。

使用混沌工程测试弹性：运行定期将故障注入到预生产和生产环境的测试。假设您的工作负载将会如何对故障做出反应，然后比较您的假设和测试结果，若不相符，重复执行。确保生产测试不会影响用户。

在云中，您可以测试工作负载的故障情况，并验证您的恢复程序。您可以采用自动化方式来模拟不同的故障，也可以重新建立之前导致故障的场景。这样做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。

混沌工程是对系统进行试验以让人们确信系统能够在生产中经受住混乱情形的规范。– 混沌工程的原则

在预生产和测试环境中，应定期执行混沌工程，并将其作为 CI/CD 周期的一部分。在生产中，团队必须注意不要中断可用性，而且应利用 实际试用 作为在生产中控制混沌工程风险的方式。

测试工作应与您的可用性目标相符。请通过测试来确保您可以实现可用性目标，这是您能够确信自己将会实现这些目标的唯一方法。

执行您为工作负载设计的预防组件故障的弹性测试。其中包括 EC2 实例丢失、主 Amazon RDS 数据库实例故障，以及可用区中断。

测试外部依赖项的不可用性。您的应用工作负载在依赖项出现暂时性故障时的恢复能力应当经过测试，持续时间可能从一秒以内到几个小时。

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）。您希望能够将此类故障注入系统，包括网络影响（如延迟和消息丢失），以及 DNS 故障（如无法解析名称或无法与依赖的服务建立连接）。

您可以选择多种第三方选项进行故障注入。其中包括开源选项，例如 Netflix Chaos Monkey、 Chaos ToolKit和 Shopify Toxiproxy以及其他商业选项，如 Gremlin。我们建议在最初针对如何实施混沌工程进行调查时采用自己编写的脚本。这样做可以使工程团队更快地适应将混沌工程引入到他们的工作负载当中。有关这些示例，请参阅 测试 EC2 RDS 和 S3 的弹性 使用多种语言，如 Bash、Python、Java 和 PowerShell。您还应实施 通过 AWS Systems Manager 将混沌注入到 Amazon EC2，它让您可以使用 AWS Systems Manager 文档模拟电压过低和高 CPU 利用率的情况。

定期进行实际测试：通过定期进行实际测试，与将参与实际故障情景的人员一起在尽可能接近生产环境的环境中（包括在生产环境中）练习您的故障程序。实际测试会强制执行相关措施，以确保生产测试不会影响用户。

通过定期安排实际演练来模拟生产中的各种事件，测试架构和流程的性能。这将帮助您了解可以从哪些方面做出改进，并有助于组织总结积累处理各种事件的经验。

在非生产环境中对您的弹性设计进行测试以后，可通过 Game Day 确保生产中的一切按照计划运行。Game Day，尤其如果是首次开展，是所有人员都应该参加的活动，工程师和运营团队都会得到关于开展时间以及活动内容的信息。行动手册准备就绪。然后，以指定的方式将故障注入到生产系统，接着对其影响进行评

估。如果所有系统如设计运行，检测和自我修复不会产生或只会产生非常轻微的影响。但如果观察到负面影响，测试将会回滚，并且（使用行动手册）修复问题，在必要时手动修复。由于 Game Day 活动在生产中进行，所以应采取全部预防措施，以确保不会对客户造成可用性影响。

资源

视频

- AWS re: Invent 2019：通过混沌工程提高弹性 (DOP309-R1)

文档

- 持续交付和持续集成
- 使用 Canary (Amazon CloudWatch Synthetics)
- 将 CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行构建
- 使用 AWS Systems Manager 自动执行您的运营手册
- AWS Marketplace：可用于实现持续集成的产品
- APN 合作伙伴：可帮助实施持续集成管道的合作伙伴

实验室

- 架构完善的实验室 第 300 级：测试 EC2 RDS 和 S3 的弹性

外部链接

- 混沌工程的原则
- 弹性工程：学会接受故障
- Apache JMeter

图书

- Casey Rosenthal、Lorin Hochstein、Aaron Blohowiak、Nora Jones、Ali Basiri。“混沌工程”（2017 年 8 月）

灾难恢复 (DR) 计划

拥有适当的备份和冗余工作负载组件是您的 DR 策略的开始。RTO 和 RPO 是您恢复可用性的目标。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。

定义停机和数据丢失的恢复目标：工作负载具有恢复时间目标 (RTO) 和恢复点目标 (RPO)。

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

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

使用定义的恢复策略来实现恢复目标：定义了灾难恢复 (DR) 策略以满足工作负载目标。

除非您需要多区域策略，否则我们建议您使用 AWS 区域内的多可用区来实现您在 AWS 中的恢复目标。

如有必要，在为您的工作负载制定多区域策略时，您应该选择以下策略中的一种。这些策略按照复杂程度升序排列，以及按 RTO 和 RPO 降序排列。DR 区域 指的是，除用于您的工作负载以外的 AWS 区域（或者任何 AWS 区域，若您的工作负载位于本地）。

- 备份和还原（RPO 以小时为单位，RTO 为 24 小时或以内）：将您的数据和应用程序备份到 DR 区域。当需要从灾难中恢复时还原这些数据。
- 指示灯（RPO 以分钟为单位，RTO 以小时为单位）：在 DR 区域内维持最小版本的环境始终运行您的系统的最关键的核心元素。在需要恢复时，您可以围绕关键核心快速预置全面的生产环境。
- 热备用（RPO 以秒为单位，RTO 以分钟为单位）：保证在 DR 区域始终运行缩减版本的全功能环境。业务关键型系统是完全重复，而且始终可用的系统，只是其队列的规模经过缩减。在需要恢复时，系统会快速扩展以处理生产负载。
- 多区域主动-主动（RPO 为无或可能以秒为单位，RTO 以秒为单位）：您的工作负载被部署到多个 AWS 区域，并且主动处理来自这些区域的流量。此策略要求您在使用的区域之间对用户和数据进行同步。在需要恢复时，使用诸如 Amazon Route 53 或 AWS Global Accelerator 之类的服务，以便将您的用户流量路由到工作负载运行正常的位置。

推荐
指示灯和热备份之间的差异有时难以区分。两者都包含在您的 DR 区域内运行的环境。在两者之间，如果您的 DR 策略涉及到部署额外的基础设施，则使用指示灯。如果它仅涉及纵向和横向扩展现有基础设施，则使用热备份。根据您的 RTO 和 RPO 需求在两者之间进行选择。

测试灾难恢复实现以验证实现效果：定期测试 DR 故障转移以确保满足 RTO 和 RPO 目标。

要避免的模式是开发很少执行的恢复路径。例如，您可能有一个用于只读查询的辅助数据存储。当您写入某个数据存储，却发现主存储故障时，您可能希望将故障转移到辅助数据存储。如果您不经常测试此故障转移，可能会发现您关于辅助数据存储容量的假设是错误的。辅助数据存储容量在您上次测试时可能是足够的，但可能无法再容纳这次情况下的负载。我们的经验表明，唯一有效的错误恢复是您经常测试的路径。因此，最好只开发几条恢复路径。您可以建立恢复模式并定期对其进行测试。如果恢复路径比较复杂或至关重要，您仍需定期在生产环境中执行该故障，让自己确信恢复路径有效。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。

管理 DR 站点或区域的配置漂移：确保 DR 站点或区域的基础设施、数据和配置满足需求。例如，检查 AMI 和服务配额是否为最新。

AWS Config 会持续监视和记录您的 AWS 资源配置。它可以检测到偏差并触发 AWS Systems Manager Automation 进行修复和发出警报。AWS CloudFormation 还可以在您已部署的堆栈中检测到偏差。

自动执行恢复：利用 AWS 或第三方工具自动进行系统恢复，并将流量路由至 DR 站点或区域。

根据已配置的运行状况检查，AWS 服务（如 Elastic Load Balancing 和 AWS Auto Scaling）可以将负载分配到运行正常的可用区，而服务（如 Amazon Route 53 和 AWS Global Accelerator）则可将负载路由到运行正常的 AWS 区域。

针对现有物理或虚拟数据中心，或者私有云 CloudEndure Disaster Recovery（可在 AWS Marketplace 中获取）上的工作负载，使组织可以为 AWS 设置自动灾难恢复策略。CloudEndure 还支持 AWS 中的跨区域/跨可用区灾难恢复。

资源

视频

- AWS re: Invent 2019：AWS 的备份与还原，以及灾难恢复解决方案 (STG208)

文档

- 什么是 AWS Backup？
- 按照 AWS Config Rules修正不合规 AWS 资源
- AWS Systems Manager Automation
- AWS CloudFormation： 在整个 CloudFormation 堆栈上检测偏差
- Amazon RDS：跨区域备份副本
- RDS：跨区域复制只读副本
- S3：跨区域复制
- Route 53：配置 DNS 故障转移
- CloudEndure 灾难恢复
- 如何在 AWS 上实施 基础设施配置管理 解决方案？
- AWS 的 CloudEndure Disaster Recovery
- AWS Marketplace：可以用于灾难恢复的产品
- APN 合作伙伴：可以帮助进行灾难恢复的合作伙伴

可用性目标的示例实施

在这个部分，我们将介绍采用典型 Web 应用程序部署的工作负载设计，其中包括：反向代理、Amazon S3 上的静态内容、应用程序服务器以及用于持久性数据存储的 SQL 数据库。对于每个可用性目标，我们都将介绍一个实施示例。此工作负载可能将容器或 AWS Lambda 用于数据库的计算和 NoSQL（如 Amazon DynamoDB），但所采用的方法都是相似的。在每个情境中，我们将介绍如何通过针对此类主题的工作负载设计来达成可用性目标：

主题	有关更多信息，请参阅此章节
监控资源	监控工作负载资源 (p. 17)
适应需求的变化	设计工作负载，以适应需求的变化 (p. 19)
实施变更	实施变更 (p. 21)
备份数据	备份数据 (p. 24)
构建弹性	使用故障隔离来保护您的工作负载 (p. 25) 将工作负载设计为能够承受组件故障的影响 (p. 28)
测试弹性	测试可靠性 (p. 30)
灾难恢复 (DR) 计划	灾难恢复 (DR) 计划 (p. 32)

依赖项选择

我们已选择将 Amazon EC2 用于应用程序。我们将介绍如何使用 Amazon RDS 和多个可用区来提高应用程序的可用性。我们会将 Amazon Route 53 用于 DNS。使用多个可用区时，我们将使用 Elastic Load Balancing。Amazon S3 用于备份和静态内容。由于我们的设计追求更高的可靠性，因此必须采用本身具有更高可用性的服务。查看 附录 A：旨在展示所选 AWS 服务的可用性 (p. 50) 以实现相应 AWS 服务的设计目标。

单区域场景

主题

- 2 个 9 (99%) 场景 (p. 35)
- 3 个 9 (99.9%) 场景 (p. 37)
- 4 个 9 (99.99%) 场景 (p. 38)

2 个 9 (99%) 场景

这些工作负载对业务有帮助，但如果不可用， 它们只会带来 不便。此类工作负载可以是内部工具、内部知识管理或项目跟踪。或者，它们可以是来自实验性服务但实际面向客户的工作负载，并且具有在必要时切换以隐藏该服务的功能。

这些工作负载可以部署为一个区域和一个可用区。

监控资源

我们将进行简单的监控，看看服务主页是否会返回 HTTP 200 OK 状态。出现问题后，我们的行动手册会说明，可以使用实例中的日志来确定根本原因。

适应需求的变化

我们拥有关于常见硬件故障、紧急软件更新和其他破坏性更改的行动手册。

实施变更

我们将使用 AWS CloudFormation 来定义我们的“基础设施即代码”，特别是在发生故障时加快重建速度。

软件更新会使用运行手册手动执行，安装和重新启动服务需要停机。如果在部署过程中出现问题，运行手册会介绍如何回滚到早期版本。

运维和开发团队将使用日志分析来更正全部错误，并且在确定修复工作的优先级并完成修复工作以后才部署更正。

备份数据

我们将使用供应商或专用的备份解决方案，按照运行手册，将加密备份数据发送给 Amazon S3。我们将按照运行手册，通过定期还原数据并确保能使用它们，测试备份数据能正常使用。我们会在 Amazon S3 对象上启用版本控制，并取消备份删除权限。我们会根据要求，使用 Amazon S3 存储桶生命周期策略，进行存档或永久删除操作。

构建弹性

工作负载会部署为一个区域和一个可用区。我们会在单个实例中部署应用程序，包括数据库。

测试弹性

新软件的部署管道已安排好，并进行了一些单元测试，主要是组装工作负载的白箱/黑箱测试。

灾难恢复 (DR) 计划

如果发生了故障，我们会等待故障结束，选择通过运行手册，使用 DNS 修改将请求路由到静态网站。具体的恢复时间取决于可部署基础设施的速度以及数据库还原到最近备份的速度。按照运行手册，如果某个可用区发生了故障，可在同一可用区或其他可用区中实施部署。

可用性设计目标

我们用 30 分钟了解并决定执行恢复，用 10 分钟在 AWS CloudFormation 中部署整个堆栈，假设我们部署到一个新的可用区，并假设数据库可以在 30 分钟内还原。这就意味着要从故障中恢复，大约需要 70 分钟的时间。假设每季度发生一次故障，预计全年的受影响时间为 280 分钟，也就是 4 小时 40 分钟。

这意味着可用性上限是 99.9%。实际可用性还取决于实际故障率、故障持续时间以及每次故障的实际恢复速度。在这种架构中，应用程序需要离线才能更新（每年预计 24 小时：每年六次更改，每次四小时），还要考虑实际事件。因此，参考白皮书前面的应用程序可用性表，我们看到 可用性设计目标 是 99%。

总结

主题	实施
监控资源	仅站点运行状况检查；无提醒。
适应需求的变化	通过重新部署以进行垂直扩展。

主题	实施
实施变更	用于部署与回滚的运行手册。
备份数据	用于备份与还原的运行手册。
构建弹性	完整重建；从备份恢复。
测试弹性	完整重建；从备份恢复。
灾难恢复 (DR) 计划	备份加密，在必要时还原至不同的可用区。

3 个 9 (99.9%) 场景

下一个可用性目标针对务必要具备高可用性的应用程序，但是这些应用程序可以承受短时间的不可用。这种类型的工作负载通常用于内部操作，而且此类操作在发生故障时会对员工造成影响。这种类型的工作负载也可能是面向客户的，它们不会带来很高的业务收入，并且可以承受较长的恢复时间或恢复点。这类工作负载包括账户或信息管理的管理应用程序。

我们可以使用两个可用区进行部署，并将应用程序分到不同的层中，从而改进工作负载的可用性。

监控资源

通过在主页上检查 HTTP 200 OK 状态，监控可以扩展为对网站上的可用性发出提醒。此外，每次更换 Web 服务器时、数据库故障转移时，系统都会发出提醒。我们还将监控 Amazon S3 上的静态内容以了解可用性，并在其不可用时发出提醒。日志记录将被汇总，以便于管理并帮助进行根本原因分析。

适应需求的变化

配置自动扩展以监控 EC2 实例上的 CPU 利用率，通过添加或移除实例将 CPU 目标维持在 70%，而且每个可用区内有不少于一个的 EC2 实例。若 RDS 实例上的负载模式显示需要进行扩展，我们将在维护时段更改实例类型。

实施变更

基础设施部署技术与之前的场景相同。

按每两到四周一次的固定日程安排交付新软件。软件更新将自动完成，不是使用 Canary 部署或蓝/绿部署模式，而是使用在位替换。回滚决策将使用运行手册做出。

我们将使用行动手册来找出问题的根本原因。在确定根本原因之后，运营和开发团队会共同确定错误修正方案，并在开发出解决方案后进行部署。

备份数据

备份和还原操作可以通过使用 Amazon RDS 完成。它将使用运行手册定期执行，以确保我们可以满足恢复要求。

构建弹性

我们可以使用两个可用区进行部署，并将应用程序分到不同的层中，从而改进应用程序的可用性。我们将使用跨多个可用区工作的服务，如 Elastic Load Balancing、Auto Scaling 和 Amazon RDS 多可用区部署，并通过 AWS Key Management Service 实现加密存储。这将确保在资源级别和可用区级别上能够容忍故障。

负载均衡器只会将流量路由到正常运行的应用程序实例。运行状况检查需要在数据平面/应用程序层上进行，以表明应用程序在实例上的容量。此检查不应针对控制平面。系统将提供 Web 应用程序运行状况检查 URL，并配置为可供负载均衡器和 Auto Scaling 使用，以便能够删除和替换发生故障的实例。如果实例在主

可用区内发生故障，Amazon RDS 将管理活动数据库引擎，使其在第二个可用区可用，然后执行修复以还原到相同弹性。

在分层之后，我们可以使用分布式系统弹性模式提高应用程序的可靠性，甚至当数据库在可用区故障转移过程中暂时不可用时，让应用程序仍然可用。

测试弹性

与之前的场景一样，我们会执行功能测试。我们不会测试 ELB、自动扩展或 RDS 故障转移的自我修复功能。

.我们将提供行动手册，其中包含针对常见数据库问题、安全相关事件，以及部署失败的相关解决方案。

灾难恢复 (DR) 计划

整个工作负载恢复和常用报告都有运行手册。恢复会使用备份，而且这些备份被存储在与工作负载相同的区域。

可用性设计目标

假设有一些故障必须要人工决定执行恢复。但为了尽可能提高这种场景下的自动化程度，我们假设每年只有两个事件需要做这种决定。我们需要 30 分钟决定执行恢复，并在 30 分钟内完成恢复。这意味着从故障中恢复需要 60 分钟。假设一年发生两次故障，预计每年受影响的时间为 120 分钟，

这意味着可用性上限是 99.95%。实际可用性还取决于实际故障率、故障持续时间以及每次故障的实际恢复速度。对于此架构，我们需要应用程序暂时离线以执行更新，但这些更新自动进行。我们估计每年需要为此花费 150 分钟：每次更改 15 分钟，每年 10 次。服务不可用的时间是每年总计 270 分钟，所以我们的 可用性设计目标 是 99.9%。

总结

主题	实施
监控资源	仅站点运行状况检查；在发生故障时发出提醒。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；调整多可用区 RDS 的大小。
实施变更	自动部署到位和用于回滚的运行手册。
备份数据	通过 RDS 自动化备份，以满足 RPO 和用于恢复的运行手册的要求。
构建弹性	自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区。
测试弹性	ELB 和应用程序会自我修复；RDS 是多可用区；无显式测试。
灾难恢复 (DR) 计划	通过 RDS 将备份加密存储于相同的 AWS 区域。

4 个 9 (99.99%) 场景

这个应用程序可用性目标需要应用程序具有较高的可用性并且可以承受组件故障。应用程序必须能够承受故障，而无需购买更多资源。此可用性目标针对的是任务关键型应用程序，这些应用程序是电子商务网站、企业对企业 Web 服务或高流量的内容/媒体站点等此类企业主要或重要的收入推动因素。

我们可以使用区域内静态稳定 的架构 进一步提高可用性。此可用性目标不需要控制平面改变工作负载的行为来承受故障。例如，应该会有足够的容量来承受一个可用区的损失。我们不要求更新到 Amazon Route 53 DNS。我们不需要创建任何新的基础设施，无论它是创建或修改 S3 存储桶、创建新的 IAM 策略（或修改策略），还是修改 Amazon ECS 任务配置。

监控资源

监控内容包括成功指标以及发生问题时的提醒。此外，每次更换出现故障的 Web 服务器时、数据库故障转移时以及可用区出现故障时，系统都会发出提醒。

适应需求的变化

我们将使用 Amazon Aurora 作为我们的 RDS，它让只读副本可以进行自动扩展。对于这些应用程序，主要内容的读取可用性优于写入可用性的设计也是一个关键的架构决策。Aurora 还可以视需要自动增加存储，以 10 GB 为单位，最高可达 64 TB。

实施变更

我们将使用 Canary 部署或蓝绿部署，将更新独立部署到每个隔离区中。部署是完全自动化的，包括在 KPI 表明存在问题时的回滚。

运行手册中应提出严格的报告要求和性能跟踪。如果成功运营趋向于无法实现性能或可用性目标，则将使用行动手册来确定导致这一趋势的原因。行动手册可用于确定未发现的故障模式和安全事件，还可用于确定发生故障的根本原因。我们还将与 AWS Support 一起提供基础设施事件管理产品。

负责构建与运营网站的团队需要确定任何意外故障的错误更正，并确定实施修复程序后进行部署的优先级。

备份数据

备份和还原操作可以通过使用 Amazon RDS 完成。它将使用运行手册定期执行，以确保我们可以满足恢复要求。

构建弹性

我们建议针对此方法使用三个可用区。使用三可用区部署，每个可用区最多拥有 50% 的静态容量。也可以使用两个可用区，但静态稳定容量的成本会更高，因为两个区域都必须拥有 100% 的峰值容量。我们将添加 Amazon CloudFront 以提供地理缓存，以及减少应用程序数据平面上的请求。

我们将使用 Amazon Aurora 作为我们的 RDS，并在所有三个区域内部署只读副本。

应用程序将在所有层上使用软件/应用程序弹性模式进行构建。

测试弹性

部署管道具有完整的测试套件，包括性能、负载和故障注入测试。

我们将在实际试用期间不断训练我们的故障恢复程序，使用运行手册确保我们可以执行任务而不会偏离程序。构建网站的团队也负责运营该网站。

灾难恢复 (DR) 计划

整个工作负载恢复和常用报告都有运行手册。恢复会使用备份，而且这些备份被存储在与工作负载相同的区域。Game Day 期间会定期演练还原程序。

可用性设计目标

假设有一些故障必须要手动决定执行恢复，尽管有更好的自动化选项。假定每年只有两个事件需要做这种决定，那么恢复操作会很快。我们需要 10 分钟决定执行恢复，并在五分钟内完成恢复。这意味着故障恢复时间为 15 分钟。假设一年发生两次故障，预计每年受影响的时间为 30 分钟，

这意味着可用性上限是 99.99%。实际可用性还将取决于实际故障率、故障持续时间以及每个因素的实际恢复速度。对于这种架构，我们假设应用程序通过更新持续在线。基于此，我们的 可用性设计目标 是 99.99%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在多个区域为 Aurora RDS 自动扩展存储与只读副本。
实施变更	自动 Canary 或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚。隔离区域会执行部署。
备份数据	通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。
构建弹性	为应用程序实施故障隔离区；自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程。
灾难恢复 (DR) 计划	通过 RDS 将备份加密存储于相同的 AWS 区域，并在实际试用期间演练。

多区域场景

在多个 AWS 区域中实施应用程序将增加运营成本，部分原因是我们需要隔离区域以保持其自主权。采用这种方法是一项深思熟虑的决定。也就是说，区域提供了强大的隔离边界，我们要竭尽全力避免跨区域的相关故障。在区域性 AWS 服务发生硬件依赖性故障时，使用多个区域可以更好地控制恢复时间。在本部分中，我们将讨论各种实施模式及其常规可用性。

主题

3½ 个 9 (99.95%)，故障恢复时间介于 5 到 30 分钟

应用程序的这种可用性目标要求停机时间极短，且特定时间内的数据丢失极少。具有此可用性目标的应用程序涉及以下领域：银行、投资、紧急服务和数据捕获。这些应用程序要求非常短的恢复时间和恢复点。

通过跨两个 AWS 区域使用 热备用 方法，我们可以进一步缩短恢复时间。我们将整个工作负载同时部署到两个区域，缩减我们的被动站点并且使所有数据保持最终一致性。两个部署在其对应的区域内均 静态稳定 。应用程序应采用分布式系统弹性模式进行构建。我们将需要创建轻量级 路由 组件监控工作负载的运行状况，并且可在必要时配置为将流量路由到被动区域。

监控资源

每次更换 Web 服务器时、数据库故障转移时以及区域出现故障时，系统都会发出提醒。我们还将监控 Amazon S3 上的静态内容以了解可用性，并在其不可用时发出提醒。日志记录将被汇总，以便于管理并帮助在每个区域进行根本原因分析。

路由组件会监控我们的应用程序运行状况，以及我们所拥有的任何区域硬依赖关系。

适应需求的变化

与 4 个 9 场景相同。

实施变更

按每两到四周一次的固定日程安排交付新软件。软件更新将使用金丝雀部署或蓝/绿部署模式自动完成。

发生区域故障转移时、这些事件期间发生常见客户问题时，以及需要常规报告时，可以参考运行手册。

我们将提供行动手册，其中提供了常见数据库问题、安全相关事件、部署失败、区域故障转移的意外客户问题，以及确定问题根本原因的相关解决方案。在确定根本原因之后，运营和开发团队会共同确定错误修正方案，并在开发出解决方案后进行部署。

我们还将与 AWS Support 一起提供基础设施事件管理。

备份数据

类似于 4 个 9 场景，我们会自动化 RDS 备份并使用 S3 版本控制。数据会自动并异步从主动区域的 Aurora RDS 集群被复制到被动区域的跨区域只读副本。S3 跨区域副本被用于自动并异步将数据从主动区域移动到被动区域。

构建弹性

与 4 个 9 场景相同，而且区域故障转移也是有可能的。它采用手动管理。在故障转移期间，我们将使用 DNS 故障转移将请求路由到静态网站，直到在第二个区域中恢复。

测试弹性

与 4 个 9 场景相同，我们将使用运行手册，并且通过实际试用验证架构。另外，对于立即实施与部署，RCA 更正优先于功能发布

灾难恢复 (DR) 计划

手动管理区域故障转移。所有数据都会被异步复制。扩展 热备用 内的基础设施。可通过在 AWS Step Functions 上执行的工作流对其进行自动化。AWS Systems Manager (SSM) 也可在您创建 SSM 文档更新 Auto Scaling 组并调整实例大小时帮助其自动化。

可用性设计目标

假设有一些故障必须要手动决定执行恢复，尽管有自动化选项。假定每年只有两个事件需要做这种决定，我们需要 20 分钟决定执行恢复，并在 10 分钟内完成恢复。这就意味着要从故障中恢复，大约需要 30 分钟的时间。假设一年发生两次故障，预计每年受影响的时间为 60 分钟，

这意味着可用性上限是 99.95%。实际可用性还将取决于实际故障率、故障持续时间以及每个因素的实际恢复速度。对于这种架构，我们假设应用程序通过更新持续在线。基于此，我们的 可用性设计目标 为 99.95%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查，包括 AWS 区域级别的 DNS 运行状况；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在主动和被动区域内的多个区域为 Aurora RDS 自动扩展存储与只读副本。在 AWS 区域之间同步数据和基础设施以获得静态稳定性。
实施变更	自动 Canary 或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚，每次在一个 AWS 区域内部署到一个隔离区域。
备份数据	在每个 AWS 区域内通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。Aurora RDS 和 S3 数据会从主动区域被自动并异步复制到被动区域。
构建弹性	自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区；在故障转移时，若出现静态站点，则手动管理区域故障转移。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程，以及用于传达问题出在哪里和如何加以更正或预防的通信路径。对于立即实施与部署，RCA 更正优先于功能发布。
灾难恢复 (DR) 计划	将热备用部署到其他区域。使用通过 AWS Step Functions 或 AWS Systems Manager 文档执行的工作流扩展基础设施。通过 RDS 加密备份。两个 AWS 区域之间的跨区域只读副本。在 Amazon S3 中跨区域复制静态资产。还原到最新的主动 AWS 区域，在实际试用期间演练，并与 AWS 协调。

5 个 9 (99.999%) 或更高的场景，恢复时间不到 1 分钟

应用程序的这种可用性目标要求几乎无停机时间，且特定时间内几乎没有数据丢失。具有此可用性目标的应用程序包括，例如某些银行、投资、金融、政府和关键业务应用程序，它们是极其庞大的创收公司的核心业务。其目标是在所有层实现强一致的数据存储和完全冗余。我们选择了一个基于 SQL 的数据存储。但在某些情况下，我们很难实现非常小的 RPO。如果您可以对数据进行分区，就可能不会出现数据丢失。这可能需要您添加应用程序逻辑和延迟，以确保在多个地理位置之间拥有一致的数据，以及在分区之间移动或复制数据的功能。如果使用 NoSQL 数据库，执行此分区可能会更容易。

我们可以跨多个 AWS 区域使用 主动-主动 或者 多主方法 来进一步提高可用性。工作负载将部署到全部所需的跨区域 静态稳定 的区域（因此其余的区域可以在丢失一个区域时处理负载）。A 路由 层将流量传送至运行状况良好的地理位置，并在这些位置的运行状况不佳时自动更改目标，同时临时停止数据复制层。Amazon Route 53 可提供间隔 10 秒的运行运行状况检查，还可以在您的记录集上提供低至 1 秒的 TTL。

监控资源

与 3½ 个 9 场景相同，而且将在检测到区域运行不正常时发出提醒，流量会被路由移出该区域。

适应需求的变化

与 3½ 个 9 场景相同。

实施变更

部署管道具有完整的测试套件，包括性能、负载和故障注入测试。我们将使用金丝雀部署或蓝/绿部署将更新部署到隔离区域中，一次部署到一个区域，部署完成后再从另一个区域开始。在部署期间，旧版本仍将在实例上继续运行以便更快地回滚。这些是完全自动化的，包括在 KPI 表明存在问题时的回滚。监控内容包括成功指标以及发生问题时的提醒。

运行手册中应提出严格的报告要求和性能跟踪。如果成功运营趋向于无法实现性能或可用性目标，则将使用行动手册来确定导致这一趋势的原因。行动手册可用于确定未发现的故障模式和安全事件，还可用于确定发生故障的根本原因。

构建网站的团队也负责运营该网站。团队需要确定任何意外故障的错误更正，并确定实施修复程序后进行部署的优先级。我们还将与 AWS Support 一起提供基础设施事件管理。

备份数据

与 3½ 个 9 场景相同。

构建弹性

应用程序应使用软件/应用程序弹性模式进行构建。实现所需的可用性可能还需要许多其他路由层。不要低估这种额外实施的复杂性。该应用程序将在部署故障隔离区域中实施，并进行分区和部署，这样一来，即使是区域级事件也不会影响所有客户。

测试弹性

我们将在实际试用期间不断验证架构，使用运行手册确保我们可以执行任务而不会偏离程序。

灾难恢复 (DR) 计划

主动-主动 多区域部署，将完整的工作负载基础设施和数据部署到多个区域。采用本地读取，全局写入策略，一个区域为所有写入的主数据库，而数据将被复制到其他区域以用于读取。若主 DB 区域发生故障，将需要提升一个新的 DB。本地读取，全局写入会把用户分配到主区域，并在该区域处理 DB 写入。这让用户能够从任何区域读取或写入，但需要复杂的逻辑以管理不同区域内的写入之间的潜在数据冲突。

当某区域被检测到运行不正常时，路由层会将流量自动路由到其余运行正常的区域。无需人工干预。

数据存储必须以可解决潜在冲突的方式在区域之间复制。由于存在延迟，您需要创建工具和自动化流程，以在各分区之间复制或移动数据，并平衡每个分区中的请求或数据量。您需要提供额外的运营手册，介绍数据冲突的补救措施。

可用性设计目标

假设我们进行了大量投资以自动执行所有恢复，且恢复可以在一分钟内完成。假设没有手动触发的恢复，但每季度最多有一次自动恢复操作，这意味着每年的恢复时间为四分钟。假设应用程序通过更新持续在线，基于此，我们的 可用性设计目标 是 99.999%。

总结

主题	实施
监控资源	对所有层和 KPI 执行运行状况检查，包括 AWS 区域级别的 DNS 运行状况；在配置的警报被触发时发出提醒；提醒发生故障。运营会议将密切探讨趋势，并设法达成设计目标。
适应需求的变化	适用于 Web 和自动扩展应用程序层的 ELB；在主动和被动区域内的多个区域为 Aurora RDS 自动扩展存储与只读副本。在 AWS 区域之间同步数据和基础设施以获得静态稳定性。
实施变更	自动 Canary 或蓝绿部署，并在 KPI 或提醒表明应用程序中有未检测到的问题时自动回滚，每次在一个 AWS 区域内部署到一个隔离区域。
备份数据	在每个 AWS 区域内通过 RDS 自动备份以满足 RPO 和在实际试用期间定期演练的自动还原要求。Aurora RDS 和 S3 数据会从主动区域被自动并异步复制到被动区域。
构建弹性	为应用程序实施故障隔离区；自动扩展以提供自我修复的 Web 和应用程序层；RDS 为多可用区；自动区域故障转移。
测试弹性	管道内有组件和隔离区域故障测试，并由运营人员在实际试用期间定期演练；存在行动手册以用于诊断未知问题；还有根本原因分析流程，以及用于传达问题出在哪里和如何加以更正或预防的通信路径。对于立即实施与部署，RCA 更正优先于功能发布。
灾难恢复 (DR) 计划	在至少两个区域内执行主动-主动部署。跨区域基础设施完全扩展并且静态稳定。跨区域对数据进行分区与同步。通过 RDS 加密备份。在 Game Day 期间演练区域故障，并与 AWS 协调。在还原期间，可能需要提升一个新的主数据库。

资源

文档

- Amazon Builders' Library - Amazon 如何构建与运营软件
- AWS 架构中心

实验室

- AWS 架构完善的可靠性实验室

外部链接

- 适应性队列模式： 大规模故障

- 计算总体系统可用性

图书

- Robert S. Hammer“适用于容错软件的模式”
- Andrew Tanenbaum 和 Marten van Steen“分布式系统：原则与范例”

总结

无论您是新手，刚开始接触可用性和可靠性主题，还是经验法丰富的老手，希望寻求见解以最大限度提高关键任务型工作负载的可用性，我们都希望本白皮书能够引发您的思考、提供新想法或引出新的质疑。我们希望它可以帮助您基于业务需求更深入地了解正确的可用性级别，以及如何设计可靠性加以实现。我们鼓励您利用本白皮书提供的设计、面向运营和面向恢复的建议，以及 AWS 解决方案架构师的知识和经验。非常期望收到您的来信，尤其是您在 AWS 上实现高可用性的成功案例。请联系您的客户团队，或使用 网站的“联系我们”。

贡献者

本文档的贡献者包括：

- Seth Eliot，Amazon Web Services“架构完善”的首席可靠性解决方案架构师
- Adrian Hornsby，Amazon Web Services 架构首席技术宣传官
- Philip Fitzsimons，Amazon Web Services Well-Architected 高级经理
- Rodney Lester，Amazon Web Services“架构完善”的可靠性主管
- Kevin Miller，Amazon Web Services 软件开发总监
- Shannon Richards，Amazon Web Services 技术项目经理

延伸阅读

如需更多信息，见：

- AWS 架构完善的框架

文档修订

要获得有关此白皮书更新的通知，请订阅 RSS 源。

update-history-change	update-history-description	update-history-date
次要更新 (p. 49)	已更新附录 A，增加了 AWS Global Accelerator 的可用性设计目标	July 24, 2020
新框架的更新 (p. 49)	大量更新和全新修订内容，包括：新增“工作负载架构”最佳实践章节，重新整理最佳实践，并将其纳入“变更管理和故障管理”章节，更新了资源，经过更新以包括最新的 AWS 资源和服务，如 AWS Global Accelerator、AWS Service Quotas 和 AWS Transit Gateway，新增/更新可靠性、可用性和弹性的定义，根据用于架构完善审查的 AWS Well-Architected Tool（问题和最佳实践）对白皮书进行调整，重新排列设计原则的顺序，将 自动从故障中恢复 移动到 测试恢复过程之前，更新等式的图表和格式，删除“关键服务”章节，并将关键 AWS 服务的参考整合到最佳实践当中。	July 8, 2020
次要更新 (p. 49)	修复错误的链接	October 1, 2019
已更新白皮书 (p. 49)	已更新附录 A	April 1, 2019
已更新白皮书 (p. 49)	新增特定的 AWS Direct Connect 联网推荐和更多服务设计目标	September 1, 2018
已更新白皮书 (p. 49)	新增“设计原则”和“限制管理”章节。更新链接，删除上游/下游术语的歧义，并为可用性场景中的其余“可靠性支柱”主题新增详细的参考。	June 1, 2018
已更新白皮书 (p. 49)	已将 DynamoDB 跨区域解决方案更改为 DynamoDB 全局表。新增服务设计目标	March 1, 2018
次要更新 (p. 49)	对可用性计算进行细微更正，以使其包括应用程序可用性	December 1, 2017
已更新白皮书 (p. 49)	经过更新以提供关于高可用性设计的指导，包括概念、最佳实践和示例实施。	November 1, 2017
原始版本 (p. 49)	已发布可靠性支柱：AWS 架构完善的框架	November 1, 2016

附录 A：旨在展示所选 AWS 服务的可用性

下面，我们提供了所选 AWS 服务旨在实现的可用性。这些值并不代表服务等级协议或我们的保证，仅用于提供对每种服务的设计目标的洞察。在某些情况下，我们会将可用性设计目标存在显着差异的部分服务区分开来。此列表并未涵盖所有 AWS 服务，我们会定期更新有关其他服务的信息。Amazon CloudFront、Amazon Route 53、AWS Global Accelerator 以及 the Identity and Access Management 控制平面提供了全局服务，并相应地说明了组件可用性目标。其他服务在 AWS 区域内提供服务，并相应地说明了可用性目标。许多服务在可用区内运行，与其他可用区内的服务分开。在这些情况下，如果使用了任意两个（或更多）可用区，我们会提供单个可用区的可用性设计目标。

Note

下表中的数字指的不是耐久性（长期保留数据）；它们是可用性编号（访问数据或功能）。

服务	组件	可用性设计目标
Amazon API Gateway	控制层面	99.950%
	数据层面	99.990%
Amazon Aurora	控制层面	99.950%
	单可用区数据平面	99.950%
	多可用区数据平面	99.990%
AWS CloudFormation	服务	99.950%
Amazon CloudFront	控制层面	99.900%
	数据平面（内容分发）	99.990%
Amazon CloudSearch	控制层面	99.950%
	数据层面	99.950%
Amazon CloudWatch	CW 指标（服务）	99.990%
	CW 事件（服务）	99.990%
	CW 日志（服务）	99.950%
AWS Database Migration Service	控制层面	99.900%
	数据层面	99.950%
AWS Data Pipeline	服务	99.990%
Amazon DynamoDB	服务（标准）	99.990%
	服务（全局表）	99.999%
Amazon EC2	控制层面	99.950%
	单可用区数据平面	99.950%
	多可用区数据平面	99.990%

服务	组件	可用性设计目标
Amazon ElastiCache	服务	99.990%
Amazon Elastic Block Store	控制层面	99.950%
	数据平面（卷可用性）	99.999%
Amazon Elasticsearch Service	控制层面	99.950%
	数据层面	99.950%
Amazon EMR	控制层面	99.950%
Amazon S3 Glacier	服务	99.900%
AWS Global Accelerator	控制层面	99.900%
	数据层面	99.995%
AWS Glue	服务	99.990%
Amazon Kinesis Data Streams	服务	99.990%
Amazon Kinesis Data Firehose	服务	99.900%
Amazon Kinesis Video Streams	服务	99.900%
Amazon Neptune	服务	99.900%
Amazon RDS	控制层面	99.950%
	单可用区数据平面	99.950%
	多可用区数据平面	99.990%
Amazon Rekognition	服务	99.980%
Amazon Redshift	控制层面	99.950%
	数据层面	99.950%
Amazon Route 53	控制层面	99.950%
	数据平面（查询解析度）	100.000%
Amazon SageMaker	数据平面（模型托管）	99.990%
	控制层面	99.950%
Amazon S3	服务（标准）	99.990%
AWS Auto Scaling	控制层面	99.900%
	数据层面	99.990%
AWS Batch	控制层面	99.900%
	数据层面	99.950%
AWS CloudHSM	控制层面	99.900%
	单可用区数据平面	99.900%

服务	组件	可用性设计目标
	多可用区数据平面	99.990%
AWS CloudTrail	控制平面（配置）	99.900%
	数据平面（数据事件）	99.990%
	数据平面（管理事件）	99.999%
AWS Config	服务	99.950%
AWS Direct Connect	控制层面	99.900%
	单个位置数据平面	99.900%
	多个位置数据平面	99.990%
Amazon Elastic File System	控制层面	99.950%
	数据层面	99.990%
AWS Identity and Access Management	控制层面	99.900%
	数据平面（身份验证）	99.995%
AWS IoT Core	服务	99.900%
AWS IoT Device Management	服务	99.900%
AWS IoT Greengrass	服务	99.900%
AWS Lambda	函数调用	99.950%
AWS Secrets Manager	服务	99.900%
AWS Shield	控制层面	99.500%
	数据平面（检测）	99.000%
	数据平面（缓解）	99.900%
AWS Storage Gateway	控制层面	99.950%
	数据层面	99.950%
AWS X-Ray	数据平面（控制台）	99.900%
	数据层面	99.950%
EC2 容器服务	控制层面	99.900%
	EC2 Container Registry	99.990%
	EC2 容器服务	99.990%
Elastic Load Balancing	控制层面	99.950%
	数据层面	99.990%
AWS Key Management Service (AWS KMS)	控制层面	99.990%

服务	组件	可用性设计目标
	数据层面	99.995%