数据库分解开启 AWS

AWS 规范性指导

AWS 规范性指导: 数据库分解开启 AWS

Table of Contents

数据库分解开启 AWS

Philippe Wanner 和 Saurabh Sharma，Amazon Web Services

2025 年 10 月（文档历史记录）

对于想要提高数据管理系统的灵活性、可扩展性和性能的组织来说，数据库现代化，尤其是整体数据库的分解，是一个关键的工作流程。随着企业的发展及其数据需求变得越来越复杂，传统的单体数据库往往难以跟上步伐。这会导致性能瓶颈、维护难题以及难以适应不断变化的业务需求。

以下是单体数据库的常见难题：

- 业务领域错位 — 单体数据库通常无法将技术与不同的业务领域保持一致，这可能会限制组织增长。
- 可扩展性限制 — 系统经常达到扩展极限，这给业务扩展造成了障碍。
- 建筑刚性 — 紧密耦合的结构使得很难在不影响整个系统的情况下更新特定组件。
- 性能降低 — 数据负载的增加和用户并发性的增加通常会导致系统性能下降。

以下是数据库分解的好处：

- 增强业务灵活性 — 分解可以快速适应不断变化的业务需求，并支持独立扩展。
- 优化性能 — 分解可帮助您创建专门的数据库解决方案，这些解决方案针对特定用例量身定制，并且可以独立扩展每个数据库。
- 改善成本管理 — 分解可以提高资源利用率并降低运营成本。
- 灵活的许可选项 — 分解为从昂贵的专有许可证过渡到开源替代品创造了机会。
- 创新支持 — 分解有助于针对特定工作负载采用专门构建的数据库。

目标受众

本指南可帮助数据库架构师、云解决方案架构师、应用程序开发团队和企业架构师。它旨在帮助您将整体数据库分解为与微服务对齐的数据存储，实施域驱动的数据库架构，规划数据库迁移策略，并扩展数据库运营以满足不断增长的业务需求。要理解本指南中的概念和建议，您应该熟悉关系数据库和 NoSQL 数据库原理、 AWS 托管数据库服务和微服务架构模式。本指南旨在帮助处于数据库分解项目初始阶段的组织。

目标

本指南可以帮助您的组织实现以下目标：

- 收集分解目标架构的要求。
- 制定评估风险和沟通的系统方法。
- 制定分解计划。
- 定义成功指标、关键绩效指标 (KPIs)、缓解策略和业务连续性计划。
- 建立更好的工作负载弹性，帮助您满足业务需求。
- 了解如何针对特定用例采用专门的数据库，从而实现创新。
- 加强组织的数据安全和治理。
- 通过以下方式降低成本：
  - 降低许可费
  - 减少供应商锁定
  - 改善获得更广泛的社区支持和创新的机会
  - 能够为不同的组件选择不同的数据库技术
  - 逐步迁移，可降低风险并随着时间的推移分摊成本
  - 提高资源利用率

数据库分解的常见挑战和管理责任

数据库分解是一个复杂的过程，需要仔细的规划、执行和管理。在组织寻求实现数据基础架构现代化的过程中，他们经常会遇到无数挑战，这些挑战可能会影响其项目的成功。本节描述了常见的障碍，并介绍了克服这些障碍的结构化方法。

常见挑战

数据库分解项目面临着技术、人员和业务方面的多项挑战。在技术方面，确保分布式系统的数据一致性是一个重大障碍。在过渡期间，它还可能对性能和稳定性产生潜在影响，因此您必须与现有系统无缝集成。与人相关的挑战包括与新系统相关的学习曲线、员工对变革的潜在抵制以及必要资源的可用性。从业务角度来看，项目必须应对时间超支、预算限制以及迁移过程中可能出现的业务中断等风险。

定义角色和职责

鉴于这些跨越技术、人员和业务层面的复杂挑战，确定明确的角色和责任对于项目成功至关重要。负责任、负责、咨询和知情 (RACI) 矩阵为应对这些挑战提供了必要的结构。它明确定义了谁做出决策、谁执行工作、谁提供意见以及谁需要在分解的每个阶段随时了解情况。这种清晰度有助于防止因决策模棱两可而导致的延误，鼓励利益相关者适当参与，并建立对关键交付成果的问责制。如果没有这样的框架，团队可能会为职责重叠、错过沟通和上报途径不明确而苦苦挣扎——这些问题可能会加剧现有的技术复杂性和变更管理挑战，同时增加时间和预算超支的风险。

以下示例 RACI 矩阵是一个起点，可以帮助您阐明组织中的潜在角色和职责。

任务或活动	项目经理	建筑师	开发者	利益相关者
确定业务成果和挑战	A/R	R	C	–
定义范围并确定需求	A	R	C	C/I
确定项目成功指标	A	R	C	我
制定并执行沟通计划	A/R	C	C	我

定义目标架构	我	A/R	C	–
控制数据库访问权限	我	A/R	R	–
制定和执行业务连续性计划	A/R	C	我	–
分析内聚力和耦合	我	A/R	R	我
将业务逻辑（例如存储过程）从数据库移动到应用程序层	我	A	R	–
解耦表关系，称为联接	我	A	R	–

定义数据库分解的范围和要求

在定义数据库分解项目的范围和确定要求时，必须从组织的需求向后推进。这需要采用系统化的方法，在技术可行性与业务价值之间取得平衡。这个初始步骤为整个过程奠定了基础，并帮助您确保项目的目标与组织的目标和能力保持一致。

本节包含以下主题：

建立核心分析框架

范围定义从一个系统的工作流程开始，该工作流程引导分析完成四个相互关联的阶段。这种全面的方法可确保数据库分解工作建立在对现有系统和操作要求的透彻了解的基础上。以下是核心分析框架的各个阶段：

1. 参与者分析-彻底识别与数据库交互的所有系统和应用程序。这包括映射执行写入操作的生产者和处理读取操作的使用者，同时记录他们的访问模式、频率和峰值使用时间。这种以客户为中心的视图可帮助您了解任何变更的影响，并确定在分解过程中需要特别注意的关键路径。
2. 活动分析 — 深入研究每个参与者执行的具体操作。您可以为每个系统创建详细的创建、读取、更新和删除 (CRUD) 矩阵，并确定它们访问哪些表以及如何访问这些表。此分析可帮助您发现分解的自然边界，并突出显示可以简化当前架构的区域。
3. 依赖关系映射 — 记录系统之间的直接和间接依赖关系，创建数据流和关系的清晰可视化。这有助于确定潜在的突破点和需要仔细规划以赢得信任的领域。该分析既考虑了技术依赖关系，例如共享表和外键，也考虑了业务流程依赖关系，例如工作流序列和报告要求。
4. 一致性要求 — 按照高标准检查每项操作的一致性需求。确定哪些操作需要即时一致性，例如财务交易。其他操作可以以最终一致性运行，例如分析更新。这种分析直接影响整个项目中分解模式的选择和架构决策。

定义数据库分解的系统边界

系统边界是逻辑边界，用于定义一个系统的终点和另一个系统的起点，包括数据所有权、访问模式和集成点。在定义系统边界时，要做出深思熟虑但果断的选择，在全面规划和实际实施需求之间取得平衡。将数据库视为可能跨越多个物理数据库或架构的逻辑单元。此边界定义实现了以下关键目标：

- 识别所有外部参与者及其互动模式
- 全面映射入站和出站依赖关系
- 记录技术和操作限制
- 清楚地描述了分解工作的范围

考虑发布周期

了解发布周期对于规划数据库分解至关重要。查看目标系统和任何相关系统的续订时间。确定协调变革的机会。考虑任何计划中的联网系统停用，因为这可能会影响您的分解策略。将现有的变更窗口和部署限制考虑在内，以最大限度地减少业务中断。确保您的实施计划与所有互联系统的发布时间表保持一致。

评估数据库分解的技术限制

在继续进行数据库分解之前，请评估影响现代化方法的关键技术限制。检查您当前技术堆栈的能力，包括数据库版本、框架、性能要求和服务级别协议。考虑安全与合规要求，尤其是针对受监管行业。查看当前的数据量、增长预测和可用的迁移工具，为您的扩展决策提供依据。最后，确认您对源代码和系统修改的访问权限，因为这将决定可行的分解策略。

了解组织背景

成功的数据库分解需要您了解系统运行所处的更广泛的组织格局。绘制跨部门的依赖关系，并在团队之间建立清晰的沟通渠道。评估您团队的技术能力，并确定您需要解决的任何培训需求或技能差距。考虑变更管理的影响，包括如何管理过渡和保持业务连续性。评估可用资源和任何限制，例如预算或人员限制。最后，将分解策略与利益相关者的期望和优先事项保持一致，以促进整个项目的持续支持。

评估数据库分解的风险

全面的风险评估对于成功分解数据库至关重要。仔细评估风险，例如迁移过程中的数据完整性、潜在的系统性能下降、可能的集成失败以及安全漏洞。这些技术挑战必须与业务风险相平衡，包括潜在的运营

中断、资源限制、时间延迟和预算限制。针对每项已确定的风险，制定具体的缓解策略和应急计划，以便在保护业务运营的同时保持项目势头。

创建风险矩阵，评估潜在问题的影响和可能性。与技术团队和业务利益相关者合作，识别风险，设定明确的干预阈值，并制定具体的缓解策略。例如，将数据丢失风险评为影响大、概率低，这需要强大的备份策略。轻微的性能下降可能是中等影响和高概率，因此需要主动监控。

建立定期的风险审查周期，以重新评估优先事项，并随着项目的发展调整缓解计划。这种系统的方法可确保将资源集中在最关键的风险上，同时为新出现的问题保持清晰的上报路径。

定义数据库分解的成功标准

数据库分解的成功标准必须明确定义并在多个维度上进行衡量。从业务角度来看，为成本降低、改进 time-to-market、系统可用性和客户满意度设定具体目标。应通过系统性能、部署效率、数据一致性和整体可靠性方面的可量化改进来衡量技术成功。对于迁移过程，定义严格的要求，包括零数据丢失、可接受的业务中断限制、预算合规性和遵守时间表。

通过维护基线和目标指标、明确的测量方法和定期的审查时间表，全面记录这些标准。为每个成功指标分配明确的所有者，并映射不同指标之间的依赖关系。这种衡量成功的全面方法使技术成就与业务成果保持一致，同时在整个分解过程中保持问责制。

在分解过程中控制数据库访问权限

许多组织都面临着一个共同的情况：一个多年来一直有机增长并可供多个服务和团队直接访问的中央数据库。这会造成几个关键问题：

- 增长不受控制 — 随着团队不断添加新功能和修改架构，数据库变得越来越复杂且难以管理。
- 性能问题 — 即使进行了硬件改进，不断增长的负载最终仍有可能超出数据库的容量。由于架构复杂或缺乏技能，无法调整查询。无法预测或解释系统性能。
- 分解瘫痪 — 当多个团队正在积极修改数据库时，拆分或重构数据库几乎是不可能的。

> **Note**
>
> 单体数据库系统通常对应用程序或服务或管理重复使用相同的凭据。这会导致数据库可追溯性差。设置专用角色并采用最低权限原则可以帮助您提高安全性和可用性。

在处理已经变得笨重的单体数据库时，控制访问的最有效模式之一称为数据库包装器服务。它为管理复杂的数据库系统提供了战略性的第一步。它可以建立受控的数据库访问权限并实现逐步现代化，同时降低风险。这种方法通过提供对数据使用模式和依赖关系的清晰可见性，为渐进式改进奠定了基础。它是一种过渡架构，是向完全数据库分解迈出的一步。包装器服务提供了成功完成旅程所需的稳定性和控制力。

本节包含以下主题：

- 使用数据库包装器服务模式控制访问
- 使用 CQRS 模式控制访问权限

使用数据库包装器服务模式控制访问

封装服务是充当数据库外观的服务层。当您需要在为将来的分解做准备的同时维护现有功能时，这种方法特别有用。这种模式遵循一个简单的原则——当某些东西太混乱时，首先要控制混乱局面。包装服务成为访问数据库的唯一授权方式，它提供了一个受控的接口，同时隐藏了底层的复杂性。

当由于架构复杂而无法立即进行数据库分解时，或者当多个服务需要持续的数据访问时，请使用此模式。它在过渡期间特别有价值，因为它为谨慎的重构提供了时间，同时保持了系统的稳定性。在将数据所有权整合到特定团队或新应用程序需要跨多个表的聚合视图时，这种模式效果很好。

例如，在以下情况下应用此模式：

- 架构复杂性无法立即分离
- 多个团队需要持续的数据访问权限
- 最好采用渐进式现代化
- 团队重组需要明确的数据所有权
- 新应用程序需要整合的数据视图

数据库包装器服务模式的优点和局限性

以下是数据库封装模式的优点：

- 受控增长 — 包装器服务可防止进一步不受控制地向数据库架构添加内容。
- 明确界限 — 实施过程可帮助您建立明确的所有权和责任界限。
- 重构自由 — 包装服务允许您在不影响消费者的情况下进行内部更改。
- 提高了可观察性 — 包装器服务是用于监控和记录的单点。
- 简化测试 — 包装服务使消费服务可以更轻松地创建用于测试的简化模拟版本。

以下是数据库包装模式的局限性。

- 技术耦合 — 当包装服务使用与消费服务相同的技术堆栈时，其效果最好。
- 初始开销 — 包装服务需要额外的基础架构，这可能会影响性能。
- 迁移工作 — 要实施包装器服务，您必须跨团队进行协调，以摆脱直接访问。
- 性能-如果打包服务遇到高流量、高使用量或频繁访问，则使用服务的性能可能会很差。在数据库之上，包装器服务必须处理分页、游标和数据库连接。根据您的用例，它可能无法很好地扩展，并且可能不适合提取、转换和加载 (ETL) 工作负载。

实现数据库包装器服务模式

实现数据库包装器服务模式分为两个阶段。首先，创建数据库包装器服务。然后，您可以通过它指挥所有访问权限并记录访问模式。

第 1 阶段：创建数据库包装器服务

创建一个充当数据库看门人的轻量级服务层。最初，它应该反映所有现有功能。此封装服务成为所有数据库操作的必备访问点，它将直接的数据库依赖关系转换为服务级别的依赖关系。在此层实施详细的日志记录和监控，以跟踪使用模式、性能指标和访问频率。保留现有的存储过程，但要确保只能通过这个新的服务接口访问它们。

第 2 阶段：实施访问控制

通过封装服务系统地重定向所有数据库访问权限，然后撤消直接访问数据库的外部系统的直接数据库权限。在迁移服务时记录每种访问模式和依赖关系。这种受控访问允许在不中断外部使用者的情况下对数据库组件进行内部重构。例如，从低风险的只读操作开始，而不是复杂的事务性工作流程。

第 3 阶段：监控数据库性能

使用包装器服务作为数据库性能的集中监控点。跟踪关键指标，包括查询响应时间、使用模式、错误率和资源利用率。为性能阈值和异常模式设置警报。例如，监控运行缓慢的查询、连接池利用率和事务吞吐量，以主动识别潜在问题。

使用此统一视图可通过查询调整、资源分配调整和使用模式分析来优化数据库性能。包装服务的集中化性质使得在保持一致的性能标准的同时，可以更轻松地实施改进并验证其对所有消费者的影响。

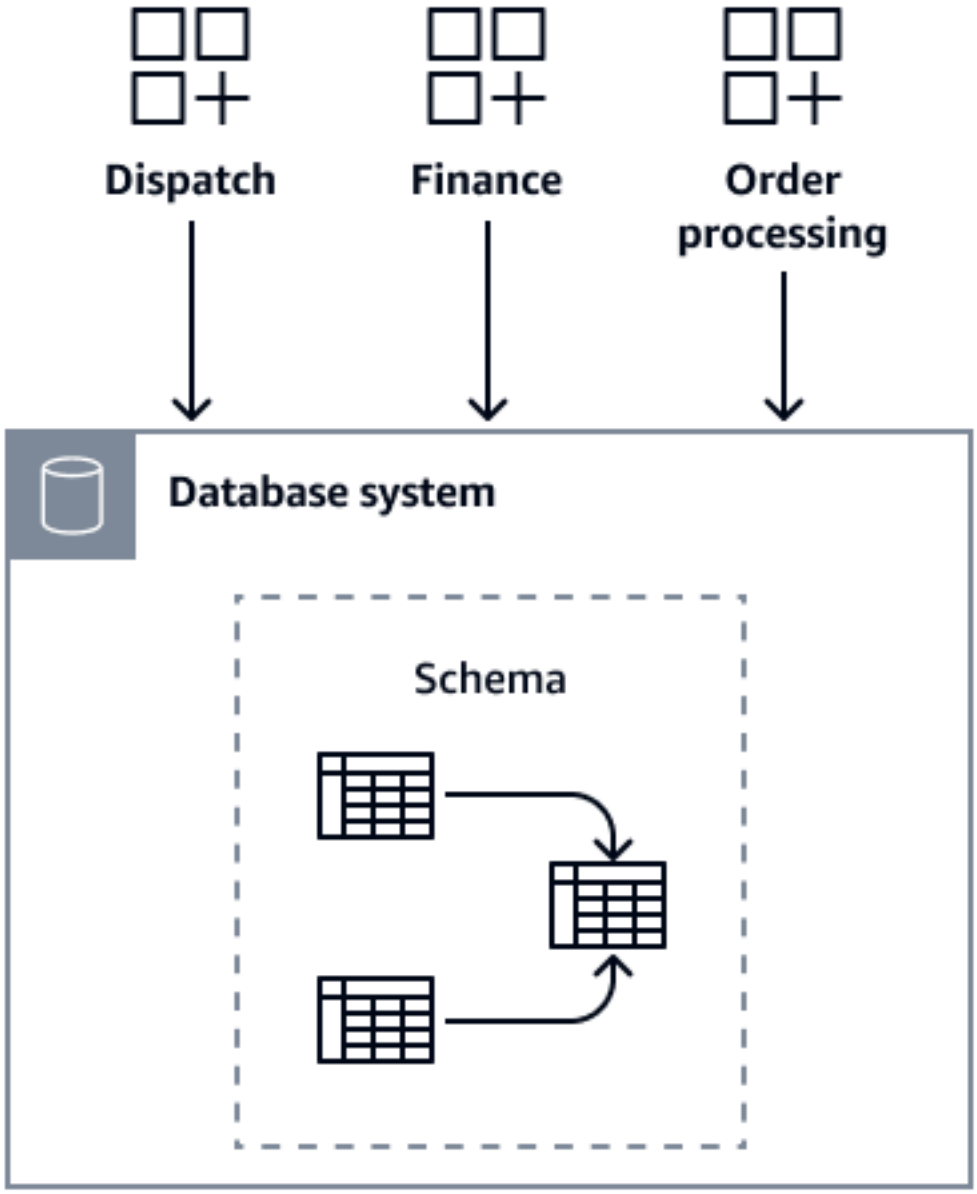
实现数据库包装器服务的最佳实践

以下最佳实践可以帮助您实现数据库包装器服务：

- 从小处着手 — 从简单代理现有功能的最小包装器开始
- 保持稳定性-在进行内部改进的同时保持服务接口稳定
- 监控使用情况-实施全面监控以了解访问模式
- 明确所有权 — 指派一个专门的团队来维护包装器和底层架构
- 鼓励本地存储 — 激励团队将数据存储在自己的数据库中

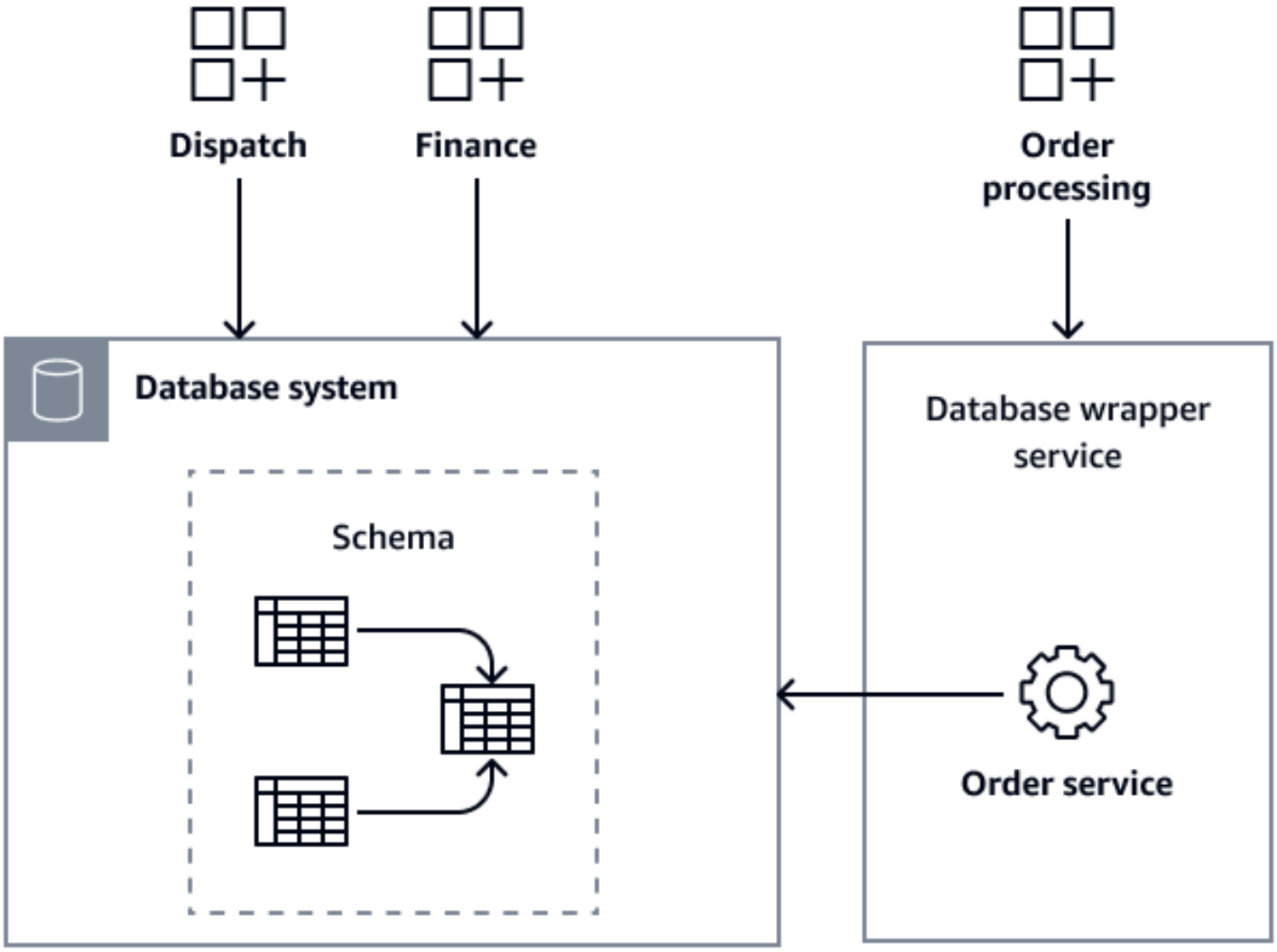
基于场景的示例

本节介绍了一个名为B AnyCompany ook s的虚构公司如何使用数据库包装模式来控制对其整体数据库系统的访问的示例。 AnyCompany Books提供三项关键服务：调度、财务和订单处理。这些服务共享对中央数据库的访问权限。每项服务都由不同的团队维护。随着时间的推移，他们会独立修改数据库架构以满足其特定需求。这导致了相互依存关系错综复杂的网络和日益复杂的数据库结构。



该公司的应用程序或企业架构师意识到需要分解这个单体数据库。他们的目标是为每项服务提供自己的专用数据库，以提高可维护性并减少跨团队的依赖性。但是，他们面临着重大挑战——在所有三个团队都继续为正在进行的项目积极修改数据库的同时，几乎不可能分解数据库。不断的架构变化和团队之间缺乏协调使得尝试任何重大重组都极具风险。

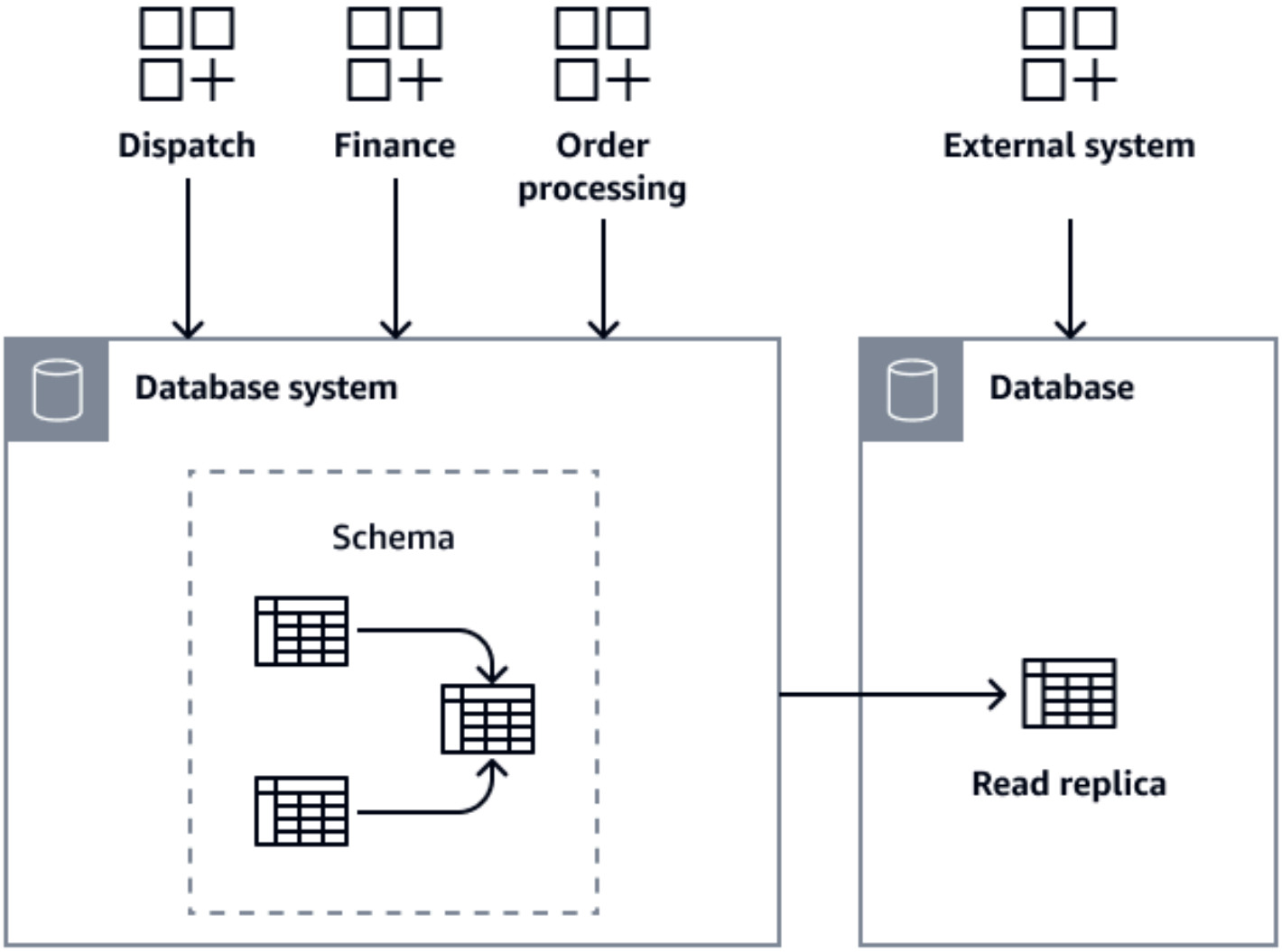
架构师使用数据库包装器服务模式开始控制对单体数据库的访问。首先，他们为一个名为 Order 服务的特定模块设置了数据库封装服务。然后，他们将订单处理服务重定向到访问包装服务，而不是直接访问数据库。下图显示了修改后的基础架构。



使用 CQRS 模式控制访问权限

可用于隔离连接到该中央数据库的外部系统的另一种模式是命令查询责任分离 (CQRS)。如果某些外部系统主要用于读取（例如分析、报告或其他读取密集型操作）连接到您的中央数据库，则可以创建单独的读取优化数据存储。

这种模式有效地将这些外部系统与数据库分解和架构更改的影响隔离开来。通过为特定查询模式维护专用的只读副本或专门构建的数据存储，团队可以继续运营，而不会受到主数据库结构变化的影响。例如，在分解整体数据库时，报告系统可以继续使用其现有的数据视图，分析工作负载可以通过专用的分析存储保持其当前的查询模式。这种方法提供了技术隔离并实现了组织自主权，因为不同的团队可以独立发展其系统，而无需与主数据库的转型过程紧密结合。



有关此模式的更多信息及其用于解耦表关系的示例，请参阅本指南的CQRS 模式后面部分。

分析数据库分解的内聚力和耦合

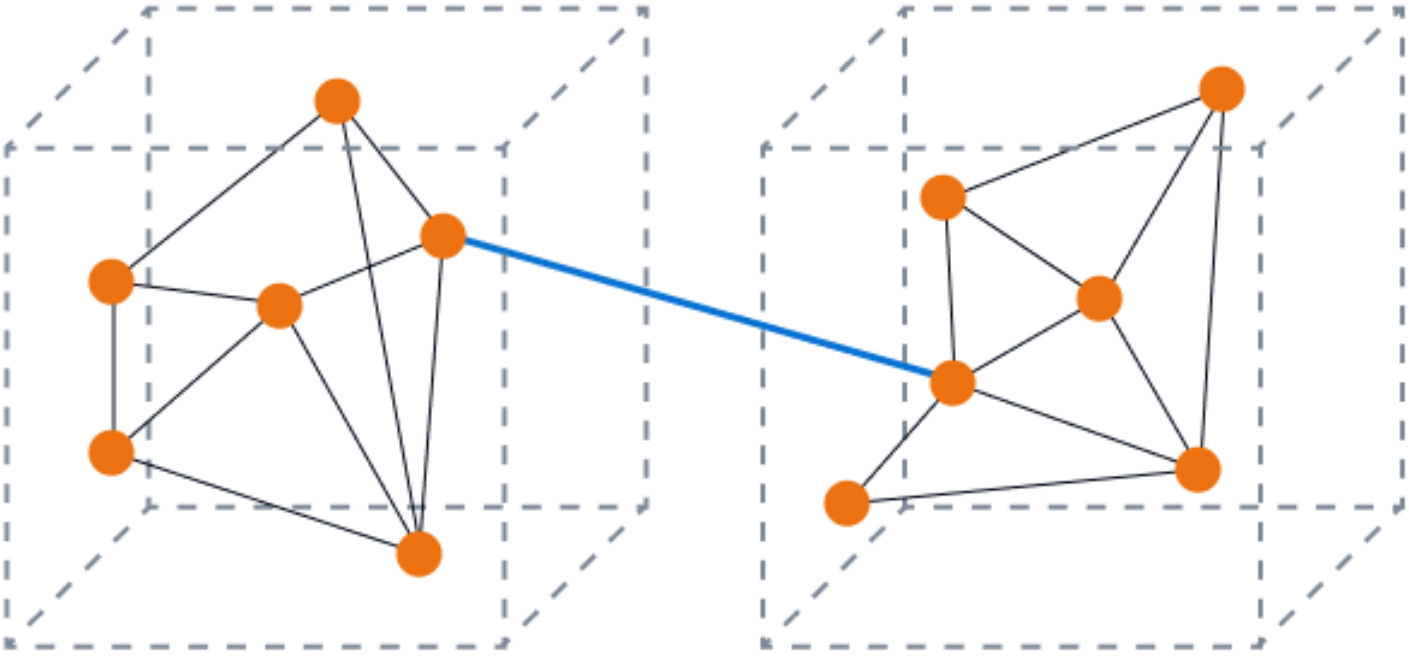
本节帮助您分析整体数据库中的耦合和内聚模式，以指导其分解。了解数据库组件如何相互交互和相互依赖对于识别自然断点、评估复杂性和规划分阶段迁移方法至关重要。该分析揭示了隐藏的依赖关系，突出显示了适合立即分离的区域，并帮助您确定分解工作的优先顺序，同时最大限度地降低转换风险。通过检查耦合和内聚力，您可以就组件分离顺序做出明智的决定，以便在整个转换过程中保持系统稳定性。

本节包含以下主题：

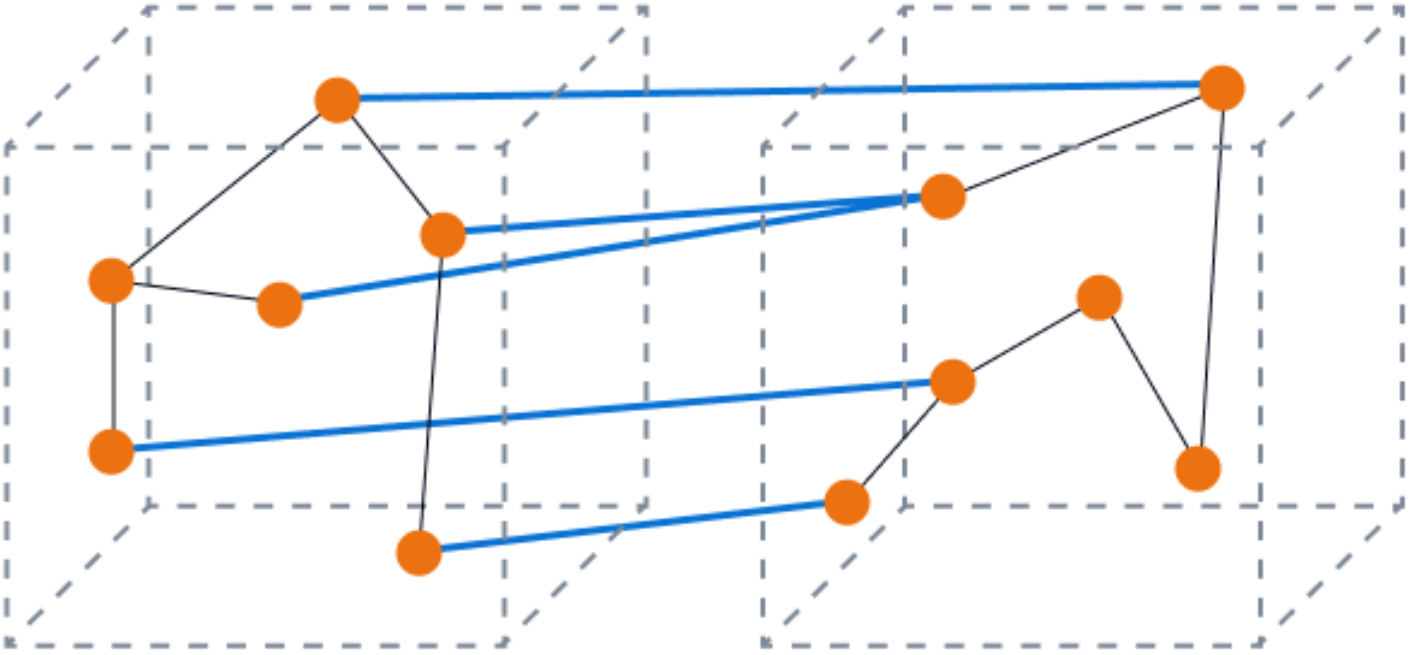
关于内聚力和耦合

耦合度量数据库组件之间的相互依赖程度。在精心设计的系统中，您希望实现松耦合，即对一个组件的更改对其他组件的影响最小。凝聚力衡量数据库组件中的元素如何协同工作以达到单一的、明确的目的。高内聚力表示组件的元素紧密相关，并且侧重于特定的功能。分解整体数据库时，必须同时分析各个组件内部的内聚力以及它们之间的耦合。此分析可帮助您就如何在保持系统完整性和性能的同时分解数据库做出明智的决定。

下图显示了具有高内聚力的松散耦合。数据库中的组件协同工作以执行特定的功能，您可以最大限度地减少更改对单个组件的影响。这是理想的状态。



下图显示了高耦合和低内聚力。数据库组件已断开连接，更改极有可能影响其他组件。



单片数据库中常见的耦合模式

将整体数据库分解为特定于微服务的数据库时，通常会发现几种耦合模式。了解这些模式对于成功的数据库现代化计划至关重要。本节介绍每种模式、其挑战以及减少耦合的最佳实践。

实现耦合模式

定义：组件在代码和架构级别紧密相连。例如，修改customer表的结构会影响orderinventory、和billing服务。

现代化的影响：每个微服务都需要自己的专用数据库架构和数据访问层。

挑战：

- 对共享表的更改会影响多项服务
- 出现意外副作用的风险很高
- 测试复杂性增加
- 难以修改单个组件

减少耦合的最佳实践：

- 定义组件之间的清晰接口
- 使用抽象层隐藏实现细节
- 实现特定于域的架构

时间耦合模式

定义：操作必须按特定顺序运行。例如，在库存更新完成之前，无法继续处理订单。

现代化的影响：每个微服务都需要自主的数据控制。

挑战：

- 打破服务之间的同步依赖关系
- 性能瓶颈
- 难以优化
- 并行处理有限

减少耦合的最佳实践：

- 尽可能实现异步处理

- 使用事件驱动架构
- 在适当时进行设计以实现最终一致性

部署耦合模式

定义：系统组件必须作为一个单元部署。例如，支付处理逻辑的微小更改需要重新部署整个数据库。

现代化影响：每项服务独立部署数据库

挑战：

- 高风险部署
- 部署频率有限
- 复杂的回滚程序

减少耦合的最佳实践：

- 分解为可独立部署的组件
- 实施数据库分片策略
- 使用蓝绿色部署模式

域耦合模式

定义：业务领域共享数据库结构和逻辑。例如，`customerorder`、和`inventory`域共享表和存储过程。

现代化的影响：特定领域的数据隔离

挑战：

- 复杂的域边界
- 难以扩展单个域
- 纠结的商业规则

减少耦合的最佳实践：

- 确定明确的域边界

- 按域上下文分隔数据
- 实施特定于域的服务

整体数据库中常见的内聚模式

在评估数据库组件的分解时，通常会发现几种内聚模式。了解这些模式对于识别结构良好的数据库组件至关重要。本节介绍每种模式、其特征以及增强凝聚力的最佳实践。

功能凝聚力模式

定义：所有元素都直接支持并有助于执行单一的、定义明确的功能。例如，付款处理模块中的所有存储过程和表格仅处理与支付相关的操作。

现代化的影响：微服务数据库设计的理想模式

挑战：

- 确定明确的功能边界
- 分离混合用途组件
- 保持单一责任

增强凝聚力的最佳实践：

- 将相关函数组合在一起
- 移除不相关的功能
- 定义清晰的组件边界

顺序凝聚力模式

定义：一个元素的输出变成另一个元素的输入。例如，订单输入到订单处理中的验证结果。

现代化的影响：需要仔细的工作流程分析和数据流映射

挑战：

- 管理步骤之间的依赖关系
- 处理故障场景

- 维护流程顺序

增强凝聚力的最佳实践：

- 记录清晰的数据流
- 实施正确的错误处理
- 在步骤之间设计清晰的接口

沟通凝聚力模式

定义：元素对相同的数据进行操作。例如，客户档案管理功能全部使用客户数据。

现代化的影响：帮助确定服务分离的数据边界，以减少模块之间的耦合

挑战：

- 确定数据所有权
- 管理共享数据访问权限
- 维护数据一致性

增强凝聚力的最佳实践：

- 定义明确的数据所有权
- 实施正确的数据访问模式
- 设计有效的数据分区

程序凝聚模式

定义：元素之所以组合在一起，是因为它们必须按特定的顺序执行，但它们在功能上可能不相关。例如，在订单处理中，处理订单验证和用户通知的存储过程被分组在一起，这仅仅是因为它们是按顺序进行的，尽管它们用于不同的目的并且可以由不同的服务来处理。

现代化的影响：需要在保持流程的同时仔细分离程序

挑战：

- 分解后保持正确的工艺流程

- 与程序依赖关系相比，识别真正的功能边界

增强凝聚力的最佳实践：

- 根据程序的功能目的而不是执行顺序来分开程序
- 使用编排模式来管理流程
- 为复杂序列实施工作流管理系统
- 设计事件驱动架构以独立处理流程步骤

时间凝聚力模式

定义：元素与时间要求相关。例如，下订单时，必须同时执行多项操作：库存检查、付款处理、订单确认和发货通知都必须在特定的时间窗口内进行，以保持一致的订单状态。

现代化影响：可能需要在分布式系统中进行特殊处理

挑战：

- 协调分布式服务之间的计时依赖关系
- 管理分布式事务
- 确认多个组件的流程完成

增强凝聚力的最佳实践：

- 实施适当的调度机制和超时
- 使用具有清晰序列处理的事件驱动架构
- 为最终与补偿模式保持一致而设计
- 为分布式事务实现传奇模式

合乎逻辑或巧合的内聚模式

定义：元素的逻辑分类是为了做同样的事情，即使它们之间的关系薄弱或没有有意义的关系。一个例子是将客户订单数据、仓库库存计数和营销电子邮件模板存储在同一个数据库架构中，因为尽管访问模式、生命周期管理和扩展要求不同，但它们都与销售运营有关。另一个例子是将订单支付处理和产品目录管理合并到同一个数据库组件中，因为它们都是电子商务系统的一部分，尽管它们具有不同的业务职能和不同的运营需求。

现代化的影响：应进行重构或重组

挑战：

- 确定更好的组织模式
- 打破不必要的依赖关系
- 重组任意分组的组件

增强凝聚力的最佳实践：

- 根据真正的职能界限和业务领域进行重组
- 删除基于肤浅关系的任意分组
- 根据业务能力对元素进行适当的分离
- 使数据库组件与其特定的操作要求保持一致

实现低耦合和高内聚力

最佳实践

以下最佳实践可以帮助您实现低耦合：

- 最大限度地减少数据库组件之间的依赖关系
- 使用定义明确的接口进行组件交互
- 避免共享状态和全局数据结构

以下最佳实践可以帮助您实现高凝聚力：

- 将相关数据和操作组合在一起
- 确保每个组成部分都有单一、明确的责任
- 在不同业务领域之间保持明确的界限

第 1 阶段：映射数据依赖关系

绘制数据关系并确定自然边界。您可以使用诸如SchemaSpy之类的工具通过在实体关系 (ER) 图中显示表来可视化数据库。这提供了对数据库的静态分析，并指出了数据库中一些明确的界限和依赖关系。

您也可以将数据库架构导出到图形数据库或Jupiter笔记本中。然后，您可以应用聚类或互连组件算法来识别自然边界和依赖关系。其他 AWS Partner 工具（例如）CAST Imaging可以帮助了解您的数据库依赖关系。

第 2 阶段：分析交易边界和访问模式

分析事务模式以保持原子性、一致性、隔离性、持久性 (ACID) 属性，并了解如何访问和修改数据。您可以使用数据库分析和诊断工具，例如Oracle Automatic Workload Repository (AWR)或PostgreSQL pg_stat_statements。此分析可帮助您了解谁在访问数据库以及事务边界是什么。它还可以帮助你了解运行时表之间的内聚和耦合。您还可以使用监控和分析工具，这些工具可以链接代码和数据库执行配置文件，例如Dynatrace AppEngine。

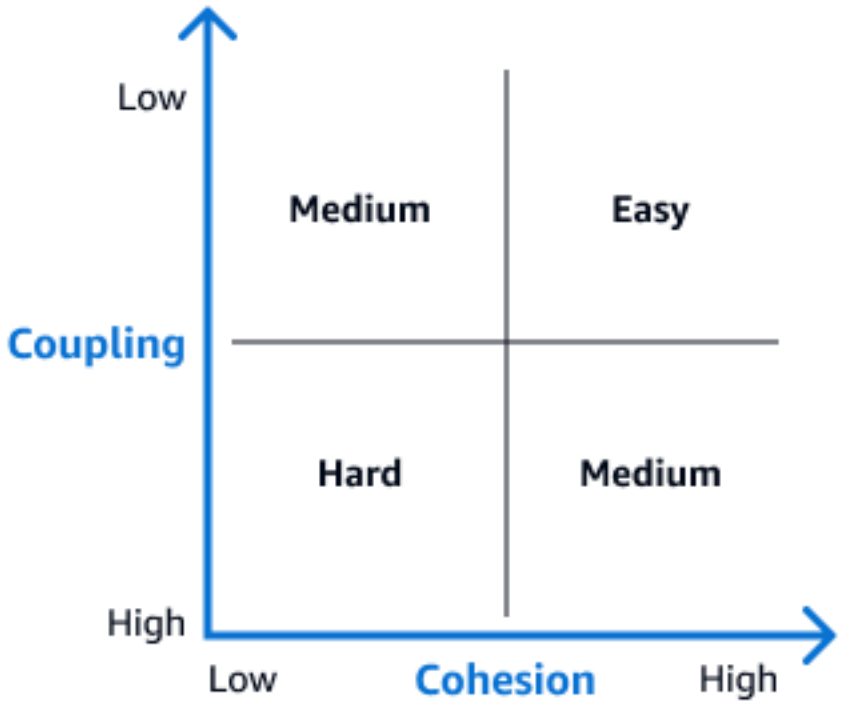
人工智能工具（例如 vFunction）可以通过分析应用程序的功能和域边界来帮助您识别域边界。尽管vFunction主要分析应用程序层，但其见解可以指导应用程序和数据库的分解，从而支持与业务领域保持一致。

第 3 阶段：识别独立表

寻找能说明两个关键特征的表格：

- 高内聚力 — 表格的内容彼此密切相关
- 低耦合 — 它们对其他表的依赖性最小。

以下耦合内聚矩阵可以帮助您确定解耦每个表的难度。出现在此矩阵右上方象限中的表格是初始解耦工作的理想候选表，因为它们最容易分开。在 ER 图中，这些表几乎没有外键关系或其他依赖关系。解除这些表的耦合后，可以转向关系更复杂的表。



> **Note**
>
> 数据库结构通常反映应用程序架构。在数据库级别更容易解耦的表通常对应于更容易在应用程序级别转换为微服务的组件。

将业务逻辑从数据库迁移到应用层

将业务逻辑从数据库存储的过程、触发器和函数迁移到应用层服务是分解单体数据库的关键步骤。这种转型提高了服务自主性，简化了维护并增强了可扩展性。本节提供有关分析数据库逻辑、规划迁移策略，然后在保持业务连续性的同时实施转型的指导。它还讨论了如何制定有效的回滚计划。

本节包含以下主题：

第 1 阶段：分析业务逻辑

在对整体数据库进行现代化改造时，必须首先对现有的数据库逻辑进行全面分析。本阶段侧重于三个主要类别：

- 存储过程通常包含关键业务操作，包括数据操作逻辑、业务规则、验证检查和计算。作为应用程序业务逻辑的核心组件，它们需要仔细分解。例如，金融组织的存储过程可能会处理利息计算、账户对账和合规性检查。
- 触发器是处理审计跟踪、数据验证、计算和跨表一致性的关键数据库组件。例如，零售组织可能使用触发器来管理整个订单处理系统的库存更新，这表明了自动化数据库操作的复杂性。
- 数据库中的@@ 函数主要管理数据转换、计算和查找操作。它们通常嵌入在多个过程和应用程序中。例如，医疗保健组织可能使用函数来标准化患者数据或查找医疗代码。

每个类别都代表嵌入在数据库层中的业务逻辑的不同方面。您需要仔细评估和规划每一项才能将其迁移到应用层。

在此分析阶段，客户通常面临三个重大挑战。首先，复杂的依赖关系是通过嵌套过程调用、跨架构引用和隐式数据依赖关系出现的。其次，事务管理变得至关重要，尤其是在处理多步骤事务和在分布式系统之间保持数据一致性时。第三，必须仔细评估性能注意事项，特别是对于批处理操作、批量数据更新和实时计算，这些问题目前受益于接近数据。

为了有效地解决这些难题，可以使用 AWS Schema Conversion Tool (AWS SCT) 进行初步分析，然后使用详细的依赖关系映射工具。这种方法可以帮助您了解数据库逻辑的全部范围，并创建全面的迁移策略，在分解过程中保持业务连续性。

通过全面了解这些组件和挑战，您可以更好地规划现代化之旅，并就迁移到基于微服务的架构期间优先考虑哪些要素做出明智的决定。

分析数据库代码组件时，请为每个存储过程、触发器和函数创建全面的文档。首先要清楚地描述其目的和核心功能，包括其实施的业务规则。详细说明所有输入和输出参数，并记下它们的数据类型和有效范围。绘制与其他数据库对象、外部系统和下游进程的依赖关系。明确定义事务边界和隔离要求以维护数据完整性。记录任何性能预期，包括响应时间要求和资源利用模式。最后，分析使用模式以了解峰值负载、执行频率和关键业务时段。

第 2 阶段：对业务逻辑进行分类

有效的数据库分解需要按关键维度对数据库逻辑进行系统分类：复杂性、业务影响、依赖关系、使用模式和迁移难度。此分类可帮助您识别高风险组件、确定测试要求和确定迁移优先级。例如，具有高业务影响力和频繁使用的复杂存储过程需要仔细的规划和大量的测试。但是，依赖性最小的简单、很少使用的函数可能适用于早期迁移阶段。

这种结构化方法创建了一个平衡的迁移路线图，在保持系统稳定的同时，最大限度地减少了业务中断。通过了解这些相互关系，您可以改进分解工作的顺序并适当地分配资源。

第 3 阶段：迁移业务逻辑

在对业务逻辑进行分析和分类之后，是时候对其进行迁移了。将业务逻辑迁出单体数据库时，有两种方法：将数据库逻辑移至应用层，或将业务逻辑移至作为微服务一部分的另一个数据库。

如果将业务逻辑迁移到应用程序，则数据库表仅存储数据，数据库不包含任何业务逻辑。这是推荐的方法。您可以使用 Ispirer 或生成式人工智能工具（例如 A mazon Q Developer）或Kiro将数据库业务逻辑转换为应用程序层，例如转换为 Java。有关更多信息，请参阅将业务逻辑从数据库迁移到应用程序以实现更快的创新和灵活性（AWS 博客文章）。

如果将业务逻辑迁移到另一个数据库，则可以使用 AWS Schema Conversion Tool (AWS SCT) 将现有数据库架构和代码对象转换为目标数据库。它支持专门构建的 AWS 数据库服务，例如亚马逊 Dynam oDB、Amazon Aurora 和 Amazon R edsh ift。通过提供全面的评估报告和自动转换功能， AWS SCT 有助于简化过渡过程，使您能够专注于优化新的数据库结构以提高性能和可扩展性。随着现代化项目的进展， AWS SCT 可以处理增量转换以支持分阶段的方法，从而使您能够验证和微调数据库转换的每个步骤。

业务逻辑的回滚策略

任何分解策略的两个关键方面是保持向后兼容性和实施全面的回滚程序。这些要素相互配合，有助于保护过渡期间的运营。本节介绍如何在分解过程中管理兼容性，以及如何建立有效的紧急回滚功能以防范潜在问题。

保持向后兼容性

在数据库分解过程中，保持向后兼容性对于平稳过渡至关重要。在逐步实现新功能的同时，暂时保留现有的数据库程序。使用版本控制来跟踪所有更改并同时管理多个数据库版本。规划较长的共存期，在此期间，源系统和目标系统都必须可靠运行。这为在停用传统组件之前测试和验证新系统提供了时间。这种方法可以最大限度地减少业务中断，并在需要时为回滚提供安全网。

紧急回滚计划

全面的回滚策略对于安全的数据库分解至关重要。在代码中实现功能标志，以控制哪个版本的业务逻辑处于活动状态。这使您无需更改部署即可在新实现和原始实现之间即时切换。这种方法可以对过渡进行精细控制，并帮助您在出现问题时快速回滚。将原始逻辑保留为经过验证的备份，并维护详细的回滚程序，其中指定触发器、职责和恢复步骤。

定期在各种条件下测试这些回滚方案，以验证其有效性，并确保团队熟悉应急程序。功能标志还可以通过有选择地为特定用户组或交易启用新功能来实现逐步推出。这为过渡期间提供了额外的风险缓解层。

在数据库分解过程中解耦表关系

本节提供有关在整体数据库分解期间分解复杂的表关系和 JOIN 操作的指导。表联接基于两个或多个表中的相关列来合并两张或更多表中的行。分离这些关系的目的是减少表之间的高度耦合，同时保持微服务间的数据完整性。

本节包含以下主题：

非正规化策略

非规范化是一种数据库设计策略，它涉及通过跨表合并或复制数据来故意引入冗余。将大型数据库拆分为小型数据库时，跨服务复制一些数据可能是有意义的。例如，将基本的客户详细信息（例如姓名和电子邮件地址）存储在营销服务和订单服务中，无需持续进行跨服务查找。营销服务可能需要客户偏好和联系信息来进行广告活动定位，而订单服务则需要相同的数据来处理订单和发出通知。虽然这会产生一些数据冗余，但它可以显著提高服务绩效和独立性，使营销团队能够在不依赖实时客户服务查询的情况下开展活动。

在实施非规范化时，请重点关注通过仔细分析数据访问模式来识别的经常访问的字段。您可以使用诸如Oracle AWR报告或pg_stat_statements之类的工具来了解哪些数据通常是一起检索的。领域专家还可以提供有关自然数据分组的宝贵见解。请记住，非规范化不是一种 all-or-nothing方法，只能使用可明显提高系统性能或减少复杂依赖关系的重复数据。

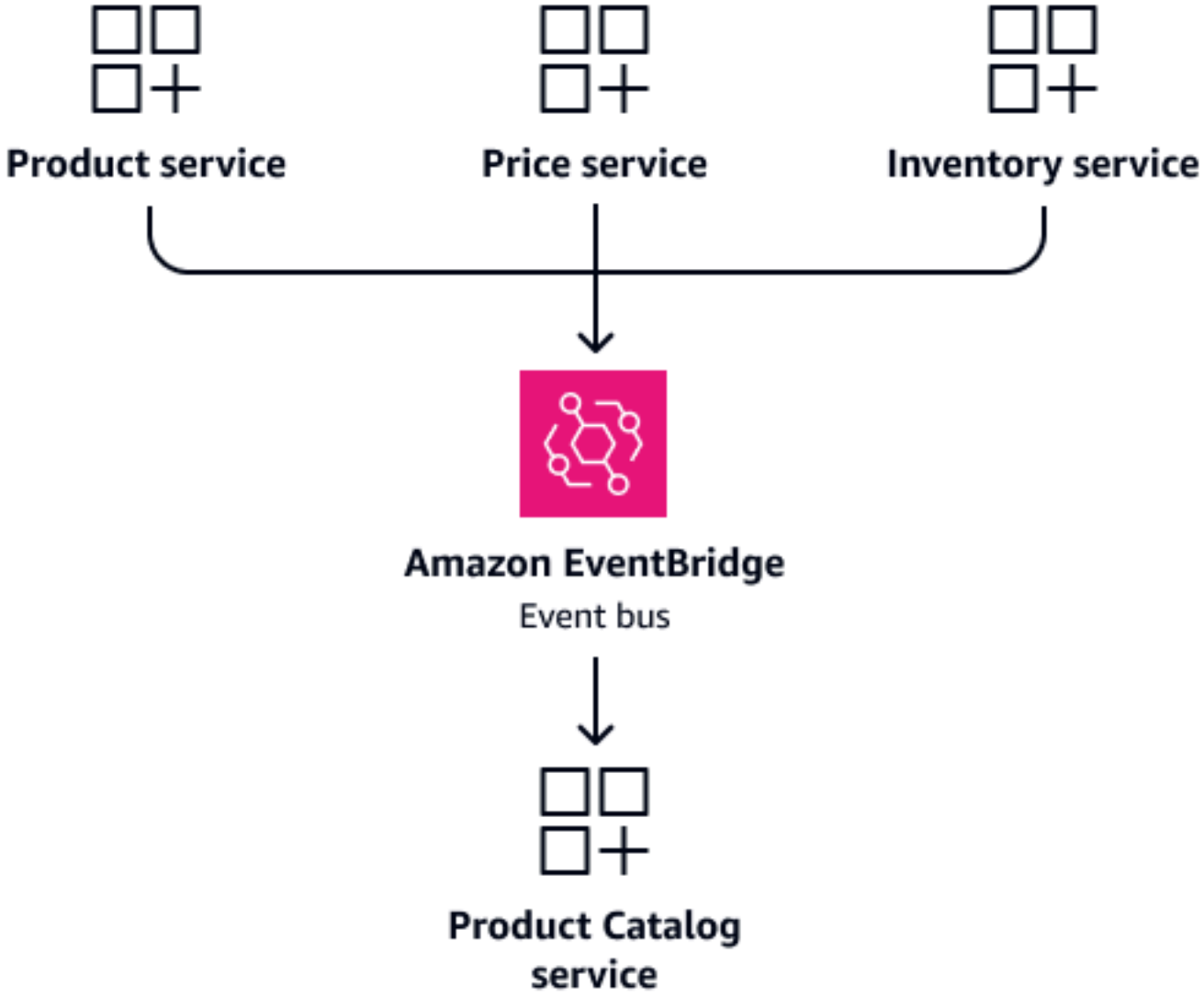
Reference-by-key 策略

reference-by-key 策略是一种数据库设计模式，其中实体之间的关系通过唯一密钥维护，而不是存储实际的相关数据。现代微服务通常只存储相关数据的唯一标识符，而不是传统的外键关系。例如，订单服务不是将所有客户详细信息保留在订单表中，而是仅存储客户 ID，并在需要时通过 API 调用检索其他客户信息。这种方法保持了服务的独立性，同时确保了对相关数据的访问。

CQRS 模式

命令查询责任分离 (CQRS) 模式将数据存储的读取和写入操作分开。这种模式在具有高性能要求的复杂系统中特别有用，尤其是那些具有非对称 read/write 负载的系统。如果您的应用程序经常需要组合来自多个来源的数据，则可以创建专用的 CQRS 模型，而不是复杂的联接。例如，与其根据每个请求联接`ProductPricing`、和`Inventory`表，不如维护一个包含必要数据的合并`Product Catalog`表。这种方法的好处可能超过额外表格的成本。

考虑这样一个场景：`ProductPrice`、和`Inventory`服务经常需要产品信息。与其将这些服务配置为直接访问共享表，不如创建专用`Product Catalog`服务。该服务维护自己的数据库，其中包含统一的产品信息。它充当与产品相关的查询的单一事实来源。当产品详情、价格或库存水平发生变化时，相应的服务可以发布事件以更新`Product Catalog`服务。这样既能保持数据一致性，又能保持服务独立性。下图显示了此配置，其中 Amazon EventBridge 充当事件总线。



如下一节所基于事件的数据同步述，通过事件更新 CQRS 模型。当产品详情、价格或库存水平发生变化时，相应的服务会发布事件。该Product Catalog服务订阅这些事件并更新其合并视图。这无需复杂联接即可快速读取，并且可以保持服务独立性。

基于事件的数据同步

基于事件的数据同步是一种捕获数据更改并作为事件传播的模式，这使不同的系统或组件能够保持同步的数据状态。当数据发生变化时，与其立即更新所有相关数据库，不如发布一个事件来通知订阅的服务。例如，当客户在Customer服务中更改其送货地址时，一个CustomerUpdated事件会根据每项Order服务的计划启动对服务和服务的更新。Delivery这种方法用灵活、可扩展的事件驱动更新取代了僵化的表格联接。有些服务可能有短暂的过时数据，但需要权衡的是系统可扩展性和服务独立性的提高。

实现表格联接的替代方案

从读取操作开始数据库分解，因为读取操作通常更易于迁移和验证。在读取路径稳定之后，处理更复杂的写入操作。对于关键的高性能要求，可以考虑实施 CQRS 模式。使用单独的、经过优化的数据库进行读取，同时保留另一个数据库进行写入。

通过为跨服务调用添加重试逻辑并实现适当的缓存层，构建弹性系统。密切监控服务交互，并针对数据一致性问题设置警报。最终目标不是所有地方的完美一致性，而是创建性能良好的独立服务，同时保持可接受的数据准确性，以满足您的业务需求。

微服务的分离性质为数据管理带来了以下新的复杂性：

- 数据是分布式的。现在，数据存储在单独的数据库中，这些数据库由独立的服务管理。
- 跨服务的实时同步通常是不切实际的，因此需要一个最终的一致性模型。
- 以前在单个数据库事务中发生的操作现在跨越多个服务。

要应对这些挑战，请执行以下操作：

- 实施事件驱动架构-使用消息队列和事件发布在服务之间传播数据更改。有关更多信息，请参阅在 Serverless Land 上构建事件驱动架构。
- 采用 saga 编排模式 — 这种模式可帮助您管理分布式事务并维护服务间的数据完整性。有关更多信息，请参阅博客上AWS 使用传奇编排模式构建无服务器分布式应用程序。
- 失败设计 — 整合重试机制、断路器和补偿事务以处理网络问题或服务故障。

- 使用版本标记-跟踪数据版本以管理冲突并确保应用了最新的更新。
- 定期对账 — 实施定期的数据同步流程，以捕捉和纠正任何不一致之处。

基于场景的示例

以下架构示例有两个表，一个Customer表和一个Order表：

```
-- Customer table
CREATE TABLE customer (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255),
    phone VARCHAR(20),
    address TEXT,
    created_at TIMESTAMP
);

-- Order table
CREATE TABLE order (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

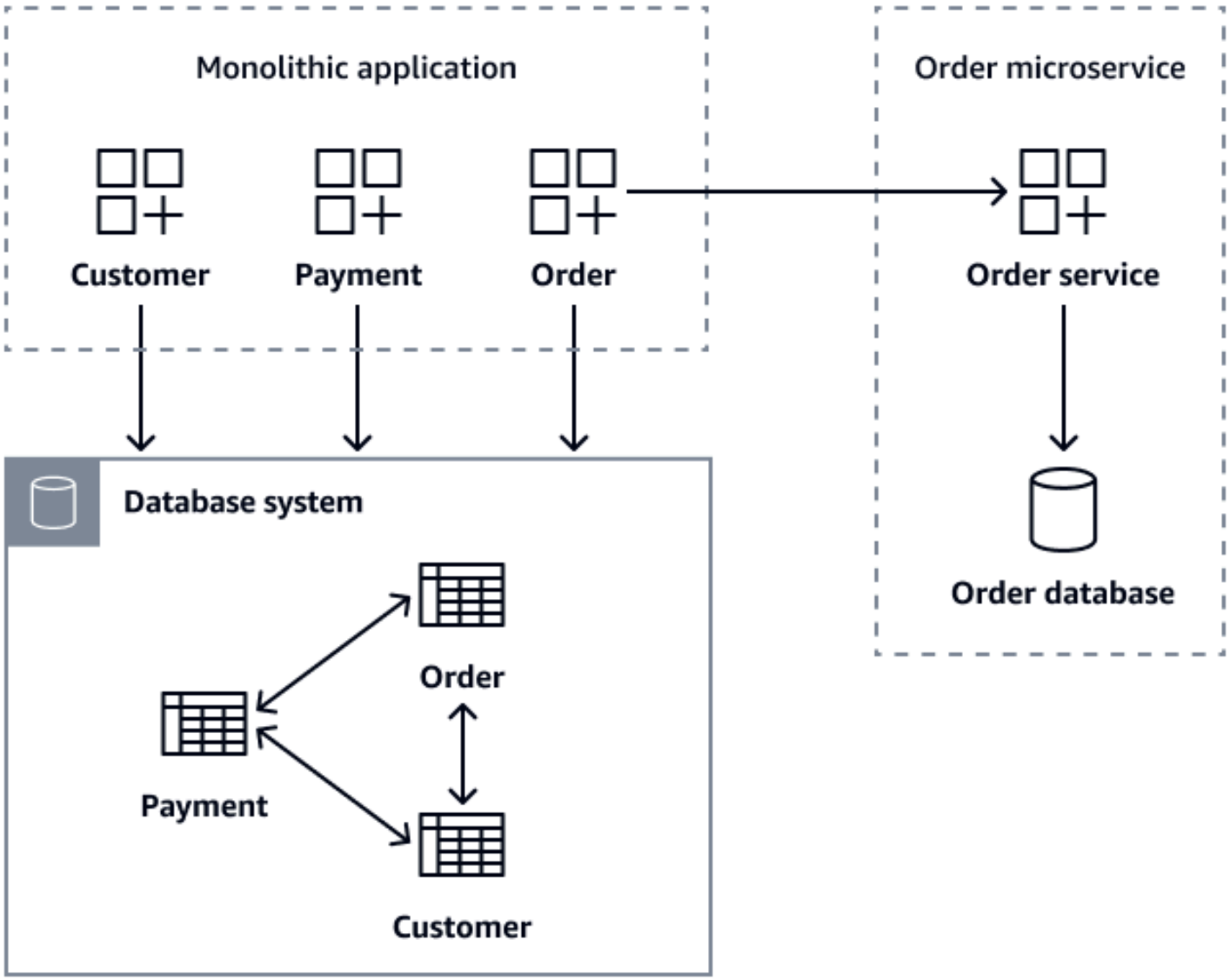
以下是如何使用非规范化方法的示例：

```
CREATE TABLE order (
    order_id INT PRIMARY KEY,
    customer_id INT,                  -- Reference only
    customer_first_name VARCHAR(100), -- Denormalized
    customer_last_name VARCHAR(100),  -- Denormalized
    customer_email VARCHAR(255),      -- Denormalized
    order_date TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(50)
);
```

新Order表中包含非标准化的客户姓名和电子邮件地址。customer_id被引用了，并且Customer表中没有外键约束。以下是这种非规范化方法的好处：

- 该Order服务可以显示带有客户详细信息的订单历史记录，并且不需要对Customer微服务进行API调用。
- 如果Customer服务已关闭，则该Order服务仍能完全正常运行。
- 订单处理和报告的查询运行速度更快。

下图显示了一个单体应用程序，该应用程序使用getOrder(customer_id)、getOrder(order_id)getCustomerOders(customer_id)、和createOrder(Order order) API 调用微服务来检索订单数据。Order



在微服务迁移期间，您可以将Order表保存在单体数据库中，以此作为过渡性安全措施，从而确保旧版应用程序保持正常运行。但是，所有与新订单相关的操作都必须通过微服务 API 进行路由，Order微服务 API 维护自己的数据库，同时写入旧数据库作为备份。这种双写模式提供了一个安全网。它允许在保持系统稳定性的同时逐步迁移。在所有客户成功迁移到新的微服务后，您可以弃用单体数据库中的旧Order表。在将整体应用程序及其数据库分解为单独Customer的Order微服务之后，保持数据一致性成为主要挑战。

数据库分解的最佳实践

在分解单体数据库时，组织必须建立明确的框架来跟踪进展、维护系统知识和应对新出现的挑战。本节提供了衡量分解成功率、维护关键文档、实施持续改进流程和应对常见挑战的最佳实践。了解并遵循这些指导方针有助于确保数据库分解工作能带来预期的收益，同时最大限度地减少运营中断和技术债务。

本节包含以下主题：

- 衡量成功
- 文档要求
- 持续改进策略
- 克服数据库分解中的常见挑战

衡量成功

通过技术、运营和业务指标的组合来跟踪分解成功率。从技术上讲，监控查询响应时间、系统正常运行时间的改善和部署频率的增加。在运营方面，衡量事件减少情况、问题解决速度和资源利用率改善情况。对于开发，跟踪功能实现速度、发布周期加速以及跨团队依赖关系的减少情况。业务影响应导致更快地降低运营成本 time-to-market，并提高客户满意度。这些指标通常是在范围阶段定义的。有关更多信息，请参阅本指南中的定义数据库分解的范围和要求。

文档要求

维护 up-to-date系统架构文档，其中包含明确的服务边界、数据流和接口规格。使用架构决策记录 (ADRs) 来记录关键的技术决策，包括其背景、后果和考虑的替代方案。例如，记录为什么要先将特定服务分开，或者是如何权衡某些数据一致性的。

安排每月架构审查，通过关键指标评估系统运行状况：性能趋势、安全合规性和跨服务依赖关系。包括开发团队关于集成挑战和运营问题的反馈。这种定期审查周期可帮助您及早发现新出现的问题，并验证分解工作是否与业务目标保持一致。

持续改进策略

将数据库分解视为迭代过程，而不是一次性项目。监控系统性能指标和服务交互以确定优化机会。每个季度，根据运营影响和维护成本优先解决技术债务。例如，自动执行经常执行的数据库操作，扩大监控范围，并根据学习到的模式完善部署程序。

克服数据库分解中的常见挑战

性能优化需要多方面的方法。在服务边界实施战略缓存，根据实际使用情况优化查询模式，并持续监控关键指标。通过分析趋势和设置明确的干预阈值，主动解决性能瓶颈。

数据一致性挑战要求谨慎选择架构。为跨服务更新实现事件驱动模式，并使用传奇编排模式处理复杂事务。定义明确的服务边界，并在业务需求允许的情况下接受最终一致性。一致性和服务自主性之间的这种平衡对于成功分解至关重要。

卓越运营需要日常任务的自动化和跨服务的标准化程序。通过明确的警报阈值进行全面监控，并投资于定期的团队培训，以了解新的模式和工具。这种系统的运营方法在管理复杂性的同时促进了可靠的服务交付。

数据库分解常见问题解答

这个全面的常见问题解答部分解决了组织在开展数据库分解项目时面临的最常见问题和挑战。从定义初始范围和要求到迁移存储过程，这些问题提供了实用的见解和战略方法，以帮助团队成功完成数据库现代化之旅。无论您是处于计划阶段还是已经在执行分解策略，这些答案都可以帮助您避免常见的陷阱并实施最佳实践以获得最佳结果。

本节包含以下主题：

FAQs 关于定义范围和要求

本指南的定义数据库分解的范围和要求部分讨论了如何分析互动、映射依赖关系和建立成功标准。本常见问题解答部分解决了有关建立和管理项目边界的关键问题。无论您是在处理不明确的技术限制、相互冲突的部门需求还是不断变化的业务需求，这些都 FAQs 为保持平衡方法提供了实用指导。

本节包含以下问题：

最初的范围定义应该有多详细？

从客户的需求出发，用足够的细节定义项目范围，以确定系统边界和关键依赖关系，同时保持发现的灵活性。绘制基本要素，包括系统接口、关键利益相关者和主要技术限制。从小处着手，选择系统中可提供可衡量价值的有限低风险部分。这种方法可以帮助团队在处理更复杂的组件之前学习和调整策略。

记录推动分解工作的关键业务需求，但要避免过度指定在实施过程中可能发生变化的细节。这种平衡的方法可确保团队能够清晰地向前迈进，同时能够适应现代化过程中出现的新见解和挑战。

如果我在启动项目后发现了其他依赖关系怎么办？

随着项目的进展，预计还会发现其他依赖关系。维护实时依赖关系日志并定期进行范围审查，以评估对时间表和资源的影响。实施明确的变更管理流程，并在项目计划中加入缓冲时间，以处理意外发现。目标不是防止变更，而是要有效地管理变更。这有助于团队在保持项目势头的同时快速适应。

我该如何处理来自不同部门的利益相关者，他们有相互矛盾的要求？

根据业务价值和系统影响，通过明确的优先顺序来处理相互冲突的部门需求。获得高管的支持，以推动关键决策并快速解决冲突。定期安排利益相关者协调会议，讨论权衡问题并保持透明度。记录所有决策及其理由，以促进清晰的沟通并保持项目势头。将讨论重点放在可量化的业务收益上，而不是部门偏好上。

当文档不佳或过时时，评估技术限制的最佳方法是什么？

面对糟糕的文档时，请将传统分析与现代 AI 工具相结合。使用大型语言模型 (LLMs) 分析代码存储库、日志和现有文档，以确定模式和潜在限制。采访经验丰富的开发人员和数据库架构师，以验证人工智能发现并发现未记录的限制。部署增强了 AI 功能的监控工具，以观察系统行为并预测潜在问题。

创建小型技术实验来验证您的假设。您可以使用 AI 驱动的测试工具来加快流程。将发现结果记录在知识库中，该知识库可通过 AI 辅助更新不断增强。考虑聘请复杂领域的主题专家，并使用人工智能配对编程工具来加快他们的分析和记录工作。

如何在眼前的业务需求和长期技术目标之间取得平衡？

制定分阶段的项目路线图，使当前业务需求与长期技术目标保持一致。尽早确定能带来切实价值的速赢之处，这样你就可以建立利益相关者的信心。将分解分解为明确的里程碑。在朝着架构目标迈进的同时，两者都应提供可衡量的业务收益。通过定期的路线图审查和调整，保持灵活性，以满足紧急业务需求。

如何确保我不会错过沉默的利益相关者的关键要求？

绘制组织内所有潜在利益相关者的地图，包括下游系统所有者和间接用户。通过结构化访谈、研讨会和定期评审会议创建多个反馈渠道。进行构建 proof-of-concepts和原型设计，使需求变得切实可行，并引发有意义的讨论。例如，显示系统依赖关系的简单仪表板通常会显示隐藏的利益相关者和最初并不明显的需求。

定期与直言不讳的利益相关者进行验证会议，并确保捕捉到所有观点。关键见解通常来自最接近日常运营的人，而不是规划会议中最响亮的声音。

这些建议是否适用于单体大型机数据库？

本指南中描述的方法也适用于分解单体大型机数据库。这些数据库面临的主要挑战是管理各利益攸关方的需求。本指南中的技术建议可能适用于单体大型机数据库。如果大型机具有关系数据库，例如在线事务处理 (OLTP) 数据库，则许多建议都适用。对于在线分析处理 (OLAP) 数据库，例如用于生成业务报告的数据库，则只有部分建议适用。

FAQs 关于控制数据库访问权限

本指南的在分解过程中控制数据库访问权限部分讨论了使用数据库包装器服务模式控制数据库访问权限。本常见问题解答部分解决了有关引入数据库包装服务的常见问题和问题，包括其对性能、处理现有存储过程、管理复杂事务和监督架构更改的潜在影响。

本节包含以下问题：

包装器服务不会成为新的瓶颈吗？

虽然数据库包装器服务确实增加了额外的网络跳跃，但影响通常微乎其微。您可以横向扩展服务，受控访问的好处通常超过较小的性能成本。将其视为性能和可维护性之间的临时权衡。

现有的存储过程会怎样？

最初，数据库包装器服务可以将存储过程作为服务方法公开。随着时间的推移，您可以逐渐将逻辑移到应用程序层，从而改善测试和版本控制。逐步迁移业务逻辑以最大限度地降低风险。

过渡期间如何管理架构更改？

通过包装器服务团队集中控制架构变更。该团队负责保持所有消费者的全面知名度。该团队审查提议的变更以获得全系统影响，与受影响的团队进行协调，并使用受控的部署流程实施修改。例如，在添加新字段时，该团队应通过实现默认值或最初允许空值来保持向后兼容性。

建立明确的变更管理流程，包括影响评估、测试要求和回滚程序。使用数据库版本控制工具，并保持所有更改的清晰文档。这种集中式方法可防止架构修改中断依赖的服务，并保持系统的稳定性。

FAQs 关于分析内聚力和耦合

了解并有效分析数据库的耦合和内聚力是成功分解数据库的基础。本指南的分析数据库分解的内聚力和耦合部分讨论了耦合和内聚力。本常见问题解答部分解决了有关确定适当的粒度级别、选择正确的分析工具、记录发现结果以及确定耦合问题优先顺序的关键问题。

本节包含以下问题：

- 在分析耦合时，如何确定正确的粒度级别？
- 我可以使用哪些工具来分析数据库的耦合和内聚力？
- 记录耦合和凝聚力发现的最佳方法是什么？
- 如何确定首先要解决的耦合问题的优先顺序？
- 如何处理跨多个操作的交易？

在分析耦合时，如何确定正确的粒度级别？

首先对数据库关系进行广泛分析，然后系统地向下钻取以确定自然分离点。使用数据库分析工具映射表级关系、架构依赖关系和事务边界。例如，检查 SQL 查询中的联接模式以了解数据访问依赖关系。您还可以分析事务日志以确定业务流程边界。

重点关注耦合自然最少的领域。它们通常与业务领域边界保持一致，代表最佳的分解点。在确定适当的服务边界时，应同时考虑技术耦合（例如共享表和外键）和业务耦合（例如流程和报告需求）。

我可以使用哪些工具来分析数据库的耦合和内聚力？

您可以结合使用自动化工具和手动分析来评估数据库的耦合度和内聚力。以下工具可以帮助您进行此评估：

- 架构可视化工具-您可以使用SchemaSpy或pgAdmin之类的工具生成 ER 图。这些图表揭示了表格关系和潜在的耦合点。
- 查询分析工具-您可以使用pg_stat_statements或SQL Server Query Store来识别经常连接的表和访问模式。
- 数据库分析工具 — 诸如Oracle SQL Developer或之类的工具，MySQL Workbench可提供对查询性能和数据依赖关系的见解。

- 依赖关系映射工具 — AWS Schema Conversion Tool (AWS SCT) 可以帮助您可视化架构关系并识别紧密耦合的组件。 vFunction可以通过分析应用程序的功能和域边界来帮助您识别域边界。
- 事务监控工具-您可以使用数据库特定的工具（例如Oracle Enterprise Manager或 SQL Server Extended Events）来分析事务边界。
- 业务逻辑迁移工具 — 您可以使用Ispirer生成式 AI 工具（例如 A mazon Q Developer）或Kiro将数据库业务逻辑转换为应用程序层，例如转换为 Java。

将这些自动分析与对业务流程和领域知识的手动审查相结合，以全面了解系统耦合。这种多方面的方法可确保在分解策略中同时考虑技术和业务视角。

记录耦合和凝聚力发现的最佳方法是什么？

创建全面的文档，以可视化数据库关系和使用模式。以下是可用于记录发现结果的资产类型：

- 依赖矩阵 — 映射表依赖关系并突出显示高耦合区域。
- 关系图-使用 ER 图显示架构连接和外键关系。
- 表格使用热图-可视化各表的查询频率和数据访问模式。
- 交易流程图-记录多表交易及其边界。
- 域边界图-根据业务领域概述潜在的服务边界。

将这些工件合并到文档中，并随着分解的进展定期对其进行更新。对于图表，您可以使用draw.io或之类的工具Lucidchart。考虑实施一个 wiki，便于团队访问和协作。这种多方面的文档方法提供了对系统耦合和内聚的清晰、共同的理解。

如何确定首先要解决的耦合问题的优先顺序？

根据对业务和技术因素的平衡评估，确定耦合问题的优先顺序。根据业务影响（例如收入和客户体验）、技术风险（例如系统稳定性和数据完整性）、实施工作和团队能力来评估每个问题。创建一个优先级矩阵，在这些维度上对每个问题进行从 1-5 的分数。此矩阵可帮助您识别风险可控的最有价值的机会。

从与现有团队专业知识相一致的高影响、低风险变更开始。这可以帮助您建立组织信心和动力，以应对更复杂的变革。这种方法可以促进现实执行并最大限度地提高业务价值。定期审查和调整优先级，以帮助与不断变化的业务需求和团队能力保持一致。

如何处理跨多个操作的交易？

通过精心设计的服务级别协调来处理多操作事务。为复杂的分布式事务实现传奇模式。将它们分解为可以独立管理的较小的、可逆的步骤。例如，订单处理流程可以分为不同的步骤，分别用于库存检查、付款处理和订单创建，每个步骤都有自己的补偿机制。

在可能的情况下，重新设计操作使其更具原子性，从而减少对分布式事务的需求。当分布式事务不可避免时，应实施强大的跟踪和补偿机制，以提高数据一致性。监控交易完成率并实施明确的错误恢复程序，以保持系统的可靠性。

FAQs 关于将业务逻辑迁移到应用层

将业务逻辑从数据库迁移到应用层是数据库现代化的一个关键而复杂的方面。本指南的将业务逻辑从数据库迁移到应用层部分将讨论这种业务逻辑迁移。本常见问题解答部分解决了有关有效管理此过渡的常见问题，从选择迁移的初始候选对象到处理复杂的存储过程和触发器。

本节包含以下问题：

- 如何确定要先迁移哪些存储过程？
- 将逻辑转移到应用层有哪些风险？
- 将逻辑移出数据库时如何保持性能？
- 我该如何处理涉及多个表的复杂存储过程？
- 迁移期间如何处理数据库触发器？
- 测试迁移的业务逻辑的最佳方法是什么？
- 当数据库和应用程序逻辑都存在时，如何管理过渡期？
- 如何处理应用程序层中以前由数据库管理的错误场景？

如何确定要先迁移哪些存储过程？

首先确定能够提供低风险和高学习价值的最佳组合的存储过程。重点关注依赖性最小、功能清晰且业务影响不大的程序。它们是初始迁移的理想人选，因为它们可以帮助团队建立信心并建立模式。例如，选择处理简单数据操作的过程，而不是管理复杂事务或关键业务逻辑的过程。

使用数据库监控工具来分析使用模式，将不经常访问的过程识别为早期候选程序。这种方法最大限度地降低了业务风险，同时为日后处理更复杂的迁移提供了宝贵的经验。根据复杂性、业务关键性和依赖性级别对每个过程进行评分，以创建按优先顺序排列的迁移顺序。

将逻辑转移到应用层有哪些风险？

将数据库逻辑迁移到应用层会带来几个关键挑战。由于网络调用的增加，系统性能可能会降低，特别是对于以前在数据库中处理的数据密集型操作。事务管理变得更加复杂，需要仔细协调才能在分布式操作中保持数据完整性。确保数据一致性变得具有挑战性，对于以前依赖数据库级限制的操作来说尤其如此。

迁移期间可能出现的业务中断以及开发人员的学习曲线也令人担忧。通过全面规划、在分阶段环境中进行广泛测试以及从不太关键的组件开始的逐步迁移来降低这些风险。实施强大的监控和回滚程序，以快速识别和解决生产中的问题。

将逻辑移出数据库时如何保持性能？

为经常访问的数据实施适当的缓存机制，优化数据访问模式以最大限度地减少网络调用，并使用批处理进行批量操作。对于 non-time-critical操作，可以考虑使用异步处理来提高系统响应能力。

密切监控应用程序性能指标，并根据需要对其进行调整。例如，您可以将多个单行操作替换为批量处理，可以缓存不经常更改的参考数据，还可以优化查询模式以减少数据传输。定期进行性能测试和调整有助于系统保持可接受的响应时间，并提高可维护性和可扩展性。

我该如何处理涉及多个表的复杂存储过程？

通过系统分解来处理复杂的多表存储过程。首先将它们分解为更小、逻辑上连贯的组件，并确定明确的事务边界和数据依赖关系。为每个逻辑组件创建服务接口。这可以帮助您在不中断现有功能的情况下逐步迁移。

实施 step-by-step迁移，从耦合度最低的组件开始。对于高度复杂的过程，可以考虑在迁移更简单的部件时将其暂时保存在数据库中。这种混合方法可在您朝着架构目标迈进的同时保持系统的稳定性。在迁移过程中持续监控性能和功能，并准备好根据结果调整策略。

迁移期间如何处理数据库触发器？

在维护系统功能的同时，将数据库触发器转换为应用程序级事件处理程序。将同步触发器替换为事件驱动的模式，这些模式会向异步操作发送消息队列。考虑使用亚马逊简单通知服务 (Amazon SNS) 或亚马逊简单队列服务 (A mazon SQS) Simple Queue Service 作为消息队列。对于审计要求，请实施应用程序级日志记录或使用数据库变更数据捕获 (CDC) 功能。

分析每个触发器的目的和重要性。有些触发器可能更适合应用程序逻辑，而另一些则可能需要事件源模式来保持数据的一致性。先从简单的触发器（例如审计日志）开始，然后再处理管理业务规则或数据完整性的复杂触发器。在迁移过程中仔细监控，确保不会丢失功能或数据一致性。

测试迁移的业务逻辑的最佳方法是什么？

在部署迁移的业务逻辑之前，先实施多层测试方法。从新应用程序代码的单元测试开始，然后添加涵盖end-to-end业务流程的集成测试。并行运行新旧实现，然后比较结果以验证功能等效性。在各种负载条件下进行性能测试，以验证系统行为是否符合或超过以前的功能。

使用功能标志来控制部署，以便在出现问题时可以快速回滚。让业务用户参与验证，尤其是关键工作流程的验证。在初始部署期间监控关键指标，并逐步增加新实施的流量。在整个过程中，保持在需要时恢复到原始数据库逻辑的能力。

当数据库和应用程序逻辑都存在时，如何管理过渡期？

当同时使用数据库和应用程序逻辑时，请实现控制流量并允许在新旧实现之间快速切换的功能标志。保持严格的版本控制，并清楚地记录实施及其各自的责任。为两个系统设置全面监控，以快速识别任何差异或性能问题。

为每个迁移的组件制定明确的回滚程序，以便在需要时可以恢复到原始逻辑。定期与所有利益相关者就过渡状态、潜在影响和上报程序进行沟通。这种方法可以帮助您逐步迁移，同时保持系统的稳定性和利益相关者的信心。

如何处理应用程序层中以前由数据库管理的错误场景？

用强大的应用层机制取代数据库级别的错误处理。针对瞬态故障，实现断路器和重试逻辑。使用补偿事务来维护分布式操作中的数据一致性。例如，如果付款更新失败，应用程序应在定义的限制内自动重试，并在需要时启动补偿措施。

设置全面的监控和警报以快速发现问题，并维护详细的审计日志以进行故障排除。将错误处理设计为尽可能自动化，并为需要人工干预的场景定义清晰的上报路径。这种多层方法提供了系统弹性，同时保持了数据的完整性和业务流程的连续性。

数据库分解的后续步骤 AWS

在通过数据库封装服务实施初始数据库分解策略并将业务逻辑转移到应用程序层之后，组织必须规划下一次演进。本节概述了继续现代化之旅的关键注意事项。

本节包含以下主题：

数据库分解的增量策略

数据库分解遵循三个不同阶段的逐渐演变。团队首先使用数据库包装器服务封装整体数据库以控制访问权限。然后，他们开始将数据拆分为特定于服务的数据库，同时维护主数据库以满足传统需求。最后，他们完成了业务逻辑的迁移，以便过渡到完全独立的服务数据库。

在整个旅程中，团队必须实施谨慎的数据同步模式，并持续验证服务之间的一致性。性能监控对于及早发现和解决潜在问题至关重要。随着服务的独立发展，应根据实际使用模式对其架构进行优化，并且应删除随着时间的推移而积累的冗余结构。

这种渐进式方法有助于最大限度地降低风险，同时在整个转型过程中保持系统稳定性。

分布式数据库环境的技术注意事项

在分布式数据库环境中，性能监控对于尽早发现和解决瓶颈至关重要。团队必须实施全面的监控系统和缓存策略，以保持绩效水平。 Read/write 拆分可以有效地平衡整个系统的负载。

数据一致性需要在分布式服务之间进行仔细的编排。团队应酌情实施最终的一致性模式，并建立明确的数据所有权界限。强大的监控功能可促进所有服务的数据完整性。

此外，安全性必须不断发展以适应分布式架构。每项服务都需要精细的安全控制，您的访问模式需要定期审查。在这种分布式环境中，增强监控和审计变得至关重要。

为支持分布式架构而进行的组织变革

团队结构应与服务界限保持一致，以便明确所有权和问责制。Organizations 必须建立新的沟通模式，并在团队内部建立额外的技术能力。这种结构应支持现有服务的维护和架构的持续发展。

您必须更新操作流程才能处理分布式架构。团队必须修改部署程序，调整事件响应流程，并发展变更管理实践，以便在多个服务之间进行协调。

Resources（资源）

以下其他资源和工具可以帮助您的组织完成数据库分解之旅。

AWS 规范性指导

- 将Oracle数据库迁移到 AWS Cloud
- 开启的平台重置Oracle Database选项 AWS
- 云设计模式、架构和实现

AWS 博客文章

- 将业务逻辑从一个数据库迁移到另一个应用程序，以实现更快的创新和灵活性

AWS 服务

- AWS Application Migration Service
- AWS Database Migration Service (AWS DMS)
- Migration Evaluator
- AWS Schema Conversion Tool (AWS SCT)
- AWS Transform

其他工具

- AppEngine（Dynatrace 网站）
- Oracle Automatic Workload Repository（Oracle 网站）
- CAST Imaging（CAST 网站）
- Kiro（Kiro 网站）
- pgAdmin（pgAdmin 网站）
- pg_stat_statements（PostgreSQL 网站）
- SchemaSpy（SchemaSpy 网站）
- SQL Developer（Oracle 网站）

- SQLWays（Ispirer 网站）
- vFunction（vFunction 网站）

其他资源

- 从@@ 整体到微服务（网站）O'Reilly

文档历史记录

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

变更	说明	日期
大型机常见问题解答和 AI 工具	我们添加了这些建议是否适用于单体大型机数据库？ 常见问题解答，我们还添加了有关可在数据库分解期间使用的 AI 工具的其他信息。	2025 年 10 月 14 日
初次发布	—	2025 年 9 月 30 日

AWS 规范性指导词汇表

以下是 AWS 规范性指导提供的策略、指南和模式中的常用术语。若要推荐词条，请使用术语表末尾的提供反馈链接。

数字

7 R

将应用程序迁移到云中的 7 种常见迁移策略。这些策略以 Gartner 于 2011 年确定的 5 R 为基础，包括以下内容：

- 重构/重新架构：充分利用云原生功能来提高敏捷性、性能和可扩展性，以迁移应用程序并修改其架构。这通常涉及到移植操作系统和数据库。示例：将本地 Oracle 数据库迁移到 Amazon Aurora PostgreSQL 兼容版。
- 更换平台：将应用程序迁移到云中，并进行一定程度的优化，以利用云功能。示例：将本地 Oracle 数据库迁移到 AWS Cloud中的 Amazon Relational Database Service（Amazon RDS）for Oracle。
- 重新购买：转换到其他产品，通常是从传统许可转向 SaaS 模式。示例：将客户关系管理（CRM）系统迁移到 Salesforce.com。
- 重新托管（直接迁移）：将应用程序迁移到云中，无需进行任何更改即可利用云功能。示例：将本地 Oracle 数据库迁移到 AWS Cloud中 EC2 实例上的 Oracle。
- 重新放置（虚拟机监控器级直接迁移）：将基础设施迁移到云中，无需购买新硬件、重写应用程序或修改现有操作。您将服务器从本地平台迁移到同一平台的云服务中。示例：将Microsoft Hyper-V应用程序迁移到 AWS。
- 保留（重访）：将应用程序保留在源环境中。其中可能包括需要进行重大重构的应用程序，并且您希望将工作推迟到以后，以及您希望保留的遗留应用程序，因为迁移它们没有商业上的理由。
- 停用：停用或删除源环境中不再需要的应用程序。

A

ABAC

请参阅基于属性的访问控制。

抽象服务

请参阅托管服务。

ACID

请参阅原子性、一致性、隔离性、持久性。

主动-主动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步（通过使用双向复制工具或双写操作），两个数据库都在迁移期间处理来自连接应用程序的事务。这种方法支持小批量、可控的迁移，而不需要一次性割接。它比主动-被动迁移更灵活，但工作量更大。

主动-被动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步，但在将数据复制到目标数据库时，只有源数据库处理来自连接应用程序的事务。目标数据库在迁移期间不接受任何事务。

聚合函数

一种 SQL 函数，它对一组行进行操作并计算该组的单个返回值。聚合函数的示例包括 SUM 和 MAX。

AI

请参阅人工智能。

AIOps

请参阅人工智能运营。

匿名化

永久删除数据集中个人信息的过程。匿名化可以帮助保护个人隐私。匿名化数据不再被视为个人数据。

反模式

一种用于解决反复出现的问题的常用解决方案，而在这类问题中，此解决方案适得其反、无效或不如替代方案有效。

应用程序控制

一种安全方法，仅允许使用经批准的应用程序，以帮助保护系统免受恶意软件的侵害。

应用程序组合

有关组织使用的每个应用程序的详细信息的集合，包括构建和维护该应用程序的成本及其业务价值。这些信息是产品组合发现和分析过程的关键，有助于识别需要进行迁移、现代化和优化的应用程序并确定其优先级。

人工智能（AI）

计算机科学领域致力于使用计算技术执行通常与人类相关的认知功能，例如学习、解决问题和识别模式。有关更多信息，请参阅什么是人工智能？

人工智能操作 (AIOps)

使用机器学习技术解决运营问题、减少运营事故和人为干预以及提高服务质量的过程。有关如何在 AIOps AWS 迁移策略中使用的更多信息，请参阅操作集成指南。

非对称加密

一种加密算法，使用一对密钥，一个公钥用于加密，一个私钥用于解密。您可以共享公钥，因为它不用于解密，但对私钥的访问应受到严格限制。

原子性、一致性、隔离性、持久性（ACID）

一组软件属性，即使在出现错误、电源故障或其他问题的情况下，也能保证数据库的数据有效性和操作可靠性。

基于属性的访问权限控制（ABAC）

根据用户属性（如部门、工作角色和团队名称）创建精细访问权限的做法。有关更多信息，请参阅 AWS Identity and Access Management (I AM) 文档 AWS中的 AB AC。

权威数据来源

存储主要数据版本的位置，被认为是最可靠的信息源。您可以将数据从权威数据来源复制到其他位置，以便处理或修改数据，例如对数据进行匿名化、编辑或假名化。

可用区

中的一个不同位置 AWS 区域 ，不受其他可用区域故障的影响，并向同一区域中的其他可用区提供低成本、低延迟的网络连接。

AWS 云采用框架 (AWS CAF)

该框架包含指导方针和最佳实践 AWS ，可帮助组织制定高效且有效的计划，以成功迁移到云端。AWS CAF将指导分为六个重点领域，称为视角：业务、人员、治理、平台、安全和运营。业务、人员和治理角度侧重于业务技能和流程；平台、安全和运营角度侧重于技术技能和流程。例如，人

员角度针对的是负责人力资源（HR）、人员配置职能和人员管理的利益相关者。从这个角度来看，AWS CAF 为人员发展、培训和沟通提供了指导，以帮助组织为成功采用云做好准备。有关更多信息，请参阅 AWS CAF 网站和 AWS CAF 白皮书。

AWS 工作负载资格框架 (AWS WQF)

一种评估数据库迁移工作负载、推荐迁移策略和提供工作估算的工具。 AWS WQF 包含在 AWS Schema Conversion Tool (AWS SCT) 中。它用来分析数据库架构和代码对象、应用程序代码、依赖关系和性能特征，并提供评测报告。

B

恶意机器人

一种旨在扰乱或伤害个人或组织的机器人。

BCP

请参阅业务连续性计划。

行为图

一段时间内资源行为和交互的统一交互式视图。您可以使用 Amazon Detective 的行为图来检查失败的登录尝试、可疑的 API 调用和类似的操作。有关更多信息，请参阅 Detective 文档中的行为图中的数据。

大端序系统

一个先存储最高有效字节的系统。另请参阅字节顺序。

二进制分类

一种预测二进制结果（两个可能的类别之一）的过程。例如，您的 ML 模型可能需要预测诸如“该电子邮件是否为垃圾邮件？”或“这个产品是书还是汽车？”之类的问题

bloom 筛选条件

一种概率性、内存高效的数据结构，用于测试元素是否为集合的成员。

蓝/绿部署

一种部署策略，您可以创建两个独立但完全相同的环境。在一个环境中运行当前应用程序版本（蓝色），在另一个环境中运行新应用程序版本（绿色）。此策略可帮助您在影响最小的情况下快速回滚。

自动程序

一种通过互联网运行自动任务并模拟人类活动或交互的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的 Web 爬网程序。还有一些被称为恶意机器人的机器人，其目的是扰乱或伤害个人或组织。

僵尸网络

被恶意软件感染并受单方（称为僵尸网络控制者或僵尸网络操作者）控制的僵尸网络。僵尸网络是最著名的扩展机器人及其影响力的机制。

分支

代码存储库的一个包含区域。在存储库中创建的第一个分支是主分支。您可以从现有分支创建新分支，然后在新分支中开发功能或修复错误。为构建功能而创建的分支通常称为功能分支。当功能可以发布时，将功能分支合并回主分支。有关更多信息，请参阅关于分支（GitHub 文档）。

紧急（break-glass）访问

在特殊情况下，通过批准的流程，用户 AWS 账户 可以快速访问他们通常没有访问权限的内容。有关更多信息，请参阅 AWS Well-Architected Guidance 中的 Implement break-glass procedures 指示器。

棕地策略

您环境中的现有基础设施。在为系统架构采用棕地策略时，您需要围绕当前系统和基础设施的限制来设计架构。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

缓冲区缓存

存储最常访问的数据的内存区域。

业务能力

企业如何创造价值（例如，销售、客户服务或营销）。微服务架构和开发决策可以由业务能力驱动。有关更多信息，请参阅在 AWS上运行容器化微服务白皮书中的围绕业务能力进行组织部分。

业务连续性计划（BCP）

一项计划，旨在应对大规模迁移等破坏性事件对运营的潜在影响，并使企业能够快速恢复运营。

C

CAF

请参阅 AWS 云采用框架。

金丝雀部署

缓慢而渐进地向最终用户发布版本。当您确信无误后，即可部署新版本，并完全替换当前版本。

CCoE

请参阅云卓越中心。

CDC

请参阅更改数据捕获。

更改数据捕获（CDC）

跟踪数据来源（如数据库表）的更改并记录有关更改的元数据的过程。您可以将 CDC 用于各种目的，例如审计或复制目标系统中的更改以保持同步。

混沌工程

故意引入故障或破坏性事件来测试系统的韧性。您可以使用 AWS Fault Injection Service (AWS FIS) 来执行实验，对您的 AWS 工作负载施加压力并评估其响应。

CI/CD

请参阅持续集成和持续交付。

分类

一种有助于生成预测的分类流程。分类问题的 ML 模型预测离散值。离散值始终彼此不同。例如，一个模型可能需要评估图像中是否有汽车。

客户端加密

在目标 AWS 服务 收到数据之前，对数据进行本地加密。

云卓越中心 (CCoE)

一个多学科团队，负责推动整个组织的云采用工作，包括开发云最佳实践、调动资源、制定迁移时间表、领导组织完成大规模转型。有关更多信息，请参阅 AWS Cloud 企业战略博客上的 CCoE 帖子。

云计算

通常用于远程数据存储和 IoT 设备管理的云技术。云计算通常连接到边缘计算技术。

云运营模型

在 IT 组织中，一种用于构建、完善和优化一个或多个云环境的运营模型。有关更多信息，请参阅构建您的云运营模型。

云采用阶段

组织迁移到 AWS Cloud中时通常会经历四个阶段：

- 项目 - 出于概念验证和学习目的，开展一些与云相关的项目
- 基础 — 进行基础投资以扩大云采用率（例如，创建着陆区、定义 CCo E、建立运营模型）
- 迁移 - 迁移单个应用程序
- 重塑 - 优化产品和服务，在云中创新

Stephen Orban在 AWS Cloud 企业战略博客的博客文章《云优先之旅和采用阶段》中定义了这些阶段。有关它们与 AWS 迁移策略的关系的信息，请参阅迁移准备指南。

CMDB

请参阅配置管理数据库。

代码存储库

通过版本控制过程存储和更新源代码和其他资产（如文档、示例和脚本）的位置。常见的云存储库包括 GitHub 或 Bitbucket Cloud。每个版本的代码都称为一个分支。在微服务结构中，每个存储库都专门用于一个功能。单个 CI/CD 管线可以使用多个存储库。

冷缓存

一种空的、填充不足或包含过时或不相关数据的缓冲区缓存。这会影响性能，因为数据库实例必须从主内存或磁盘读取，这比从缓冲区缓存读取要慢。

冷数据

很少访问的数据，且通常是历史数据。查询此类数据时，通常可以接受慢速查询。将这些数据转移到性能较低且成本更低的存储层或类别可以降低成本。

计算机视觉（CV）

一种 AI 领域，它使用机器学习来分析和提取数字图像和视频等视觉格式中的信息。例如，Amazon SageMaker AI 为 CV 提供了图像处理算法。

配置偏移

对于工作负载而言，一种偏离预期状态的配置更改。这可能会导致工作负载变得不合规，且通常是渐进的，不是故意的。

配置管理数据库（CMDB）

一种存储库，用于存储和管理有关数据库及其 IT 环境的信息，包括硬件和软件组件及其配置。您通常在迁移的产品组合发现和分析阶段使用来自 CMDB 的数据。

合规性包

一系列 AWS Config 规则和补救措施，您可以汇编这些规则和补救措施，以自定义您的合规性和安全性检查。您可以使用 YAML 模板将一致性包作为单个实体部署在 AWS 账户 和区域或整个组织中。有关更多信息，请参阅 AWS Config 文档中的一致性包。

持续集成和持续交付（CI/CD）

自动执行软件发布过程的源代码、构建、测试、暂存和生产阶段的过程。 CI/CD 通常被描述为管道。 CI/CD 可以帮助您实现流程自动化、提高生产力、提高代码质量和更快地交付。有关更多信息，请参阅持续交付的优势。CD 也可以表示持续部署。有关更多信息，请参阅持续交付与持续部署。

CV

请参阅计算机视觉。

D

静态数据

网络中静止的数据，例如存储中的数据。

数据分类

根据网络中数据的关键性和敏感性对其进行识别和分类的过程。它是任何网络安全风险管理策略的关键组成部分，因为它可以帮助您确定对数据的适当保护和保留控制。数据分类是 Well-Architecte AWS d Framework 中安全支柱的一个组成部分。有关详细信息，请参阅数据分类。

数据漂移

生产数据与用来训练机器学习模型的数据之间的有意义差异，或者输入数据随时间推移的有意义变化。数据漂移可能降低机器学习模型预测的整体质量、准确性和公平性。

传输中数据

在网络中主动移动的数据，例如在网络资源之间移动的数据。

数据网格

一种架构框架，可提供分布式、去中心化的数据所有权以及集中式管理和治理。

数据最少化

仅收集并处理绝对必要数据的原则。在中进行数据最小化 AWS Cloud 可以降低隐私风险、成本和分析碳足迹。

数据边界

AWS 环境中的一组预防性防护措施，可帮助确保只有可信身份才能访问来自预期网络的可信资源。有关更多信息，请参阅在上构建数据边界。 AWS

数据预处理

将原始数据转换为 ML 模型易于解析的格式。预处理数据可能意味着删除某些列或行，并处理缺失、不一致或重复的值。

数据溯源

在数据的整个生命周期跟踪其来源和历史的过程，例如数据如何生成、传输和存储。

数据主体

正在收集和处理其数据的个人。

数据仓库

一种支持商业智能（例如分析）的数据管理系统。数据仓库通常包含大量历史数据，通常用于查询和分析。

数据库定义语言（DDL）

在数据库中创建或修改表和对象结构的语句或命令。

数据库操作语言（DML）

在数据库中修改（插入、更新和删除）信息的语句或命令。

DDL

请参阅数据库定义语言。

深度融合

组合多个深度学习模型进行预测。您可以使用深度融合来获得更准确的预测或估算预测中的不确定性。

深度学习

一个 ML 子字段使用多层人工神经网络来识别输入数据和感兴趣的目标变量之间的映射。

defense-in-depth

一种信息安全方法，经过深思熟虑，在整个计算机网络中分层实施一系列安全机制和控制措施，以保护网络及其中数据的机密性、完整性和可用性。当你采用这种策略时 AWS，你会在 AWS

Organizations 结构的不同层面添加多个控件来帮助保护资源。例如，一种 defense-in-depth方法可以结合多因素身份验证、网络分段和加密。

委派管理员

在中 AWS Organizations，兼容的服务可以注册 AWS 成员帐户来管理组织的帐户并管理该服务的权限。此账户被称为该服务的委托管理员。有关更多信息和兼容服务列表，请参阅 AWS Organizations 文档中使用 AWS Organizations的服务。

部署

使应用程序、新功能或代码修复在目标环境中可用的过程。部署涉及在代码库中实现更改，然后在应用程序的环境中构建和运行该代码库。

开发环境

请参阅环境。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出提醒。这些控制是第二道防线，提醒您注意绕过现有预防性控制的安全事件。有关更多信息，请参阅在 AWS上实施安全控制中的侦测性控制。

开发价值流映射（DVSM）

用于识别对软件开发生命周期中的速度和质量产生不利影响的限制因素并确定其优先级的流程。DVSM 扩展了最初为精益生产实践设计的价值流映射流程。其重点关注在软件开发过程中创造和转移价值所需的步骤和团队。

数字孪生

真实世界系统的虚拟再现，如建筑物、工厂、工业设备或生产线。数字孪生支持预测性维护、远程监控和生产优化。

维度表

星型架构中的一种较小的表，其中包含事实表中定量数据的数据属性。维度表属性通常是文本字段或行为类似于文本的离散数字。这些属性通常用于查询约束、筛选和结果集标注。

灾难

阻止工作负载或系统在其主要部署位置实现其业务目标的事件。这些事件可能是自然灾害、技术故障或人为操作的结果，例如无意的配置错误或恶意软件攻击。

灾难恢复（DR）

您用来最大程度地减少由灾难造成的停机时间和数据丢失的策略和流程。有关更多信息，请参阅 Well-Architected Fr ame AWS work 中的 “工作负载灾难恢复：云端 AWS 恢复”。

DML

请参阅数据库操作语言。

领域驱动设计

一种开发复杂软件系统的方法，通过将其组件连接到每个组件所服务的不断发展的领域或核心业务目标。Eric Evans 在其著作领域驱动设计：软件核心复杂性应对之道（Boston: Addison-Wesley Professional, 2003）中介绍了这一概念。有关如何将领域驱动设计与 strangler fig 模式结合使用的信息，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

DR

请参阅灾难恢复。

偏差检测

跟踪与基准配置的偏差。例如，您可以使用 AWS CloudFormation 来检测系统资源中的偏差，也可以使用 AWS Control Tower 来检测着陆区中可能影响监管要求合规性的变化。

DVSM

请参阅开发价值流映射。

E

EDA

请参阅探索性数据分析。

EDI

请参阅电子数据交换。

边缘计算

该技术可提高位于 IoT 网络边缘的智能设备的计算能力。与云计算比较时，边缘计算可以减少通信延迟并缩短响应时间。

电子数据交换（EDI）

组织之间业务文件的自动交换。有关更多信息，请参阅什么是电子数据交换。

加密

一种将人类可读的纯文本数据转换为加密文字的计算流程。

加密密钥

由加密算法生成的随机位的加密字符串。密钥的长度可能有所不同，而且每个密钥都设计为不可预测且唯一。

字节顺序

字节在计算机内存中的存储顺序。大端序系统先存储最高有效字节。小端序系统先存储最低有效字节。

端点

请参阅服务端点。

端点服务

一种可以在虚拟私有云（VPC）中托管，与其他用户共享的服务。您可以使用其他 AWS 账户 或 AWS Identity and Access Management (IAM) 委托人创建终端节点服务， AWS PrivateLink 并向其授予权限。这些账户或主体可通过创建接口 VPC 端点来私密地连接到您的端点服务。有关更多信息，请参阅 Amazon Virtual Private Cloud（Amazon VPC）文档中的创建端点服务。

企业资源规划（ERP）

一种自动化和管理企业关键业务流程（例如会计、MES 和项目管理）的系统。

信封加密

用另一个加密密钥对加密密钥进行加密的过程。有关更多信息，请参阅 AWS Key Management Service (AWS KMS) 文档中的信封加密。

环境

正在运行的应用程序的实例。以下是云计算中常见的环境类型：

- 开发环境 — 正在运行的应用程序的实例，只有负责维护应用程序的核心团队才能使用。开发环境用于测试更改，然后再将其提升到上层环境。这类环境有时称为测试环境。
- 下层环境 — 应用程序的所有开发环境，比如用于初始构建和测试的环境。

- 生产环境 — 最终用户可以访问的正在运行的应用程序的实例。在 CI/CD 管道中，生产环境是最后一个部署环境。
- 上层环境 — 除核心开发团队以外的用户可以访问的所有环境。这可能包括生产环境、预生产环境和用户验收测试环境。

epic

在敏捷方法学中，有助于组织工作和确定优先级的功能类别。epics 提供了对需求和实施任务的总体描述。例如， AWS CAF 安全史诗包括身份和访问管理、侦探控制、基础设施安全、数据保护和事件响应。有关 AWS 迁移策略中 epics 的更多信息，请参阅计划实施指南。

ERP

请参阅企业资源规划。

探索性数据分析（EDA）

分析数据集以了解其主要特征的过程。您收集或汇总数据，并进行初步调查，以发现模式、检测异常并检查假定情况。EDA 通过计算汇总统计数据和创建数据可视化得以执行。

F

事实表

星型架构中的中心表。它存储有关业务运营的定量数据。通常，事实表包含两种类型的列：包含度量的列和包含维度表外键的列。

快速失效机制

一种使用频繁且增量式的测试来缩短开发生命周期的理念。这是敏捷方法的关键部分。

故障隔离边界

在中 AWS Cloud，诸如可用区 AWS 区域、控制平面或数据平面之类的边界，它限制了故障的影响并有助于提高工作负载的弹性。有关更多信息，请参阅 AWS 故障隔离边界。

功能分支

请参阅分支。

特征

您用来进行预测的输入数据。例如，在制造环境中，特征可能是定期从生产线捕获的图像。

特征重要性

特征对于模型预测的重要性。这通常表示为数值分数，可以通过各种技术进行计算，例如 Shapley 加法解释（SHAP）和积分梯度。有关更多信息，请参阅使用机器学习模型的可解释性 AWS。

功能转换

为 ML 流程优化数据，包括使用其他来源丰富数据、扩展值或从单个数据字段中提取多组信息。这使得 ML 模型能从数据中获益。例如，如果您将“2021-05-27 00:15:37”日期分解为“2021”、“五月”、“星期四”和“15”，则可以帮助学习与不同数据成分相关的算法学习精细模式。

少样本提示

在要求 LLM 执行类似任务之前，先向其提供少量示例，以演示任务和预期输出。此技术是上下文内学习的一种应用，其中模型可以从提示中嵌入的示例（样本）中学习。对于需要特定格式、推理或领域知识的任务，少样本提示可能非常有效。另请参阅零样本提示。

FGAC

请参阅精细访问控制。

精细访问控制（FGAC）

使用多个条件允许或拒绝访问请求。

快闪迁移

一种数据库迁移方法，通过更改数据捕获使用连续数据复制，在极短的时间内迁移数据，而非使用分阶段方法。目标是将停机时间降至最低。

FM

请参阅基础模型。

基础模型（FM）

一个大型深度学习神经网络，一直在广义和未标记数据的大量数据集上进行训练。 FMs 能够执行各种各样的一般任务，例如理解语言、生成文本和图像以及用自然语言进行对话。有关更多信息，请参阅什么是基础模型。

G

生成式人工智能

AI 模型的一个子集，这些模型已经过大量数据训练，可以使用简单的文本提示来创建新的内容和构件，例如图像、视频、文本和音频。有关更多信息，请参阅什么是生成式人工智能。

地理阻止

请参阅地理限制。

地理限制（地理阻止）

在 Amazon 中 CloudFront，一种阻止特定国家/地区的用户访问内容分发的选项。您可以使用允许列表或阻止列表来指定已批准和已禁止的国家/地区。有关更多信息，请参阅 CloudFront 文档中的限制内容的地理分布。

GitFlow 工作流程

一种方法，在这种方法中，下层和上层环境在源代码存储库中使用不同的分支。Gitflow 工作流程被认为是传统的工作流程，而基于中继的工作流程则是现代的、首选的方法。

黄金映像

系统或软件的快照，用作部署该系统或软件的新实例的模板。例如，在制造业中，黄金映像可用于在多个设备上预调配软件，并有助于提高设备制造操作的速度、可扩展性和生产效率。

全新策略

在新环境中缺少现有基础设施。在对系统架构采用全新策略时，您可以选择所有新技术，而不受对现有基础设施（也称为棕地）兼容性的限制。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

防护机制

帮助管理各组织单位的资源、策略和合规性的高级规则 (OUs)。预防性防护机制会执行策略以确保符合合规性标准。它们是使用服务控制策略和 IAM 权限边界实现的。侦测性护栏会检测策略违规和合规性问题，并生成提醒以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub CSPM GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

请参阅高可用性。

异构数据库迁移

将源数据库迁移到使用不同数据库引擎的目标数据库（例如，从 Oracle 迁移到 Amazon Aurora）。异构迁移通常是重新架构工作的一部分，而转换架构可能是一项复杂的任务。AWS 提供了 AWS SCT 来帮助实现架构转换。

高可用性（HA）

在遇到挑战或灾难时，工作负载无需干预即可连续运行的能力。HA 系统旨在自动进行故障转移、持续提供良好性能，并以最小的性能影响处理不同负载和故障。

历史数据库现代化

一种用于实现运营技术（OT）系统现代化和升级以更好满足制造业需求的方法。历史数据库是一种用于收集和存储工厂中各种来源数据的数据库。

保留数据

从用于训练机器学习模型的数据集中保留的一部分标注的历史数据。通过将模型预测与保留数据进行比较，您可以使用保留数据来评估模型性能。

同构数据库迁移

将源数据库迁移到共享同一数据库引擎的目标数据库（例如，从 Microsoft SQL Server 迁移到 Amazon RDS for SQL Server）。同构迁移通常是更换主机或更换平台工作的一部分。您可以使用本机数据库实用程序来迁移架构。

热数据

经常访问的数据，例如实时数据或近期的转化数据。这些数据通常需要高性能存储层或存储类别才能提供快速的查询响应。

修补程序

针对生产环境中关键问题的紧急修复。由于其紧迫性，修补程序通常是在典型的 DevOps 发布工作流程之外进行的。

hypercare 周期

割接之后，迁移团队立即管理和监控云中迁移的应用程序以解决任何问题的时间段。通常，这个周期持续 1-4 天。在 hypercare 周期结束时，迁移团队通常会将应用程序的责任移交给云运营团队。

我

IaC

请参阅基础设施即代码。

基于身份的策略

附加到一个或多个 IAM 委托人的策略，用于定义他们在 AWS Cloud 环境中的权限。

空闲应用程序

90 天内平均 CPU 和内存使用率在 5% 到 20% 之间的应用程序。在迁移项目中，通常会停用这些应用程序或将其保留在本地。

IIoT

请参阅工业物联网。

不可变基础设施

一种模型，可为生产工作负载部署新的基础设施，而不是更新、修补或修改现有基础设施。不可变基础设施本质上比可变基础设施更一致、更可靠、更可预测。有关更多信息，请参阅 AWS Well-Architected Framework 中的使用不可变基础设施进行部署最佳实践。

入站（入口）VPC

在 AWS 多账户架构中，一种接受、检查和路由来自应用程序外部的网络连接的 VPC。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

增量迁移

一种割接策略，在这种策略中，您可以将应用程序分成小部分进行迁移，而不是一次性完整割接。例如，您最初可能只将几个微服务或用户迁移到新系统。在确认一切正常后，您可以逐步迁移其他微服务或用户，直到停用遗留系统。这种策略降低了大规模迁移带来的风险。

工业 4.0

该术语由 Klaus Schwab 在 2016 年提出，指的是通过连接、实时数据、自动化、分析和 AI/ML 的进步来实现制造流程的现代化。

基础设施

应用程序环境中包含的所有资源和资产。

基础设施即代码（IaC）

通过一组配置文件预调配和管理应用程序基础设施的过程。IaC 旨在帮助您集中管理基础设施、实现资源标准化和快速扩展，使新环境具有可重复性、可靠性和一致性。

工业物联网 (IIoT)

在工业领域使用联网的传感器和设备，例如制造业、能源、汽车、医疗保健、生命科学和农业。有关更多信息，请参阅制定工业物联网 (IIoT) 数字化转型战略。

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理对 VPCs（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量的检查。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

物联网（IoT）

由带有嵌入式传感器或处理器的连接物理对象组成的网络，这些传感器或处理器通过互联网或本地通信网络与其他设备和系统进行通信。有关更多信息，请参阅什么是 IoT？

可解释性

它是机器学习模型的一种特征，描述了人类可以理解模型的预测如何取决于其输入的程度。有关更多信息，请参阅使用机器学习模型的可解释性 AWS。

物联网

请参阅物联网。

IT 信息库（ITIL）

提供 IT 服务并使这些服务符合业务要求的一套最佳实践。ITIL 是 ITSM 的基础。

IT 服务管理（ITSM）

为组织设计、实施、管理和支持 IT 服务的相关活动。有关将云运营与 ITSM 工具集成的信息，请参阅运营集成指南。

ITIL

请参阅 IT 信息库。

ITSM

请参阅 IT 服务管理。

L

基于标签的访问控制（LBAC）

强制访问控制（MAC）的一种实施方式，其中明确为用户和数据本身分配了安全标签值。用户安全标签和数据安全标签之间的交集决定了用户可以看到哪些行和列。

登录区

landing zone 是一个架构精良的多账户 AWS 环境，具有可扩展性和安全性。这是一个起点，您的组织可以从这里放心地在安全和基础设施环境中快速启动和部署工作负载和应用程序。有关登录区的更多信息，请参阅设置安全且可扩展的多账户 AWS 环境。

大语言模型（LLM）

一种基于大量数据进行预训练的深度学习 AI 模型。LLM 可以执行多项任务，例如回答问题、总结文档、将文本翻译成其他语言以及完成句子。有关更多信息，请参阅什么是 LLMs。

大规模迁移

迁移 300 台或更多服务器。

LBAC

请参阅基于标签的访问控制。

最低权限

授予执行任务所需的最低权限的最佳安全实践。有关更多信息，请参阅 IAM 文档中的应用最低权限许可。

直接迁移

请参阅 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参阅字节顺序。

LLM

请参阅大型语言模型。

下层环境

请参阅环境。

M

机器学习（ML）

一种使用算法和技术进行模式识别和学习的人工智能。ML 对记录的数据（例如物联网（IoT）数据）进行分析和学习，以生成基于模式的统计模型。有关更多信息，请参阅机器学习。

主分支

请参阅分支。

恶意软件

旨在危害计算机安全或隐私的软件。恶意软件可能会破坏计算机系统、泄露敏感信息或获得未经授权的访问权限。恶意软件的示例包括病毒、蠕虫、勒索软件、木马、间谍软件和键盘记录器。

托管式服务

AWS 服务 它 AWS 运行基础设施层、操作系统和平台，您可以访问端点来存储和检索数据。Amazon Simple Storage Service（Amazon S3）和 Amazon DynamoDB 就是托管服务的示例。这些服务也称为抽象服务。

制造执行系统（MES）

一种软件系统，用于跟踪、监控、记录和控制在车间将原材料转化为成品的生产过程。

MAP

请参阅迁移加速计划。

机制

一个完整的过程，您可以在其中创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运作过程中自我强化和改善的循环。有关更多信息，请参阅在 Well-Architect AWS ed 框架中构建机制。

成员账户

AWS 账户 除属于组织中的管理账户之外的所有账户 AWS Organizations。一个账户一次只能是一个组织的成员。

MES

请参阅制造执行系统。

消息队列遥测传输（MQTT）

一种基于发布/订阅模式的轻量级 machine-to-machine (M2M) 通信协议，适用于资源受限的物联网设备。

微服务

一种小型的独立服务，通过明确的定义进行通信 APIs ，通常由小型的独立团队拥有。例如，保险系统可能包括映射到业务能力（如销售或营销）或子域（如购买、理赔或分析）的微服务。微服务

的好处包括敏捷、灵活扩展、易于部署、可重复使用的代码和恢复能力。有关更多信息，请参阅使用 AWS 无服务器服务集成微服务。

微服务架构

一种使用独立组件构建应用程序的方法，这些组件将每个应用程序进程作为微服务运行。这些微服务使用轻量级通过定义明确的接口进行通信。 APIs该架构中的每个微服务都可以更新、部署和扩展，以满足对应用程序特定功能的需求。有关更多信息，请参阅在上实现微服务。 AWS

迁移加速计划（MAP）

AWS 该计划提供咨询支持、培训和服务，以帮助组织为迁移到云奠定坚实的运营基础，并帮助抵消迁移的初始成本。MAP 提供了一种以系统的方式执行遗留迁移的迁移方法，以及一套用于自动执行和加速常见迁移场景的工具。

大规模迁移

将大部分应用程序组合分波迁移到云中的过程，在每一波中以更快的速度迁移更多应用程序。本阶段使用从早期阶段获得的最佳实践和经验教训，实施由团队、工具和流程组成的迁移工厂，通过自动化和敏捷交付简化工作负载的迁移。这是 AWS 迁移策略的第三阶段。

迁移工厂

跨职能团队，通过自动化、敏捷的方法简化工作负载迁移。迁移工厂团队通常包括运营、业务分析师和所有者、迁移工程师、开发人员和冲刺 DevOps 领域的专业人员。20% 到 50% 的企业应用程序组合由可通过工厂方法优化的重复模式组成。有关更多信息，请参阅本内容集中有关迁移工厂的讨论和云迁移工厂指南。

迁移元数据

有关完成迁移所需的应用程序和服务器的信息。每种迁移模式都需要一套不同的迁移元数据。迁移元数据的示例包括目标子网、安全组和 AWS 账户。

迁移模式

一种可重复的迁移任务，详细列出了迁移策略、迁移目标以及所使用的迁移应用程序或服务。示例：使用 AWS 应用程序迁移服务重新托管向 Amazon EC2 的迁移。

迁移组合评测（MPA）

一种在线工具，提供了用于验证迁移到 AWS Cloud的业务案例的信息。MPA 提供了详细的组合评测（服务器规模调整、定价、TCO 比较、迁移成本分析）以及迁移计划（应用程序数据分析和数据收集、应用程序分组、迁移优先级排序和波次规划）。所有 AWS 顾问和 APN 合作伙伴顾问均可免费使用 MPA 工具（需要登录）。

迁移准备情况评测（MRA）

使用 AWS CAF 深入了解组织的云就绪状态、确定优势和劣势以及制定行动计划以缩小已发现差距的过程。有关更多信息，请参阅迁移准备指南。MRA 是 AWS 迁移策略的第一阶段。

迁移策略

将工作负载迁移到 AWS Cloud的方法。有关更多信息，请参见术语表中的 7 R 词条，以及动员您的组织以加快大规模迁移。

ML

请参阅机器学习。

现代化

将过时的（原有的或单体）应用程序及其基础设施转变为云中敏捷、弹性和高度可用的系统，以降低成本、提高效率和利用创新。有关更多信息，请参阅在 AWS Cloud中实现应用程序现代化的策略。

现代化准备情况评估

一种评估方式，有助于确定组织应用程序的现代化准备情况；确定收益、风险和依赖关系；确定组织能够在多大程度上支持这些应用程序的未来状态。评估结果是目标架构的蓝图、详细说明现代化进程发展阶段和里程碑的路线图以及解决已发现差距的行动计划。有关更多信息，请参阅在 AWS Cloud中评估应用程序的现代化准备情况。

单体应用程序（单体式）

作为具有紧密耦合进程的单个服务运行的应用程序。单体应用程序有几个缺点。如果某个应用程序功能的需求激增，则必须扩展整个架构。随着代码库的增长，添加或改进单体应用程序的功能也会变得更加复杂。若要解决这些问题，可以使用微服务架构。有关更多信息，请参阅将单体分解为微服务。

MPA

请参阅迁移组合评测。

MQTT

请参阅消息队列遥测传输。

多分类器

一种帮助为多个类别生成预测（预测两个以上结果之一）的过程。例如，ML 模型可能会询问“这个产品是书、汽车还是手机？”或“此客户最感兴趣什么类别的产品？”

可变基础设施

一种用于更新和修改生产工作负载的现有基础设施的模型。为了提高一致性、可靠性和可预测性，Well-Architect AWS ed Framework 建议使用不可变基础设施作为最佳实践。

O

OAC

请参阅来源访问控制。

OAI

请参阅来源访问身份。

OCM

请参阅组织变革管理。

离线迁移

一种迁移方法，在这种方法中，源工作负载会在迁移过程中停止运行。这种方法会延长停机时间，通常用于小型非关键工作负载。

OI

请参阅运营集成。

OLA

请参阅运营级别协议。

在线迁移

一种迁移方法，在这种方法中，源工作负载无需离线即可复制到目标系统。在迁移过程中，连接工作负载的应用程序可以继续运行。这种方法的停机时间为零或最短，通常用于关键生产工作负载。

OPC-UA

请参阅开放流程通信 – 统一架构。

开放流程通信 – 统一架构（OPC-UA）

一种用于工业自动化的 machine-to-machine（M2M）通信协议。OPC-UA 提供了一个包含数据加密、身份验证和授权方案的互操作性标准。

运营级别协议（OLA）

一项协议，阐明了 IT 职能部门承诺相互交付的内容，以支持服务水平协议（SLA）。

运营准备情况审查（ORR）

一份问题核对清单和关联的最佳实践，可帮助您了解、评估、预防或缩小事件和可能的故障的范围。有关更多信息，请参阅 AWS Well-Architected Framework 中的运营准备情况审查（ORR）。

运营技术（OT）

与物理环境配合使用以控制工业运营、设备和基础设施的硬件和软件系统。在制造业中，OT 和信息技术（IT）系统的集成是工业 4.0 转型的关键重点。

运营整合（OI）

在云中实现运营现代化的过程，包括就绪计划、自动化和集成。有关更多信息，请参阅运营整合指南。

组织跟踪

由 AWS CloudTrail 此创建的跟踪记录组织 AWS 账户 中所有人的所有事件 AWS Organizations。该跟踪是在每个 AWS 账户 中创建的，属于组织的一部分，并跟踪每个账户的活动。有关更多信息，请参阅 CloudTrail文档中的为组织创建跟踪。

组织变革管理（OCM）

一个从人员、文化和领导力角度管理重大、颠覆性业务转型的框架。OCM 通过加快变革采用、解决过渡问题以及推动文化和组织变革，帮助组织为新系统和战略做好准备和过渡。在 AWS 迁移策略中，该框架被称为人员加速，因为云采用项目需要变更的速度。有关更多信息，请参阅 OCM 指南。

来源访问控制（OAC）

在中 CloudFront，一个增强的选项，用于限制访问以保护您的亚马逊简单存储服务 (Amazon S3) 内容。OAC 全部支持所有 S3 存储桶 AWS 区域、使用 AWS KMS (SSE-KMS) 进行服务器端加密，以及对 S3 存储桶的动态PUT和DELETE请求。

来源访问身份（OAI）

在中 CloudFront，一个用于限制访问权限以保护您的 Amazon S3 内容的选项。当您使用 OAI 时，CloudFront 会创建一个 Amazon S3 可以对其进行身份验证的委托人。经过身份验证的委托人只能通过特定 CloudFront 分配访问 S3 存储桶中的内容。另请参阅 OAC，其中提供了更精细和增强的访问控制。

ORR

请参阅运营准备情况审查。

OT

请参阅运营技术。

出站（出口）VPC

在 AWS 多账户架构中，一种处理从应用程序内部启动的网络连接的 VPC。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

P

权限边界

附加到 IAM 主体的 IAM 管理策略，用于设置用户或角色可以拥有的最大权限。有关更多信息，请参阅 IAM 文档中的权限边界。

个人身份信息（PII）

直接查看其他相关数据或与之配对时可用于合理推断个人身份的信息。PII 的示例包括姓名、地址和联系信息。

PII

请参阅个人身份信息。

playbook

一套预定义的步骤，用于捕获与迁移相关的工作，例如在云中交付核心运营功能。playbook 可以采用脚本、自动化运行手册的形式，也可以是操作现代化环境所需的流程或步骤的摘要。

PLC

请参阅可编程逻辑控制器。

PLM

请参阅产品生命周期管理。

policy

一个对象，可以定义权限（请参阅基于身份的策略）、指定访问条件（请参阅基于资源的策略）或定义 AWS Organizations 的组织中所有账户的最大权限（请参阅服务控制策略）。

多语言持久性

根据数据访问模式和其他要求，独立选择微服务的数据存储技术。如果您的微服务采用相同的数据存储技术，它们可能会遇到实现难题或性能不佳。如果微服务使用最适合其需求的数据存储，则可以更轻松地实现微服务，并获得更好的性能和可扩展性。

组合评测

一个发现、分析和确定应用程序组合优先级以规划迁移的过程。有关更多信息，请参阅评估迁移准备情况。

谓词

返回 `true` 或 `false` 的查询条件，通常位于 `WHERE` 子句中。

谓词下推

一种数据库查询优化技术，可在传输之前筛选查询中的数据。这将减少从关系数据库检索和处理的数据量，并提高查询性能。

预防性控制

一种安全控制，旨在防止事件发生。这些控制是第一道防线，帮助防止未经授权的访问或对网络的意外更改。有关更多信息，请参阅在 AWS上实施安全控制中的预防性控制。

主体

中 AWS 可以执行操作和访问资源的实体。此实体通常是 IAM 角色的根用户或用户。 AWS 账户有关更多信息，请参阅 IAM 文档中角色术语和概念中的主体。

隐私设计

一种在整个开发过程中都考虑隐私的系统工程方法。

私有托管区

一个容器，其中包含有关您希望 Amazon Route 53 如何响应针对一个或多个 VPCs域名及其子域名的 DNS 查询的信息。有关更多信息，请参阅 Route 53 文档中的私有托管区的使用。

主动控制

一种安全控制，旨在防止部署不合规资源。这些控制会在资源预置之前对其进行扫描。如果资源与控制不兼容，则不会预置它。有关更多信息，请参阅 AWS Control Tower 文档中的控制参考指南，并参见在上实施安全控制中的主动控制 AWS。

产品生命周期管理（PLM）

对产品在其整个生命周期内的数据和流程的管理，从设计、开发和发布，到增长和成熟，再到衰退和淘汰。

生产环境

请参阅环境。

可编程逻辑控制器（PLC）

在制造业中，一种高度可靠、适应性强的计算机，用于监控机器并实现制造过程自动化。

提示串接

使用一个 LLM 提示的输出作为下一个提示的输入，以生成更好的响应。该技术用于将复杂的任务分解为子任务，或者迭代地完善或扩展初步响应。它有助于提高模型响应的准确性和相关性，并允许获得更精细的个性化结果。

假名化

用占位符值替换数据集中个人标识符的过程。假名化可以帮助保护个人隐私。假名化数据仍被视为个人数据。

publish/subscribe (pub/sub)

一种支持微服务间异步通信的模式，可提高可扩展性和响应能力。例如，在基于微服务的 MES 中，微服务可以将事件消息发布到其他微服务可以订阅的频道。系统可以在不更改发布服务的情况下添加新的微服务。

Q

查询计划

一系列用于访问 SQL 关系数据库系统中的数据的步骤，类似于指令。

查询计划回归

当数据库服务优化程序选择的最佳计划不如数据库环境发生特定变化之前时。这可能是由统计数据、约束、环境设置、查询参数绑定更改和数据库引擎更新造成的。

R

RACI 矩阵

请参阅责任、问责、咨询和知情（RACI）。

RAG

请参阅检索增强生成。

勒索软件

一种恶意软件，旨在阻止对计算机系统或数据的访问，直到付款为止。

RASCI 矩阵

请参阅责任、问责、咨询和知情（RACI）。

RCAC

请参阅行列访问控制。

只读副本

用于只读目的的数据库副本。您可以将查询路由到只读副本，以减轻主数据库的负载。

重新架构

请参阅 7 R。

恢复点目标（RPO）

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。

恢复时间目标（RTO）

服务中断和服务恢复之间可接受的最大延迟。

重构

请参阅 7 R。

Region

地理区域内的 AWS 资源集合。每一个 AWS 区域 都相互隔离，相互独立，以提供容错、稳定性和弹性。有关更多信息，请参阅指定您的账户可以使用的 AWS 区域。

回归

一种预测数值的 ML 技术。例如，要解决“这套房子的售价是多少？”的问题 ML 模型可以使用线性回归模型，根据房屋的已知事实（如建筑面积）来预测房屋的销售价格。

重新托管

请参阅 7 R。

版本

在部署过程中，推动生产环境变更的行为。

重新放置

请参阅 7 R。

更换平台

请参阅 7 R。

重新购买

请参阅 7 R。

韧性

应用程序抵御中断或从中断中恢复的能力。在 AWS Cloud中规划韧性时，高可用性和灾难恢复是常见的考虑因素。有关更多信息，请参阅 AWS Cloud 韧性。

基于资源的策略

一种附加到资源的策略，例如 AmazonS3 存储桶、端点或加密密钥。此类策略指定了允许哪些主体访问、支持的操作以及必须满足的任何其他条件。

责任、问责、咨询和知情（RACI）矩阵

定义参与迁移活动和云运营的所有各方的角色和责任的矩阵。矩阵名称源自矩阵中定义的责任类型：负责（R）、问责（A）、咨询（C）和知情（I）。支持（S）类型是可选的。如果包括支持，则该矩阵称为 RASCI 矩阵，如果将其排除在外，则称为 RACI 矩阵。

响应性控制

一种安全控制，旨在推动对不良事件或偏离安全基线的情况进行修复。有关更多信息，请参阅在AWS上实施安全控制中的响应性控制。

保留

请参阅 7 R。

停用

请参阅 7 R。

检索增强生成（RAG）

一种生成式人工智能技术，其中 LLM 在生成响应之前引用其训练数据来源之外的权威数据来源。例如，RAG 模型可以对组织的知识库或自定义数据执行语义搜索。有关更多信息，请参阅什么是 RAG。

轮换

定期更新密钥以使攻击者更难访问凭证的过程。

行列访问控制（RCAC）

使用已定义访问规则的基本、灵活的 SQL 表达式。RCAC 由行权限和列掩码组成。

RPO

请参阅恢复点目标。

RTO

请参阅恢复时间目标。

运行手册

执行特定任务所需的一套手动或自动程序。它们通常是为了简化重复性操作或高错误率的程序而设计的。

S

SAML 2.0

许多身份提供商 (IdPs) 使用的开放标准。此功能支持联合单点登录 (SSO)，因此用户无需在 IAM 中为组织中的所有人创建用户即可登录 AWS 管理控制台 或调用 AWS API 操作。有关基于 SAML 2.0 的联合身份验证的更多信息，请参阅 IAM 文档中的关于基于 SAML 2.0 的联合身份验证。

SCADA

请参阅监督控制和数据采集。

SCP

请参阅服务控制策略。

机密密钥

在中 AWS Secrets Manager，您以加密形式存储的机密或受限信息，例如密码或用户凭证。它由密钥值及其元数据组成。密钥值可以是二进制、单个字符串或多个字符串。有关更多信息，请参阅 Secrets Manager 文档中的什么是 Amazon Secrets Manager 密钥？。

安全设计

一种在整个开发过程中都考虑安全的系统工程方法。

安全控制

一种技术或管理防护机制，可防止、检测或降低威胁行为体利用安全漏洞的能力。安全控制有以下四种类型：预防性、检测性、响应性和主动性。

安全固化

缩小攻击面，使其更能抵御攻击的过程。这可能包括删除不再需要的资源、实施授予最低权限的最佳安全实践或停用配置文件中不必要的功能等操作。

安全信息和事件管理（SIEM）系统

结合了安全信息管理（SIM）和安全事件管理（SEM）系统的工具和服务。SIEM 系统会收集、监控和分析来自服务器、网络、设备和其他来源的数据，以检测威胁和安全漏洞，并生成警报。

安全响应自动化

一种预定义的程序化操作，旨在自动响应或修复安全事件。这些自动化可作为侦探或响应式安全控制措施，帮助您实施 AWS 安全最佳实践。自动响应操作的示例包括修改 VPC 安全组、修补 Amazon EC2 实例或轮换凭证。

服务器端加密

由接收数据的人在目的地对数据 AWS 服务 进行加密。

服务控制策略（SCP）

一种策略，用于集中控制组织中所有账户的权限 AWS Organizations。 SCPs 定义防护措施或限制管理员可以委托给用户或角色的操作。您可以使用 SCPs 允许列表或拒绝列表来指定允许或禁止哪些服务或操作。有关更多信息，请参阅 AWS Organizations 文档中的服务控制策略。

服务端点

的入口点的 URL AWS 服务。您可以使用端点，通过编程方式连接到目标服务。有关更多信息，请参阅 AWS 一般参考 中的 AWS 服务 端点。

服务水平协议（SLA）

一份协议，阐明了 IT 团队承诺向客户交付的内容，比如服务正常运行时间和性能。

服务水平指示器（SLI）

对服务性能方面的衡量，例如错误率、可用性或吞吐量。

服务水平目标（SLO）

代表服务运行状况的目标指标，由服务水平指示器衡量。

责任共担模式

描述您在云安全与合规方面共同承担 AWS 的责任的模型。 AWS 负责云的安全，而您则负责云中的安全。有关更多信息，请参阅责任共担模式。

SIEM

请参阅安全信息和事件管理系统。

单点故障（SPOF）

应用程序的单个关键组件出现故障，可能会中断系统。

SLA

请参阅服务水平协议。

SLI

请参阅服务水平指示器。

SLO

请参阅服务水平目标。

split-and-seed 模型

一种扩展和加速现代化项目的模式。随着新功能和产品发布的定义，核心团队会拆分以创建新的产品团队。这有助于扩展组织的能力和服务，提高开发人员的工作效率，支持快速创新。有关更多信息，请参阅在 AWS Cloud中实现应用程序现代化的分阶段方法。

SPOF

请参阅单点故障。

星型架构

一种数据库组织结构，它使用一个大型事实表来存储事务数据或测量数据，并使用一个或多个较小的维度表来存储数据属性。此结构专为在数据仓库中使用或用于商业智能目的而设计。

strangler fig 模式

一种通过逐步重写和替换系统功能直至可以停用原有的系统来实现单体系统现代化的方法。这种模式用无花果藤作为类比，这种藤蔓成长为一棵树，最终战胜并取代了宿主。该模式是由 Martin Fowler 提出的，作为重写单体系统时管理风险的一种方法。有关如何应用此模式的示例，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

子网

您的 VPC 内的一个 IP 地址范围。子网必须位于单个可用区中。

监督控制和数据采集（SCADA）

在制造业中，一种使用硬件和软件来监控实物资产和生产操作的系统。

对称加密

一种加密算法，它使用相同的密钥来加密和解密数据。

综合测试

以模拟用户交互的方式测试系统，以检测潜在问题或监控性能。你可以使用 Amazon S CloudWatch ynthetics 来创建这些测试。

系统提示

一种为 LLM 提供上下文、说明或准则以指导其行为的技术。系统提示有助于设置上下文并制定与用户交互的规则。

T

标签

键值对，用作组织资源的元数据。 AWS 标签有助于您管理、识别、组织、搜索和筛选 资源。有关更多信息，请参阅标记您的 AWS 资源。

目标变量

您在监督式 ML 中尝试预测的值。这也被称为结果变量。例如，在制造环境中，目标变量可能是产品缺陷。

任务列表

一种通过运行手册用于跟踪进度的工具。任务列表包含运行手册的概述和要完成的常规任务列表。对于每项常规任务，它包括预计所需时间、所有者和进度。

测试环境

请参阅环境。

训练

为您的 ML 模型提供学习数据。训练数据必须包含正确答案。学习算法在训练数据中查找将输入数据属性映射到目标（您希望预测的答案）的模式。然后输出捕获这些模式的 ML 模型。然后，您可以使用 ML 模型对不知道目标的新数据进行预测。

中转网关

一个网络传输中心，可用于将您的网络 VPCs 和本地网络互连。有关更多信息，请参阅 AWS Transit Gateway 文档中的什么是公交网关。

基于中继的工作流程

一种方法，开发人员在功能分支中本地构建和测试功能，然后将这些更改合并到主分支中。然后，按顺序将主分支构建到开发、预生产和生产环境。

可信访问权限

向您指定的服务授予权限，该服务可代表您在其账户中执行任务。 AWS Organizations 当需要服务相关的角色时，受信任的服务会在每个账户中创建一个角色，为您执行管理任务。有关更多信息，请参阅 AWS Organizations 文档中的AWS Organizations 与其他 AWS 服务一起使用。

优化

更改训练过程的各个方面，以提高 ML 模型的准确性。例如，您可以通过生成标签集、添加标签，并在不同的设置下多次重复这些步骤来优化模型，从而训练 ML 模型。

双披萨团队

一个小 DevOps 团队，你可以用两个披萨来喂食。双披萨团队的规模可确保在软件开发过程中充分协作。

U

不确定性

这一概念指的是不精确、不完整或未知的信息，这些信息可能会破坏预测式 ML 模型的可靠性。不确定性有两种类型：认知不确定性是由有限的、不完整的数据造成的，而偶然不确定性是由数据中固有的噪声和随机性导致的。有关更多信息，请参阅量化深度学习系统中的不确定性指南。

无差别任务

也称为繁重工作，即创建和运行应用程序所必需的工作，但不能为最终用户提供直接价值或竞争优势。无差别任务的示例包括采购、维护和容量规划。

上层环境

请参阅环境。

V

vacuum 操作

一种数据库维护操作，包括在增量更新后进行清理，以回收存储空间并提高性能。

版本控制

跟踪更改的过程和工具，例如存储库中源代码的更改。

VPC 对等连接

两者之间的连接 VPCs ，允许您使用私有 IP 地址路由流量。有关更多信息，请参阅 Amazon VPC 文档中的什么是 VPC 对等连接。

漏洞

损害系统安全的软件缺陷或硬件缺陷。

W

热缓存

一种包含经常访问的当前相关数据的缓冲区缓存。数据库实例可以从缓冲区缓存读取，这比从主内存或磁盘读取要快。

暖数据

不常访问的数据。查询此类数据时，通常可以接受中速查询。

窗口函数

一种对与当前记录有某种关联的一组行执行计算的 SQL 函数。窗口函数对于处理任务很有用，例如计算移动平均值或根据当前行的相对位置访问行的值。

工作负载

一系列资源和代码，它们可以提供商业价值，如面向客户的应用程序或后端过程。

工作流

迁移项目中负责一组特定任务的职能小组。每个工作流都是独立的，但支持项目中的其他工作流。例如，组合工作流负责确定应用程序的优先级、波次规划和收集迁移元数据。组合工作流将这些资产交付给迁移工作流，然后迁移服务器和应用程序。

WORM

请参阅一次写入多次读取。

WQF

请参阅 AWS 工作负载资格鉴定框架。

一次写入多次读取（WORM）

一种存储模型，可一次写入数据并防止数据被删除或修改。授权用户可以根据需要多次读取数据，但无法对其进行更改。此数据存储基础设施被认为不可变。

Z

零日漏洞利用

一种利用零日漏洞的攻击，通常为恶意软件。

零日漏洞

生产系统中不可避免的缺陷或漏洞。威胁主体可能利用这种类型的漏洞攻击系统。开发人员经常因攻击而意识到该漏洞。

零样本提示

为 LLM 提供执行任务的说明，但没有可以帮助指导的示例（样本）。LLM 必须使用预先训练的知识来处理任务。零样本提示的有效性取决于任务的复杂性和提示的质量。另请参阅少样本提示。

僵尸应用程序

平均 CPU 和内存使用率低于 5% 的应用程序。在迁移项目中，通常会停用这些应用程序。

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