在上构建六角形架构 AWS

AWS 规范性指导

AWS 规范性指导: 在上构建六角形架构 AWS

Table of Contents

在上构建六角形架构 AWS

Furkan Oruc、Dominik Goby、Darius Kunce 和 Amazon Web Services 的 Michal Ploski ()AWS

2022 年 6 月（文档历史记录）

本指南描述了用于开发软件架构的心理模型和一系列模式。随着产品采用率的增长，这些架构易于在整个组织中维护、扩展和扩展。诸如 Amazon Web Services (AWS) 之类的云超大规模企业为小型和大型企业提供了创新和开发新软件产品的基石。这些新服务和功能的快速引入使业务利益相关者期望他们的开发团队更快地对新的最低可行产品 (MVPs) 进行原型设计，以便可以尽快测试和验证新的想法。通常， MVPs 它们会被采用并成为企业软件生态系统的一部分。在制作这些规则的过程中 MVPs，团队有时会放弃软件开发规则和最佳实践，例如SOLID原则和单元测试。他们认为这种方法将加快开发速度并缩短上市时间。但是，如果他们未能为所有级别的软件架构创建基础模型和框架，则很难甚至不可能为产品开发新功能。在开发过程中，缺乏确定性和不断变化的要求也会减慢团队的工作速度。

本指南介绍了拟议的软件架构，从低级六边形架构到高级架构和组织分解，该架构使用域驱动设计 (DDD) 来应对这些挑战。随着新功能的开发，DDD 可帮助管理业务复杂性并扩大工程团队规模。它使用无处不在的语言，使业务和技术利益相关者与业务问题（称为域名）保持一致。Hexagonal 架构是一个非常具体的领域（称为有界上下文）中这种方法的技术推动力。有限的上下文是业务问题中一个高度凝聚力和松散耦合的子领域。我们建议您对所有企业软件项目采用六边形架构，无论其复杂性如何。

Hexagonal 架构鼓励工程团队首先解决业务问题，而传统的分层架构则将工程重点从领域转移到首先解决技术问题上。此外，如果软件遵循六边形架构，则更容易采用测试驱动的开发方法，从而减少开发人员测试业务需求所需的反馈循环。最后，使用命令和命令处理程序是应用 SOLID 的单一责任和开放式封闭原则的一种方式。遵循这些原则可以生成一个代码库，开发人员和架构师可以轻松浏览和理解该项目，并降低对现有功能进行重大更改的风险。

本指南适用于有兴趣了解在软件开发项目中采用六角形架构和 DDD 的好处的软件架构师和开发人员。它包括一个为应用程序设计支持六角形架构的基础架构 AWS 的示例。有关实现示例，请参阅 AWS Prescriptive Guidence AWS Lambda网站上使用在六角形架构中构建 Python 项目。

概述

域驱动型设计 (DDD)

在域驱动设计 (DDD) 中，域是软件系统的核心。在开发任何其他模块之前，首先定义域模型，它不依赖于其他低级模块。相反，诸如数据库、表示层和外部之类的模块 APIs 都依赖于域。

在 DDD 中，架构师使用基于业务逻辑的分解而不是技术分解，将解决方案分解为有限的上下文。本目标业务成果节讨论了这种方法的好处。

当团队使用六角形架构时，DDD 更容易实现。在六角形架构中，应用程序核心是应用程序的中心。它通过端口和适配器与其他模块分离，并且不依赖于其他模块。这与 DDD 完全一致，其中，域是解决业务问题的应用程序的核心。本指南提出了一种方法，将六角形架构的核心建模为有界上下文的域模型。下一节将更详细地介绍六角形架构。

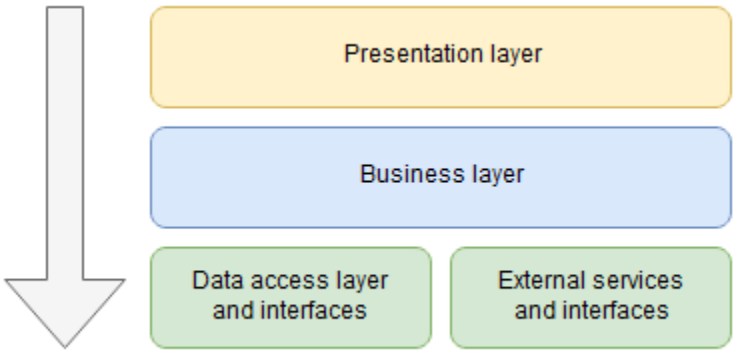
本指南并未涵盖DDD的所有方面，这是一个非常广泛的话题。为了更好地理解，您可以查看域名语言网站上列出的DDD资源。

六角形架构

Hexagonal 架构，也称为端口和适配器或洋葱架构，是管理软件项目中依赖关系反转的原理。Hexagonal 架构促进在开发软件时高度关注核心领域的业务逻辑，并将外部集成点视为次要集成点。Hexagonal 架构可帮助软件工程师采用诸如测试驱动开发 (TDD) 之类的良好实践，这反过来又可以促进架构的演进，并帮助您长期管理复杂的领域。

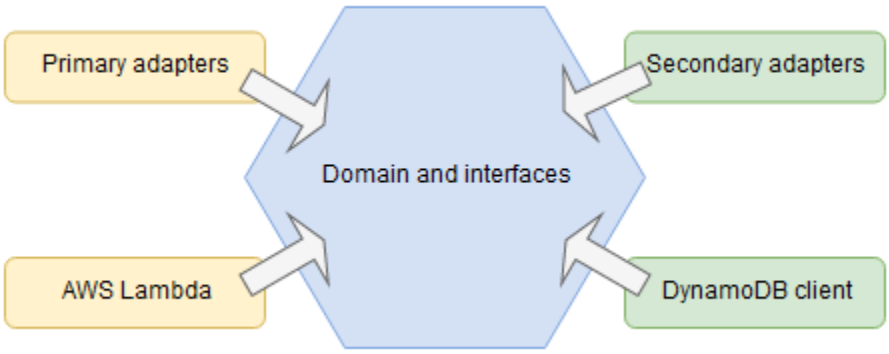
让我们比较一下六角形架构和经典的分层架构，后者是建模结构化软件项目的最常用选择。这两种方法之间存在微妙但很大的区别。

在分层架构中，软件项目是分层结构的，这代表了广泛的关注点，例如业务逻辑或演示逻辑。这种架构使用依赖关系层次结构，其中顶层依赖于其下方的层，但不相反。在下图中，表示层负责用户交互，因此它包括用户界面、 APIs、命令行界面和类似的组件。表示层依赖于实现域逻辑的业务层。反过来，业务层依赖于数据访问层和多个外部服务。



这种配置的主要缺点是依赖结构。例如，如果在数据库中存储数据的模型发生变化，则会影响数据访问接口。对数据模型的任何更改也会影响业务层，业务层依赖于数据访问接口。因此，软件工程师无法在不影响域逻辑的情况下对基础架构进行任何更改。这反过来又增加了出现回归错误的可能性。

六角形架构以不同的方式定义依赖关系，如下图所示。它围绕定义所有接口的域业务逻辑进行决策。外部组件通过称为端口的接口与业务逻辑交互。端口是定义域与外部世界交互的抽象概念。每个基础架构组件都必须实现这些端口，因此这些组件的更改不再影响核心域逻辑。



周围的组件称为适配器。适配器是外部世界和内部世界之间的代理，它实现了域中定义的端口。适配器可以分为两组：主适配器和辅助适配器。主适配器是软件组件的入口点。它们允许外部参与者、用户和服务与核心逻辑进行交互。 AWS Lambda 是主适配器的一个很好的例子。它与多个 AWS 服务集成，这些服务可以调用 Lambda 函数作为入口点。辅助适配器是处理与外部世界通信的外部服务库封装器。辅助适配器的一个很好的例子是用于数据访问的 Amazon DynamoDB 客户端。

目标业务成果

本指南中讨论的六角形架构可帮助您实现以下目标：

- 通过改善开发周期来缩短上市时间

- 提高软件质量
- 更轻松地适应变化

以下各节将详细讨论这些过程。

改善开发周期

云端开发软件给软件工程师带来了新的挑战，因为在开发计算机上本地复制运行时环境非常困难。验证软件的一种直接方法是将其部署到云端并在那里进行测试。但是，这种方法涉及漫长的反馈周期，尤其是当软件架构包含多个无服务器部署时。改善这种反馈周期可以缩短开发功能的时间，从而大大缩短上市时间。

在云端进行测试

直接在云端进行测试是确保您的架构组件（例如 Amazon API Gateway 中的网关、 AWS Lambda 函数、Amazon DynamoDB 表 AWS Identity and Access Management 和 (IAM) 权限）正确配置的唯一方法。它也可能是测试组件集成的唯一可靠方法。尽管有些 AWS 服务（例如 D ynam oDB）可以在本地部署，但大多数服务无法在本地设置中复制。同时，第三方工具（例如 Moto）和用于测试目的LocalStack的模拟 AWS 服务可能无法准确反映真实的服务API合同，或者功能的数量可能有限。

但是，企业软件中最复杂的部分在于业务逻辑，而不是云架构。架构的更改频率低于域名，因为域必须适应新的业务需求。因此，在云端测试业务逻辑成为一个紧张的过程，包括更改代码、启动部署、等待环境准备就绪并验证更改。如果部署只需 5 分钟，则在业务逻辑中进行和测试 10 项更改将需要一个小时或更长时间。如果业务逻辑更复杂，则测试可能需要等待几天的时间才能完成部署。如果您的团队中有多个功能和工程师，那么延长的期限很快就会引起业务的注意。

本地测试

六角形架构可以帮助开发人员专注于领域，而不是基础架构的技术细节。这种方法使用本地测试（您选择的开发框架中的单元测试工具）来满足域逻辑要求。您不必花时间解决技术集成问题或将软件部署到云端来测试业务逻辑。你可以在本地运行单元测试，并将反馈回路从几分钟缩短到几秒钟。如果部署需要 5 分钟，但单元测试在 5 秒钟内完成，那么检测错误所需的时间就会大大缩短。本指南后面的首先测试业务逻辑部分将更详细地介绍这种方法。

开发的并行化

六角形架构方法使开发团队能够并行完成开发工作。开发人员可以单独设计和实现服务的不同组件。这种并行化是通过隔离每个组件以及每个组件之间定义的接口来实现的。

产品上市时间

如前所述，本地单元测试可以改善开发反馈周期，缩短新产品或功能的上市时间，尤其是当这些产品或功能包含复杂的业务逻辑时。此外，通过单元测试增加代码覆盖率可以显著降低更新或重构代码库时引入回归错误的风险。单元测试覆盖率还使您能够持续重构代码库以使其井井有条，从而加快新工程师的入职流程。这一点将在本质量源于设计节中进一步讨论。最后，如果业务逻辑经过良好的隔离和测试，它可以使开发人员快速适应不断变化的功能和非功能需求。本适应变化节将对此进行进一步解释。

质量源于设计

采用六边形架构有助于从项目一开始就提高代码库的质量。重要的是要建立一个从一开始就帮助您满足预期质量要求的流程，同时又不会减慢开发过程。

本地化更改和更高的可读性

使用六边形架构方法，开发人员可以在不影响其他类或组件的情况下更改一个类或组件中的代码。这种设计促进了已开发组件的凝聚力。通过将域与适配器解耦并使用众所周知的接口，可以提高代码的可读性。识别问题和极端案例变得更加容易。

这种方法还可以促进开发过程中的代码审查，并限制引入未被发现的更改或技术债务。

首先测试业务逻辑

本地测试可以通过向项目引入 end-to-end、集成和单元测试来完成。 End-to-end测试涵盖整个传入请求生命周期。他们通常会调用应用程序入口点并进行测试，以查看其是否已满足业务需求。每个软件项目应至少有一个使用已知输入并产生预期输出的测试方案。但是，添加更多的极端情况可能会变得复杂，因为必须将每个测试配置为通过入口点（例如，通过 REST API 或队列）发送请求，遍历业务操作所需的所有集成点，然后断言结果。为测试场景设置环境并断言结果可能需要开发人员花费大量时间。

在六角形架构中，您可以隔离测试业务逻辑，并使用集成测试来测试辅助适配器。你可以在业务逻辑测试中使用模拟或假适配器。您还可以将业务用例测试与域模型的单元测试相结合，以保持高覆盖率和低耦合。作为一种好的做法，集成测试不应验证业务逻辑。相反，他们应该验证辅助适配器是否正确调用了外部服务。

理想情况下，您可以使用测试驱动开发 (TDD)，并在开发之初通过适当的测试开始定义领域实体或业务用例。首先编写测试可以帮助您创建域所需接口的模拟实现。当测试成功并且满足域逻辑规则时，您可以实现实际的适配器并将软件部署到测试环境。此时，您对域逻辑的实现可能并不理想。然后，您可以通过引入设计模式或重新排列一般代码来重构现有架构以对其进行改进。通过使用这种方法，您可以避免引入回归错误，并且可以随着项目的发展改进架构。通过将这种方法与您在持续集成过程中运行的自动测试相结合，可以在潜在错误投入生产之前减少其数量。

如果您使用无服务器部署，则可以在您的 AWS 账户中快速配置应用程序实例以进行手动集成和 end-to-end测试。完成这些实施步骤后，我们建议您对推送到存储库的每个新更改进行自动测试。

可维护性

可维护性是指操作和监控应用程序，以确保其满足所有要求并最大限度地降低系统故障的可能性。要使系统可运行，必须使其适应未来的流量或运营需求。您还必须确保它可用且易于部署，同时尽量减少或不影响客户端。

要了解系统的当前和历史状态，必须使其可观察。为此，您可以提供特定的指标、日志和跟踪，操作员可以使用这些指标、日志和跟踪来确保系统按预期运行并跟踪错误。这些机制还应允许操作员进行根本原因分析，而不必登录机器并阅读代码。

六边形架构旨在提高 Web 应用程序的可维护性，从而减少代码所需的总体工作量。通过分离模块、本地化更改以及将应用程序业务逻辑与适配器实现分离，您可以生成指标和日志，帮助操作员深入了解系统并了解对主适配器或辅助适配器所做的特定更改的范围。

适应变化

软件系统往往会变得复杂。造成这种情况的原因之一可能是业务需求经常发生变化，几乎没有时间相应地调整软件架构。另一个原因可能是投资不足，无法在项目开始时设置软件架构，以适应频繁的变化。不管是什么原因，软件系统都可能变得复杂，以至于几乎不可能做出改变。因此，从项目一开始就构建可维护的软件架构非常重要。良好的软件架构可以轻松适应变化。

本节介绍如何使用可轻松适应非功能或业务需求的六角形架构来设计可维护的应用程序。

使用端口和适配器适应新的非功能要求

作为应用程序的核心，域模型定义了满足业务需求所需的外部世界操作。这些操作是通过抽象定义的，这些抽象被称为端口。这些端口由单独的适配器实现。每个适配器负责与其他系统的交互。例如，您可能有一个适配器用于数据库存储库，另一个适配器用于与第三方 API 交互。该域不知道适配器的实现，因此很容易将一个适配器替换为另一个适配器。例如，应用程序可能会从 SQL 数据库切换到 NoSQL 数据库。在这种情况下，必须开发一个新的适配器来实现由域模型定义的端口。该域不依赖于数据库存储库，而是使用抽象进行交互，因此无需在域模型中进行任何更改。因此，六角形架构可以轻松适应非功能性需求。

通过使用命令和命令处理程序来适应新的业务需求

在经典的分层架构中，域取决于持久层。如果要更改域，则还必须更改持久层。相比之下，在六角形架构中，该域不依赖于软件中的其他模块。域是应用程序的核心，所有其他模块（端口和适配器）都取决于域模型。该域使用依赖反转原理通过端口与外界通信。依赖关系反转的好处是，你可以自由地更改域模型，而不必害怕破坏代码的其他部分。由于域名模型反映了您要解决的业务问题，因此更新域模型以适应不断变化的业务需求不是问题。

在开发软件时，关注点分离是需要遵循的重要原则。要实现这种分离，你可以使用稍微修改过的命令模式。这是一种行为设计模式，其中完成操作所需的所有信息都封装在命令对象中。然后，这些操作由命令处理程序处理。命令处理程序是接收命令、更改域状态然后向调用者返回响应的方法。您可以使用不同的客户端（例如同步队列 APIs 或异步队列）来运行命令。我们建议您对域上的每个操作都使用命令和命令处理程序。通过采用这种方法，您可以通过引入新的命令和命令处理程序来添加新功能，而无需更改现有的业务逻辑。因此，使用命令模式可以更轻松地适应新的业务需求。

使用服务外墙或 CQRS 模式解耦组件

在六角形架构中，主适配器负责将来自客户端的传入读写请求松散耦合到域中。有两种方法可以实现这种松散耦合：使用服务立面模式或使用命令查询责任分离 (CQRS) 模式。

服务外观模式提供了一个前置接口，用于为客户端（例如表示层或微服务）提供服务。服务外观为客户提供多种读取和写入操作。它负责将传入的请求传输到域，并将从该域收到的响应映射到客户端。对于具有单一职责和多项操作的微服务来说，使用服务外观很容易。但是，在使用服务外墙时，很难遵循单一责任和开放式封闭原则。单一责任原则规定，每个模块只能对软件的单一功能负责。开放-封闭原则规定，代码应开放以供扩展，关闭以供修改。随着服务外观的扩展，所有操作都收集在一个接口中，更多的依赖关系被封装到该接口中，更多的开发人员开始修改相同的外观。因此，我们建议只有在开发期间服务显然不会扩展太多时才使用服务外观。

在六角形架构中实现主适配器的另一种方法是使用 CQRS 模式，该模式使用查询和命令将读取和写入操作分开。如前所述，命令是包含更改域状态所需的所有信息的对象。命令由命令处理程序方法执行。另一方面，查询不会改变系统的状态。它们的唯一目的是将数据返回给客户。在 CQRS 模式中，命令和查询是在单独的模块中实现的。这对于遵循事件驱动架构的项目尤其有利，因为命令可以作为异步处理的事件来实现，而查询可以使用 API 同步运行。查询也可以使用针对其进行优化的其他数据库。CQRS 模式的缺点是，与服务外墙相比，实施所需的时间更长。对于计划长期扩展和维护的项目，我们建议使用 CQRS 模式。命令和查询为应用单一责任原则和开发松散耦合软件提供了一种有效的机制，尤其是在大型项目中。

从长远来看，CQRS有很大的好处，但需要初始投资。因此，我们建议您在决定使用 CQRS 模式之前仔细评估您的项目。但是，您可以从一开始就使用命令和命令处理程序来构建应用程序，而无需将 read/write 操作分开。如果您稍后决定采用该方法，这将帮助您轻松地为 CQRS 重构项目。

组织规模

六角形架构、域驱动设计和（可选）CQRS 相结合，使您的组织能够快速扩展您的产品。根据康威定律，软件架构往往会演变以反映公司的通信结构。从历史上看，这种观察结果具有负面含义，因为大型组织通常根据数据库、企业服务总线等技术专业知识来组建团队。这种方法的问题在于，产品和功能开发总是涉及跨领域的问题，例如安全性和可扩展性，这需要团队之间的持续沟通。基于技术特征构建团队会在组织中造成不必要的孤岛，从而导致沟通不畅、缺乏所有权和忽视大局。最终，这些组织问题反映在软件架构中。

另一方面，Inverse Conw ay Manever根据促进软件架构的领域来定义组织结构。例如，跨职能团队负责一组特定的有限上下文，这些上下文是通过使用 DDD 和事件风暴来识别的。这些有界限的上下文可能反映了产品的非常具体的特征。例如，账户团队可能负责付款上下文。每个新功能都分配给一个新团

队，该团队的职责具有高度凝聚力和松散的职责，因此他们只能专注于该功能的交付，从而缩短上市时间。团队可以根据功能的复杂性进行扩展，因此可以将复杂的功能分配给更多的工程师。

最佳实践

对业务领域进行建模

从业务领域回到软件设计，确保你正在编写的软件符合业务需求。

使用领域驱动的设计 (DDD) 方法（例如事件风暴）对业务领域进行建模。活动风暴采用灵活的研讨会形式。在研讨会期间，领域和软件专家共同探讨业务领域的复杂性。软件专家利用研讨会的交付内容开始软件组件的设计和开发过程。

从一开始就编写和运行测试

使用测试驱动开发 (TDD) 来验证您正在开发的软件的正确性。TDD 在单元测试级别上效果最好。开发人员通过先编写测试来设计软件组件，然后调用该组件。该组件一开始没有实现，因此测试失败。下一步，开发人员实现组件的功能，使用带有模拟对象的测试夹具来模拟外部依赖项或端口的行为。测试成功后，开发人员可以通过实现真正的适配器来继续。这种方法可以提高软件质量并生成更具可读性的代码，因为开发人员了解用户将如何使用组件。Hexagonal 架构通过分离应用程序核心来支持 TDD 方法。开发人员编写的单元测试侧重于域核心行为。他们不必编写复杂的适配器来运行测试；相反，他们可以使用简单的模拟对象和固定装置。

使用行为驱动开发 (BDD) 来确保在功能层面上 end-to-end被接受。在 BDD 中，开发人员定义功能场景并与业务利益相关者进行验证。BDD 测试使用尽可能多的自然语言来实现这一目标。Hexagonal 架构通过其主适配器和辅助适配器的概念支持 BDD 方法。开发人员可以创建无需调用外部服务即可在本地运行的主适配器和辅助适配器。他们将 BDD 测试套件配置为使用本地主适配器来运行应用程序。

自动运行持续集成管道中的每项测试，以持续评估系统的质量。

定义域的行为

将域分解为实体、值对象和聚合（阅读有关实现域驱动设计的信息），并定义它们的行为。实现域的行为，以便在项目开始时编写的测试成功。定义调用域对象行为的命令。定义域对象在完成行为后发出的事件。

定义适配器可用于与域交互的接口。

自动测试和部署

在初步验证概念后，我们建议您花时间实施 DevOps实践。例如，持续集成和持续交付 (CI/CD) 管道以及动态测试环境可帮助您保持代码质量并避免部署过程中出现错误。

- 在 CI 流程中运行单元测试，并在合并代码之前对其进行测试。
- 构建 CD 流程，将应用程序部署到静态 dev/test 环境或动态创建的支持自动集成和 end-to-end测试的环境中。
- 自动执行专用环境的部署流程。

使用微服务和 CQRS 扩展您的产品

如果您的产品成功，请通过将软件项目分解为微服务来扩展您的产品。利用六角形架构提供的便携性来提高性能。将查询服务和命令处理程序拆分为单独的同步和异 APIs步。考虑采用命令查询责任分离 (CQRS) 模式和事件驱动架构。

如果您收到许多新功能请求，请考虑根据 DDD 模式扩展您的组织。正如本组织规模节前面所讨论的那样，以这样的方式组织您的团队，使他们拥有一个或多个功能作为受限上下文。然后，这些团队可以使用六角形架构来实现业务逻辑。

设计符合六角形建筑概念的项目结构

基础设施即代码 (IaC) 是云开发中广泛采用的做法。它允许您将基础架构资源（例如网络、负载均衡器、虚拟机和网关）定义和维护为源代码。这样，您就可以使用版本控制系统来跟踪架构的所有更改。此外，您还可以轻松创建和移动基础架构以进行测试。我们建议您在开发云应用程序时将应用程序代码和基础设施代码保存在同一个存储库中。这种方法可以轻松维护应用程序的基础架构。

我们建议您将应用程序分成三个与六角形架构概念对应的文件夹或项目：`entrypoints`（主适配器）、`domain`（域和接口）和`adapters`（辅助适配器）。

以下项目结构提供了在上设计 API 时采用这种方法的示例 AWS。按照之前的建议，该项目将应用程序代码 (app) 和基础设施代码 (infra) 保存在同一个存储库中。

```
app/  # application code
|--- adapters/  # implementation of the ports defined in the domain
     |--- tests/  # adapter unit tests
|--- entrypoints/  # primary adapters, entry points
```

```
     |--- api/  # api entry point
          |--- model/  # api model
          |--- tests/  # end to end api tests
|--- domain/  # domain to implement business logic using hexagonal architecture
     |--- command_handlers/  # handlers used to run commands on the domain
     |--- commands/  # commands on the domain
     |--- events/  # events emitted by the domain
     |--- exceptions/  # exceptions defined on the domain
     |--- model/  # domain model
     |--- ports/  # abstractions used for external communication
     |--- tests/  # domain tests
infra/  # infrastructure code
```

如前所述，域是应用程序的核心，不依赖于任何其他模块。我们建议您对domain文件夹进行结构化以包含以下子文件夹：

- command handlers包含在域上运行命令的方法或类。
- commands包含命令对象，这些对象定义了在域上执行操作所需的信息。
- events包含通过域发出然后路由到其他微服务的事件。
- exceptions包含域内定义的已知错误。
- model包含域实体、值对象和域服务。
- ports包含域与数据库或其他外部组件通信所使用的 APIs抽象。
- tests包含在域上运行的测试方法（例如业务逻辑测试）。

主适配器是应用程序的入口点，如entrypoints文件夹所示。此示例使用该api文件夹作为主适配器。此文件夹包含一个 APImodel，它定义了主适配器与客户端通信所需的接口。该tests文件夹包含 API 的 end-to-end测试。这些是浅层测试，用于验证应用程序的组件是否已集成并协同工作。

辅助适配器（如adapters文件夹所示）实现域端口所需的外部集成。数据库存储库就是辅助适配器的一个很好的例子。当数据库系统发生变化时，您可以使用域定义的实现来编写新的适配器。无需更改域名或业务逻辑。tests子文件夹包含每个适配器的外部集成测试。

上的基础架构示例 AWS

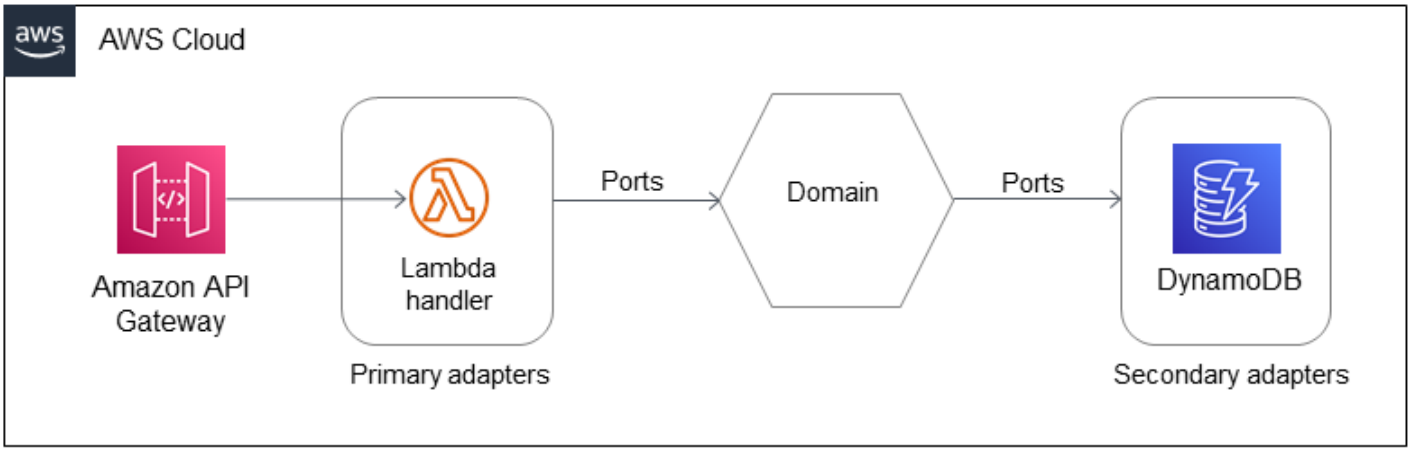
本节提供了为应用程序设计基础架构的示例 AWS ，您可以使用该基础架构来实现六边形架构。我们建议您从构建最低可行产品 (MVP) 的简单架构入手。大多数微服务需要一个入口点来处理客户端请求，一个计算层来运行代码，一个持久层来存储数据。以下 AWS 服务非常适合在六角形架构中用作客户端、主适配器和辅助适配器：

- 客户：亚马逊 API Gateway、亚马逊简单队列服务 (Amazon SQS)、Amazon Squee SQS、Elastic Load Balancing、亚马逊 EventBridge
- 主要适配器： AWS Lambda、亚马逊弹性容器服务 (Amazon ECS)、亚马逊 Elastic Kubernetes Service（亚马逊 EKS）、亚马逊弹性计算云（亚马逊 EC2）Amazon EC2
- 辅助适配器：亚马逊 DynamoDB、亚马逊关系数据库服务 (亚马逊 RDS)、亚马逊 Aurora、API Gateway、亚马逊 SQS、Elastic Load Balancing EventBridge、亚马逊简单通知服务 (Amazon SNS) Simple Notification Service

以下各节将在六角形架构的背景下更详细地讨论这些服务。

从简单开始

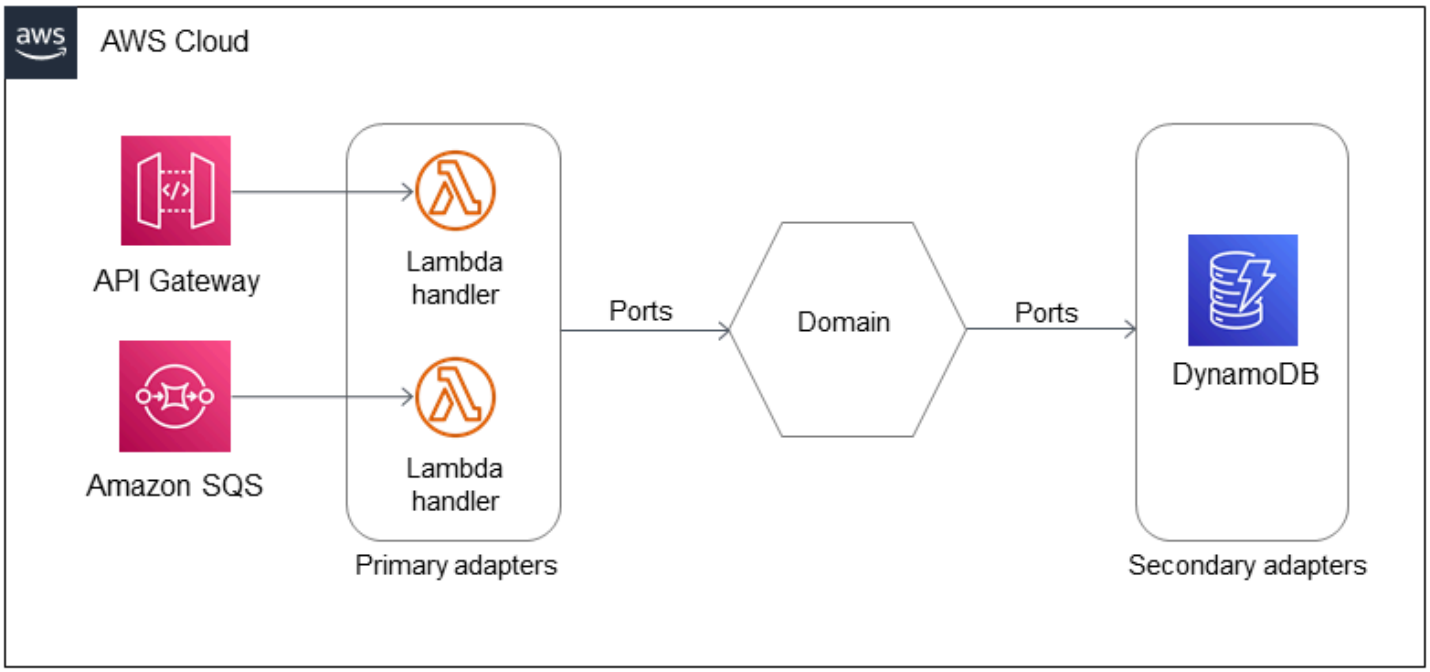
我们建议您在使用六角形架构架构应用程序时从简单开始。在此示例中，API Gateway 用作客户端 (REST API)，Lambda 用作主适配器（计算），DynamoDB 用作辅助适配器（持久性）。网关客户端调用入口点，在本例中，入口点是 Lambda 处理程序。



这种架构是完全无服务器的，为架构师提供了一个良好的起点。我们建议您在域中使用命令模式，因为它可以使代码更易于维护，并且可以适应新的业务和非功能需求。这种架构可能足以通过少量操作构建简单的微服务。

应用 CQRS 模式

如果域上的操作数量要扩展，我们建议您切换到 CQRS 模式。您可以使用以下示例将 CQRS 模式作为完全无服务器 AWS 的架构应用于中。

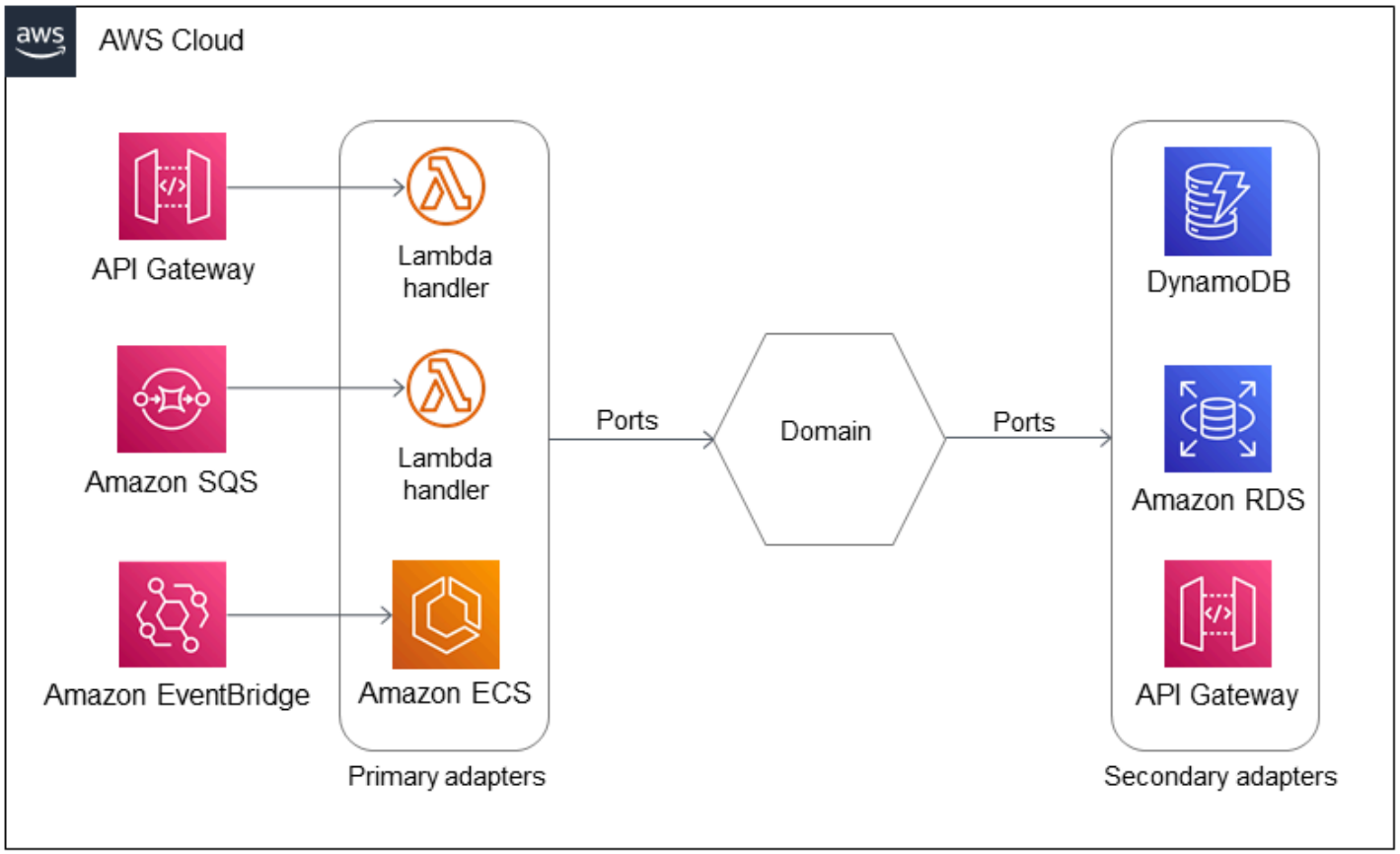


此示例使用两个 Lambda 处理程序，一个用于查询，一个用于命令。使用 API 网关作为客户端同步运行查询。使用 Amazon SQS 作为客户端，可以异步运行命令。

该架构包括多个客户端（API Gateway 和 Amazon SQS）和多个主适配器（Lambda），它们由相应的入口点（Lambda 处理程序）调用。所有组件都属于同一个有界上下文，因此它们位于同一个域中。

通过添加容器、关系数据库和外部 API 来改进架构

对于长时间运行的任务，容器是一个不错的选择。如果您有预定义的数据架构并希望从 SQL 语言的强大功能中受益，则可能还需要使用关系数据库。此外，该域必须与外部通信 APIs。您可以改进架构示例以支持这些要求，如下图所示。

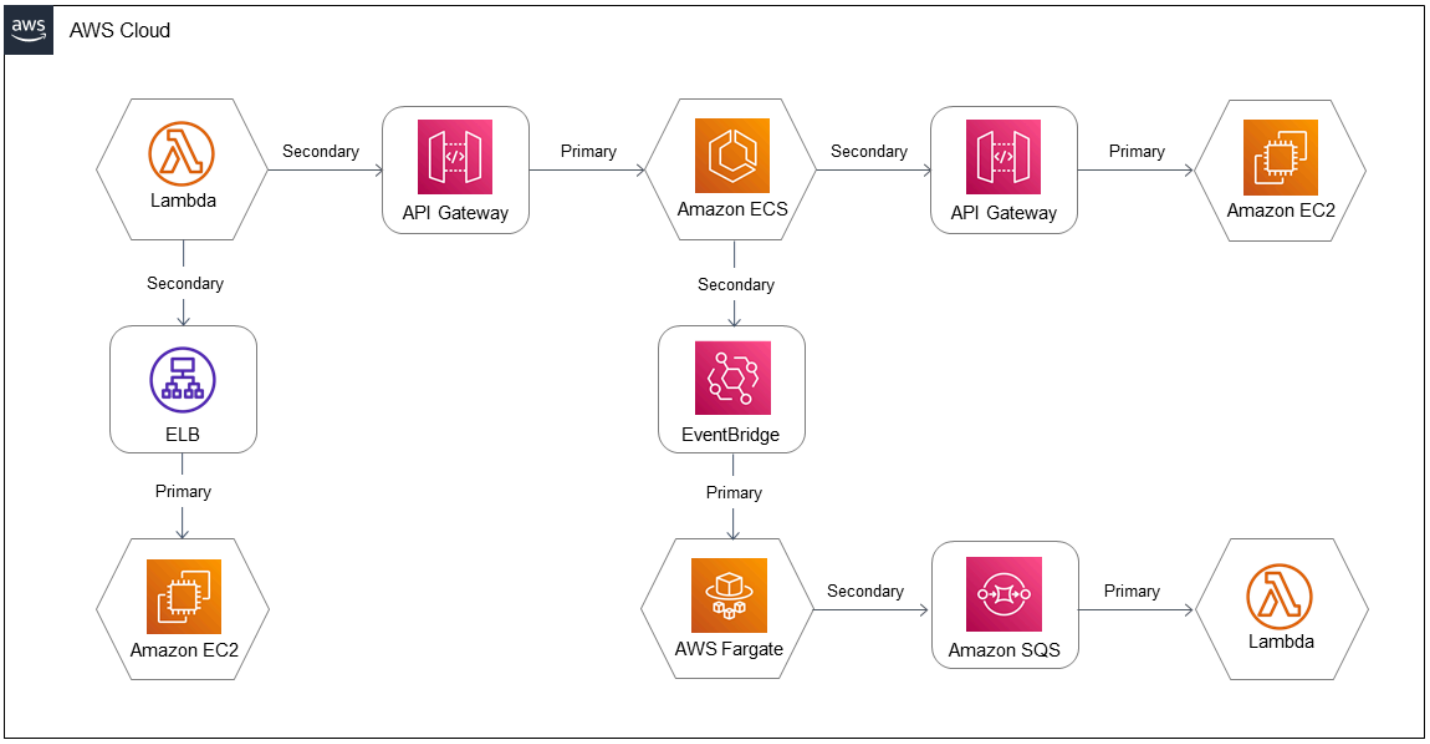


此示例使用 Amazon ECS 作为在域中启动长时间运行的任务的主要适配器。当特定事件发生时，亚马逊 EventBridge （客户端）会启动 Amazon ECS 任务（入口点）。该架构包括 Amazon RDS 作为另一个用于存储关系数据的辅助适配器。它还添加了另一个 API 网关作为辅助适配器，用于调用外部 API 调用。因此，该架构使用多个主适配器和辅助适配器，这些适配器依赖于一个业务域中的不同底层计算层。

该域始终通过称为端口的抽象与所有主适配器和辅助适配器松散耦合。该域使用端口定义了它对外界的需求。由于实现端口是适配器的责任，因此从一个适配器切换到另一个适配器不会影响域。例如，您可以通过编写新的适配器从 Amazon DynamoDB 切换到 Amazon RDS，而不会影响域。

添加更多域名（缩小）

六角形架构与微服务架构的原理非常吻合。到目前为止显示的架构示例包含单个域（或有界上下文）。应用程序通常包括多个域，这些域需要通过主适配器和辅助适配器进行通信。每个域代表一项微服务，并且与其他域松散耦合。



在此架构中，每个域使用一组不同的计算环境。（每个域也可能有多个计算环境，如前面的示例所示。）每个域都定义了通过端口与其他域通信所需的接口。端口是通过使用主适配器和辅助适配器实现的。这样，如果适配器发生变化，域名就不会受到影响。此外，域之间是相互分离的。

在上图所示的架构示例中，Lambda、Amazon EC2、Amazon ECS 和 AWS Fargate 用作主适配器。API Gateway、Elastic Load Balancing 和 Amazon SQS 用作辅助适配器。 EventBridge

常见问题解答

我为什么要使用六角形架构？

Hexagonal 架构将开发人员的注意力转移到领域逻辑上，简化了测试自动化，并提高了代码质量和适应性。这些改进可以缩短上市时间，更轻松地扩大技术和组织规模。

我为什么要使用域名驱动的设计？

域驱动设计 (DDD) 使您能够使用业务利益相关者和工程师之间的通用语言来构建软件组件和结构。DDD 可帮助您管理软件的复杂性，是长期维护软件产品的有效策略。

我可以在没有六角形架构的情况下练习测试驱动的开发吗？

可以。测试驱动开发 (TDD) 不仅限于特定的软件设计模式。但是，六角形架构使练习 TDD 变得更加容易。

我能否在没有六角形架构和域驱动设计的情况下扩展我的产品？

可以。大多数设计模式都可以实现技术和组织产品的扩展。但是，六角形架构和DDD使其更易于扩展，并且从长远来看，对于大型项目更有效。

我应该使用哪些技术来实现六角形架构？

Hexagonal 架构不仅限于特定的技术堆栈。我们建议您选择支持依赖反转和单元测试的技术。

我正在开发一种最低限度的可行产品。花时间思考软件架构有意义吗？

可以。我们建议您使用自己熟悉的设计模式 MVPs。我们鼓励您尝试练习六角形架构，直到您的工程师对它感到满意。为新项目建立六角形架构不需要比在没有任何架构的情况下开始所需的时间投入要多得多。

我正在开发一种最低限度的可行产品，没有时间写测试。

如果您的 MVP 包含业务逻辑，我们强烈建议您为其编写自动测试。这将减少反馈回路并节省时间。

我可以在六角形架构中使用哪些其他设计模式？

使用 C QRS 模式来支持整个系统的扩展。使用存储库模式来存储和恢复您的域模型。使用工作单位模式来管理事务处理步骤。使用组合而不是继承来对域聚合、实体和值对象进行建模。不要构建复杂的对象层次结构。

后续步骤

- 阅读本节中收集的链接，进一步熟悉领域驱动的设计概念。资源
- 如果您要实施新项目，请使用本指南中提供的项目结构模板并实现一些功能。
- 如果您正在实施现有项目，请确定可以在只读和只写操作之间拆分的代码。将只读代码抽象到查询服务中，并将只写代码放入命令处理程序中。
- 基本项目结构到位后，编写单元测试，建立与测试自动化的持续集成 (CI)，并遵循测试驱动开发 (TDD) 实践。

资源

参考

- 使用 AWS Lambda（AWS 规范指导模式）在六角形架构中构建 Python 项目
- 敏捷团队（规模化敏捷框架网站）
- 哈里·珀西瓦尔和鲍勃·格雷戈里的 Python 架构模式（O'Reilly Media，2020 年 3 月 31 日），特别是以下章节：
  - 命令和命令处理程序
  - 命令查询责任分离 (CQRS)
  - 存储库模式
- 活动风暴：超越孤岛界限的最明智的协作方法，作者：Alberto Brandolini（Event Storming 网站）
- 揭开康威定律的神秘面纱，作者：山姆·纽曼（Thoughtworks 网站，2014 年 6 月 30 日）
- 与詹姆斯·贝斯威克 AWS Lambda共同开发进化架构（AWS 计算博客，2021 年 7 月 8 日）
- 领域语言：解决软件核心的复杂性（域名语言网站）
- Facade，摘自亚历山大·什维茨的《深入了解设计模式》（电子书，2018 年 12 月 5 日）
- GivenWhenThen，作者：马丁·福勒（2013 年 8 月 21 日）
- 实施领域驱动的设计，作者：Vaughn Vernon（Addison-Wesley Professional，2013 年 2 月）
- 反向康威机动（Thoughtworks 网站，2014 年 7 月 8 日）
- 本月模式：红绿重构（DZone 网站，2017 年 6 月 2 日）
- Thorben Janssen 的《SOL@@ ID 设计原理解释：依赖关系反转原理与代码示例》（Stackify 网站，2018 年 5 月 7 日）
- 《扎实的原则：解释和示例》，作者：西蒙·霍伯格（ITNEXT 网站，2019 年 1 月 1 日）
- James Shore 和 Shane Warden 的《敏捷开发的艺术：测试驱动开发》（O'Reilly Media，2010 年 3 月 25 日）
- Y@@ igit Kemal Erinc 用通俗易懂的英语解释的面向对象编程的可靠原理（freeCodeCamp 面向对象编程帖子，2020 年 8 月 20 日）
- 什么是事件驱动架构？（AWS 网站）

AWS 服务

- Amazon API Gateway

- Amazon Aurora
- Amazon DynamoDB
- 亚马逊弹性计算云（亚马逊 EC2）
- Amazon Elastic Container Service（Amazon ECS）
- Amazon Elastic Kubernetes Service(Amazon EKS)
- Elastic Load Balancing
- Amazon EventBridge
- AWS Fargate
- AWS Lambda
- Amazon Relational Database Service (Amazon RDS)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)

其他工具

- 摩托
- LocalStack

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
初次发布	—	2022 年 6 月 15 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、开发人员和从事冲刺工作的 DevOps 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，彼此独立，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，该服务可以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。