了解和实现微前端 AWS

AWS 规范性指导

AWS 规范性指导: 了解和实现微前端 AWS

Table of Contents

了解和实现微前端 AWS

亚马逊 Web Services（贡献者）

2024 年 7 月（文件历史记录）

随着组织追求敏捷性和可扩展性，传统的整体架构往往成为瓶颈，阻碍了快速开发和部署。微前端通过将复杂的用户界面分解为可以自主开发、测试和部署的更小、独立的组件来缓解这种情况。这种方法提高了开发团队的效率，促进了后端和前端之间的协作，从而促进了分布式系统的 end-to-end 协调。

本规范性指南旨在帮助不同专业领域的 IT 负责人、产品负责人和架构师了解微前端架构并在 Amazon Web Services 上构建微前端应用程序 ()。AWS

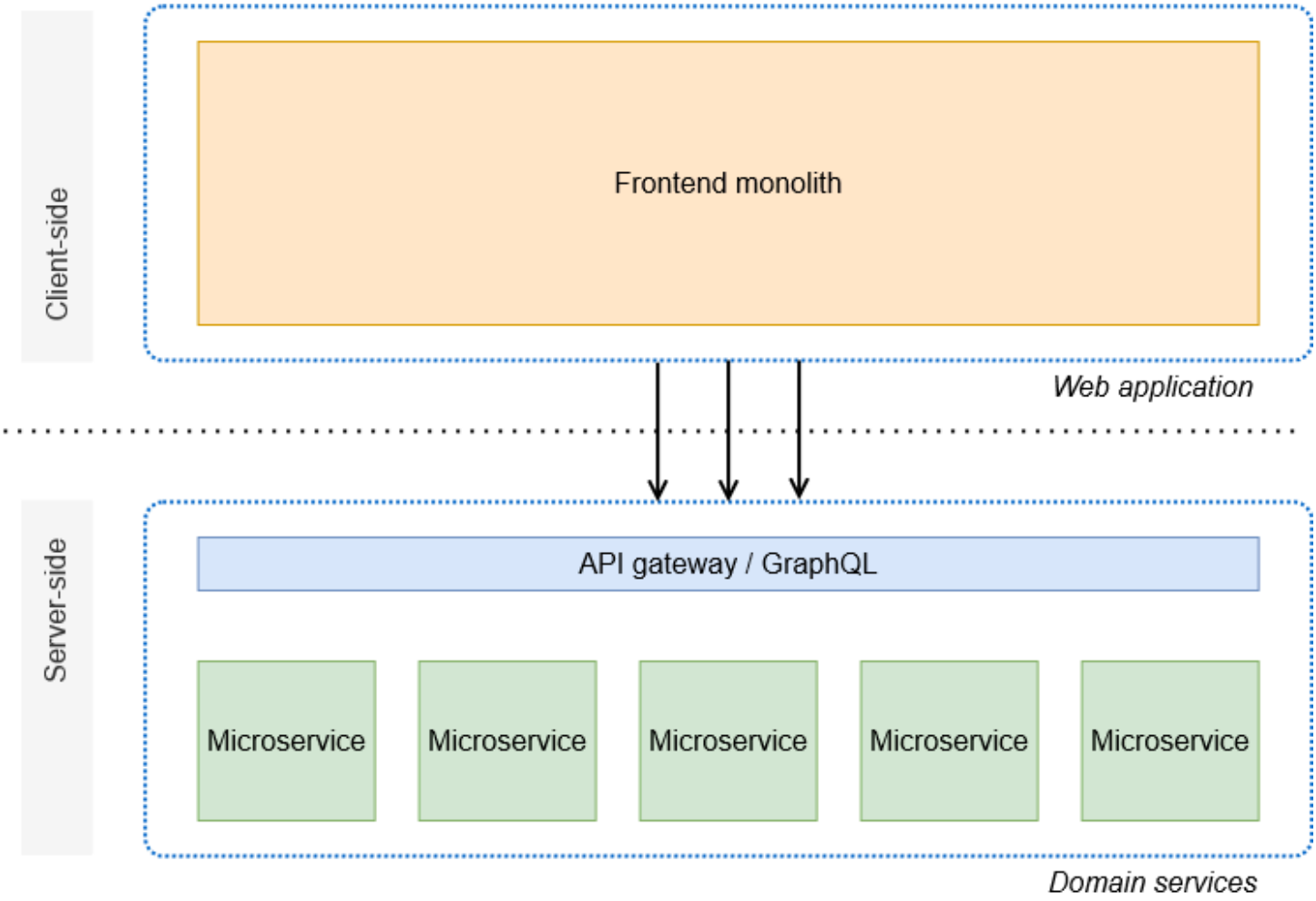
概述

微前端是一种架构，其基础是将应用程序前端分解为独立开发和部署的工件。当您将大型前端拆分为自主软件工件时，您可以封装业务逻辑并减少依赖关系。这支持更快、更频繁地交付产品增量。

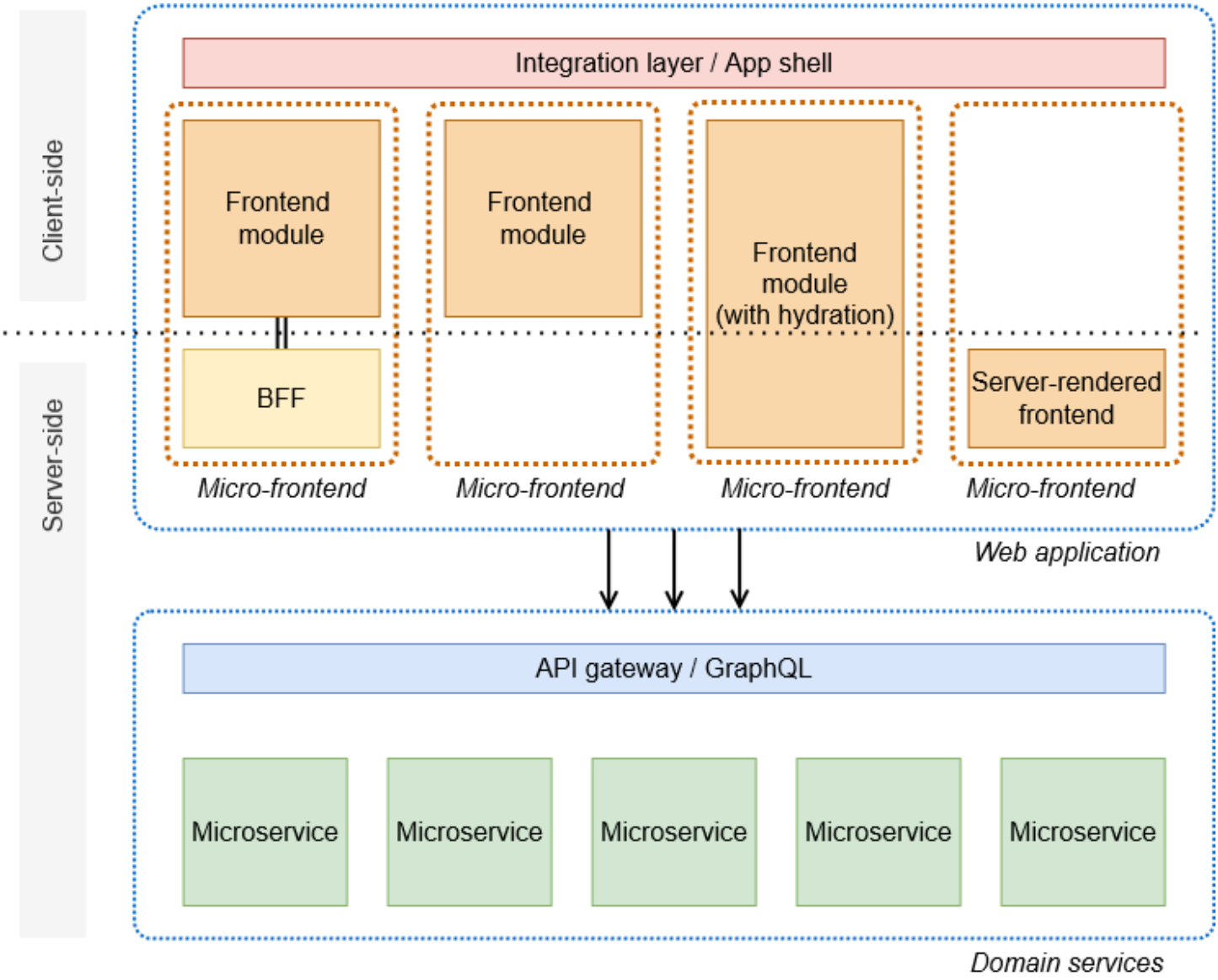
微前端与微服务类似。实际上，微前端一词源自微服务一词，它旨在传达微服务作为前端的概念。虽然微服务架构通常将后端的分布式系统与单片前端结合在一起，但微前端是独立的分布式前端服务。可以通过两种方式设置这些服务：

- 仅限前端，与在后面运行微服务架构的共享 API 层集成
- 全栈，这意味着每个微前端都有自己的后端实现。

下图显示了传统的微服务架构，其前端整体结构使用 API 网关连接到后端微服务。



下图显示了具有不同微服务实现方式的微前端架构。



如上图所示，您可以将微前端与客户端渲染或服务器端渲染架构配合使用：

- 客户端渲染的微前端可以直接使用集中式 API Gateway 公开的 API。
- 团队可以在有限的上下文中创建 backend-for-frontend (BFF)，以减少前端对 API 的闲聊。
- 在服务器端，微前端可以通过使用一种称为水合的技术在客户端增强服务器端方法来表达。当浏览器渲染页面时，关联的页面会 JavaScript 被水合以允许与用户界面元素进行交互，例如单击按钮。
- 微前端可以在后端进行渲染，并使用超链接路由到网站的新部分。

微前端非常适合想要执行以下操作的组织：

- 通过多个团队在同一个项目上进行扩展。

- 拥抱决策的去中心化，使开发人员能够在已确定的系统边界内进行创新。

这种方法大大减轻了团队的认知负担，因为他们负责系统的特定部分。它提高了业务灵活性，因为可以在不中断其余部分的情况下对系统的一部分进行修改。

微前端是一种独特的架构方法。尽管构建微前端有不同的方法，但它们都有共同的特征：

- 微前端架构由多个独立元素组成。其结构类似于后端微服务所采用的模块化。
- 微前端完全负责其有限环境中的前端实现，其中包括以下内容：
  - 用户界面
  - 数据
  - 状态或会话
  - 业务逻辑
  - 流

有界上下文是一个内部一致的系统，其边界经过精心设计，可以调解可以进入和退出的内容。微前端应尽可能少地与其他微前端共享业务逻辑和数据。无论何时需要共享，都通过明确定义的接口（例如自定义事件或响应式流）进行。但是，当涉及到一些跨领域问题（例如设计系统或日志库）时，欢迎有意共享。

推荐的模式是使用跨职能团队来构建微前端。这意味着每个微前端都是由从后端到前端工作的同一个团队开发的。从编码到系统在生产中的运营，团队所有权至关重要。

本指南无意推荐一种特定的方法。相反，它讨论了不同的模式、最佳实践、权衡取舍，以及架构和组织方面的考虑因素。

基础概念

微前端架构在很大程度上受到前面三个架构概念的启发：

- 领域驱动的设计是将复杂的应用程序结构化为连贯域的心理模型。
- 分布式系统是一种将应用程序构建为松散耦合子系统的方法，这些子系统是独立开发并在自己的专用基础设施上运行的。
- 云计算是一种通过 pay-as-you-go 模型将 IT 基础设施作为服务运行的方法。

以领域为导向的设计

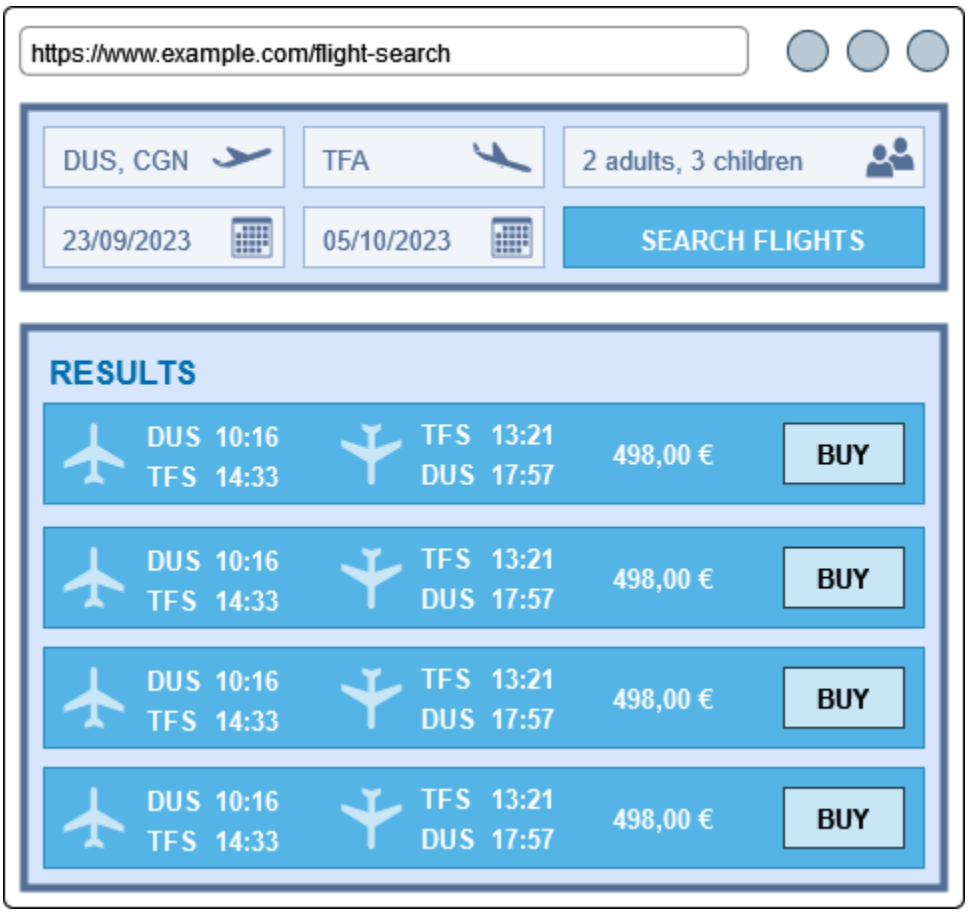
领域驱动设计 (DDD) 是埃里克·埃文斯开发的一种范式。埃文斯在2003年的著作《领域驱动的设计：解决软件核心的复杂性》中假设软件开发应该由业务问题而不是技术问题驱动。埃文斯提出，IT项目首先开发一种无处不在的语言，以帮助技术和领域专家找到共同的理解。基于这种语言，他们可以制定一个相互理解的商业现实模型。

尽管这种方法可能显而易见，但许多软件项目都存在业务与IT脱节的困扰。这些脱节往往会导致严重的误解，从而导致预算超支、质量下降或项目失败。

埃文斯引入了其他多个重要术语，其中一个是有限上下文。有限的上下文是大型 IT 应用程序中独立的部分，其中包含仅针对一个业务问题的解决方案或实施。大型应用程序将由多个有界上下文组成，这些上下文通过集成模式松散耦合。这些有限的上下文甚至可以有自己的无处不在的语言的方言。例如，应用程序付款环境中的用户可能与交付环境中的用户有不同的方面，因为在付款过程中，运费的概念无关紧要。

Evans 没有定义有界上下文应该有多小或多大。大小由软件项目决定，并且可能会随着时间的推移而演变。衡量上下文边界的良好指标是实体（域对象）和业务逻辑之间的凝聚程度。

在微前端的背景下，可以用复杂的网页（例如航班预订页面）来说明域驱动的设计。



在此页面上，主要的构建块是搜索表单、筛选器面板和结果列表。要确定边界，必须确定独立的功能上下文。此外，还要考虑非功能方面，例如可重用性、性能和安全性。“属于同一事物” 的最重要指标是它们的沟通模式。如果架构中的某些元素必须频繁通信并交换复杂的信息，则它们可能共享相同的边界上下文。

单个用户界面元素（例如按钮）不是受限上下文，因为它们在功能上并不独立。此外，整个页面不适合有限的上下文，因为它可以分解为较小的独立上下文。合理的方法是将搜索表单视为一个有界上下文，将结果列表视为第二个有界上下文。现在，这两个有界上下文中的每一个都可以作为单独的微前端来实现。

分布式系统

为了简化维护和支持发展能力，大多数非平凡的 IT 解决方案都是模块化的。在这种情况下，模块化意味着 IT 系统由可识别的构建块组成，这些构建块通过接口解耦以实现关注点分离。

除了模块化之外，分布式系统本身还应该是独立的系统。在单纯的模块化系统中，每个模块都经过理想的封装，并通过接口公开其功能，但它不能独立部署，甚至无法单独运行。此外，模块通常与属于同一

系统的其他模块遵循相同的生命周期。另一方面，分布式系统的各个组成部分都有自己的生命周期。应用领域驱动的设计范式，每个构建块都针对一个业务领域或子领域，并生活在自己的有限环境中。

当分布式系统在构建期间进行交互时，常见的方法是开发用于快速识别问题的机制。例如，你可以采用类型化语言，并在单元测试上投入大量资金。多个团队可以协作开发和维护模块，这些模块通常作为库分发，供系统使用 npm、Apache Maven 和 pip 等工具。 NuGet

在运行期间，交互式分布式系统通常由各个团队拥有。由于错误处理、性能平衡和安全性，使用依赖关系会导致操作复杂性。对集成测试和可观察性的投资是降低风险的基础。

当今最受欢迎的分布式系统示例是微服务。在微服务架构中，后端服务由域驱动（而不是由用户界面或身份验证等技术问题驱动），由自治团队拥有。微前端具有相同的原理，将解决方案范围扩展到前端。

云计算

云计算是一种通过 pay-as-you-go 模型购买 IT 基础设施作为服务的方式，而不是自己建造数据中心并购买硬件在内部运行这些中心。云计算具有以下几个优点：

- 您的组织能够尝试新技术，而无需预先做出大量的长期财务承诺，从而获得显著的业务灵活性。
- 通过使用诸如之类的云提供商 AWS，您的组织可以访问一系列维护成本低且高度集成的服务（例如 API 网关、数据库、容器编排和云功能）。访问这些服务可以让您的员工腾出时间专注于让您的组织在竞争中脱颖而出的工作。
- 当您的组织准备好在全球范围内推出解决方案时，您可以将该解决方案部署到世界各地的云基础架构。

云计算通过提供高度托管的基础架构来支持微前端。这使得跨职能团队更容易 end-to-end 拥有所有权。虽然团队应具备丰富的运营知识，但基础设施配置、操作系统更新和联网等手动任务会分散注意力。

由于微前端存在于有限的环境中，因此团队可以选择最合适的服务来运行它们。例如，团队可以在云函数和容器之间进行选择，也可以在不同类型的 SQL 和 NoSQL 数据库或内存缓存之间进行选择。团队甚至可以在高度集成的工具包上构建自己的微前端 AWS Amplify，例如，该工具包带有用于无服务器基础架构的预配置构建块。

将微前端与替代架构进行比较

与所有架构策略一样，采用微前端的决定必须基于以组织原则为指导的评估标准。微前端各有优缺点。如果您的组织决定使用微前端，则必须制定策略来应对分布式系统的挑战

在选择应用程序架构时，最受欢迎的微前端替代方案是单体结构、n 层应用程序以及与单页应用程序 (SPA) 前端相结合的微服务。这些都是有效的方法，每种方法都有优点和缺点。

巨石

不需要频繁更改的小型应用程序可以作为整体快速交付。即使在预计会有显著增长的情况下，巨石也是自然而然的第一步。稍后，可以将巨石淘汰或重构为更灵活的结构。从一块巨石开始，您的组织可以更快地进入市场、获得客户反馈并改进产品。

但是，如果不谨慎维护或代码库随着时间的推移而扩大，单片应用程序往往会降级。当多个团队为同一个代码库做出重大贡献时，他们很少都为其维护和运营做出贡献。这会导致责任失衡，从而影响速度并导致效率低下。同时，随着代码库的发展，整体模块之间的无意耦合会导致意想不到的副作用。这些副作用可能导致故障和中断。

N 层应用程序

具有相对静态发展速度的更复杂的应用程序可以构建为三层架构（演示、应用程序、数据），在前端和后端之间有一个 REST 或 GraphQL 层。这要灵活得多，不同级别的团队可以在一定程度上独立发展。n 层应用程序的缺点是部署功能要困难得多。前端和后端通过 API 合约分离，因此重大更改必须一起部署，或者必须对 API 进行版本控制。

考虑以下常见场景：如果发布新功能需要更改数据架构，则产品负责人可能需要几天时间才能与前端团队就一组功能达成一致。然后，前端团队将要求后端团队自行开发和发布功能。后端团队将与数据所有者合作发布数据库架构更新。接下来，后端团队将发布新版本的 API，以便前端团队可以开发和发布他们的更改。在这种情况下，将所有更改传播到生产环境可能需要数周甚至数月的时间，因为每个团队在开发、测试和发布更改方面都有自己的待办事项、优先事项和机制。

微服务

在微服务架构中，后端被分解为小型服务，每个服务都在有限的环境中解决特定的业务问题。通过公开明确定义的接口合约，每个微服务还与其他服务实现了高度的分离。

值得一提的是，有限的上下文和接口合约也应该存在于精心制作的巨体和 n 层架构中。但是，在微服务架构中，通信通过网络（通常是 HTTP 协议）进行，并且服务具有专用的运行时基础架构。这支持每项后端服务的独立开发、交付和操作。

根据您的要求选择方法

Monoliths 和 n 层架构将多个领域问题捆绑到一个技术工件中。这使得依赖关系和内部数据流等方面易于管理，但却使新功能的交付变得更加困难。为了维护连贯的代码库，团队经常在重构和解耦上投入时间，因为他们必须处理庞大的代码库。

由几个团队开发的应用程序可能不需要迁移到微前端所带来的额外复杂性。如果团队没有为发布变更付出高耦合度和长交货时间的罚款，则尤其如此。

总而言之，对于复杂且快速发展的应用程序，更复杂的分布式架构通常是正确的选择。对于中小型应用程序，分布式架构不一定优于单片架构，尤其是在应用程序不会在短时间内显著演变的情况下。

微前端中的架构决策

在应用程序中应用微前端架构模式的团队必须尽早就架构做出多项决定：

- 微前端的识别和边界的定义
- 使用微前端撰写页面和视图
- 跨微前端的路由、状态管理和通信
- 管理跨领域问题的依赖关系

以下各节将更深入地介绍这些主题。

在做出架构决策时，必须有正确的指标，了解使用模式、应用程序特征和权衡取舍。例如，与视频编辑工具或可观察性仪表板相比，电子商务网站具有不同的特征和使用模式。

可以针对初始页面加载指标（例如互动时间 (TTI) 和 First Contentful Paint (FCP)）对流量大、会话深度较短的面向公众的应用程序进行优化。相比之下，用户在一天开始时登录并全天保持互动的应用程序可能会针对应用程序内体验进行优化。应用团队可能会在每次导航后而不是初始页面加载后针对首次输入延迟 (FID) 指标进行优化。

公共网站必须适应各种浏览器环境。对客户端环境有已知限制的企业应用程序可以根据其限制条件优化其微前端组合。

架构决策没有一个正确的选择。了解权衡取舍、业务运营环境、使用模式和指标，以指导适合每个应用程序的决策。

识别微前端边界

为了提高团队自主权，可以将应用程序提供的业务功能分解为几个微前端，彼此之间的依赖性最小。

按照前面讨论的 DDD 方法，团队可以将应用程序域分解为业务子域和受限上下文。然后，自治团队可以拥有其受限上下文的功能，并将这些上下文作为微前端交付。有关关注点分离的更多信息，请参阅 Serverless Land 图。

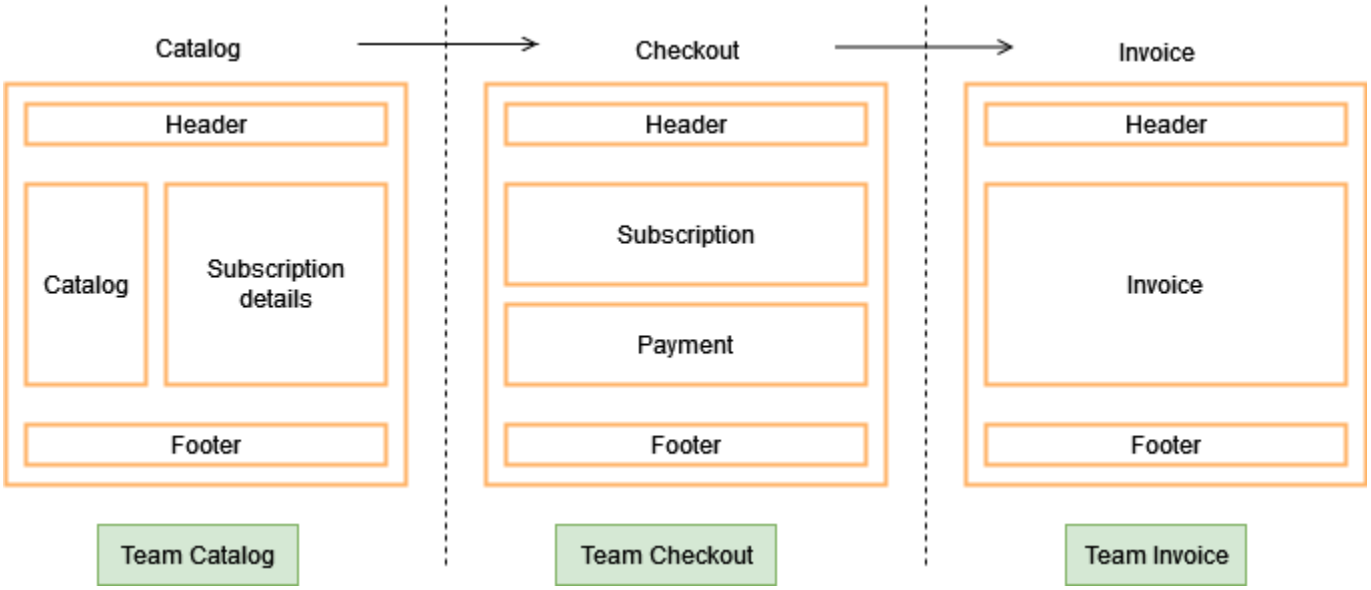
定义明确的有界上下文应最大限度地减少功能重叠以及跨上下文进行运行时通信的需求。可以使用事件驱动的方法实现所需的通信。这与用于微服务开发的事件驱动架构没有什么不同。

架构良好的应用程序还应支持新团队在未来交付扩展，从而为客户提供一致的体验。

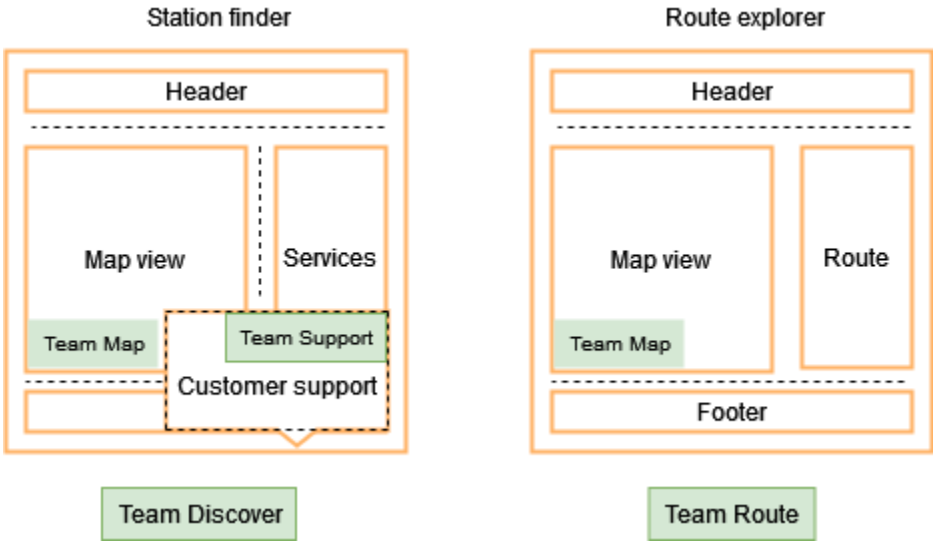
如何将单片应用程序切成微前端

概述部分包括一个在网页上识别独立功能上下文的示例。出现了几种在用户界面上拆分功能的模式。

例如，当业务域形成用户旅程的各个阶段时，可以在前端应用垂直分割，即用户旅程中的视图集合作为微前端交付。下图显示了垂直分割，其中 “目录”、“结账” 和 “发票” 步骤由不同的团队作为单独的微前端交付。



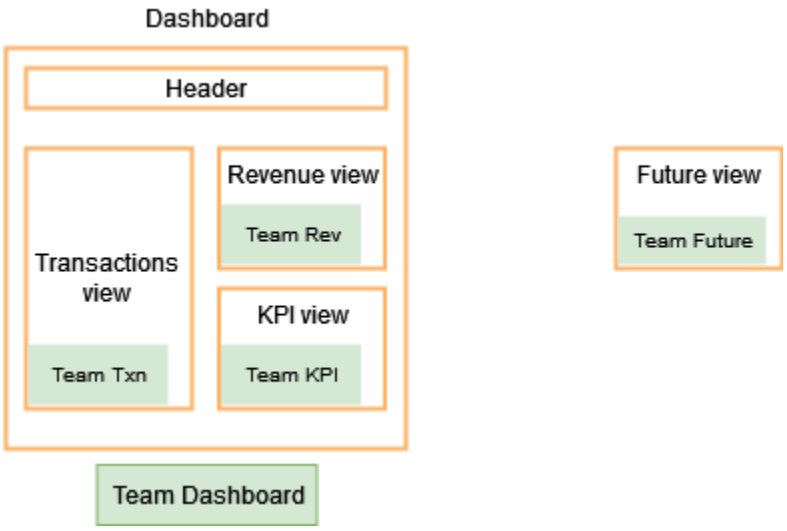
对于某些应用程序，仅靠垂直拆分可能还不够。例如，可能需要在许多视图中提供某些功能。对于这些应用程序，您可以应用混合拆分。下图显示了一种混合拆分解决方案，其中 Station finder 和 Route Explorer 的微前端都使用地图视图功能。



门户型或仪表板型应用程序通常将前端功能整合到一个视图中。在这些类型的应用程序中，每个控件都可以作为微前端交付，托管应用程序定义了微前端应实现的约束和接口。

这种方法为微前端提供了一种机制来处理诸如视口大小、身份验证提供程序、配置设置和元数据之类的问题。这些类型的应用程序针对可扩展性进行了优化。新团队可以开发新功能来扩展仪表板功能。

下图显示了由三个独立团队开发的仪表板应用程序，这些团队是团队控制面板的一部分。



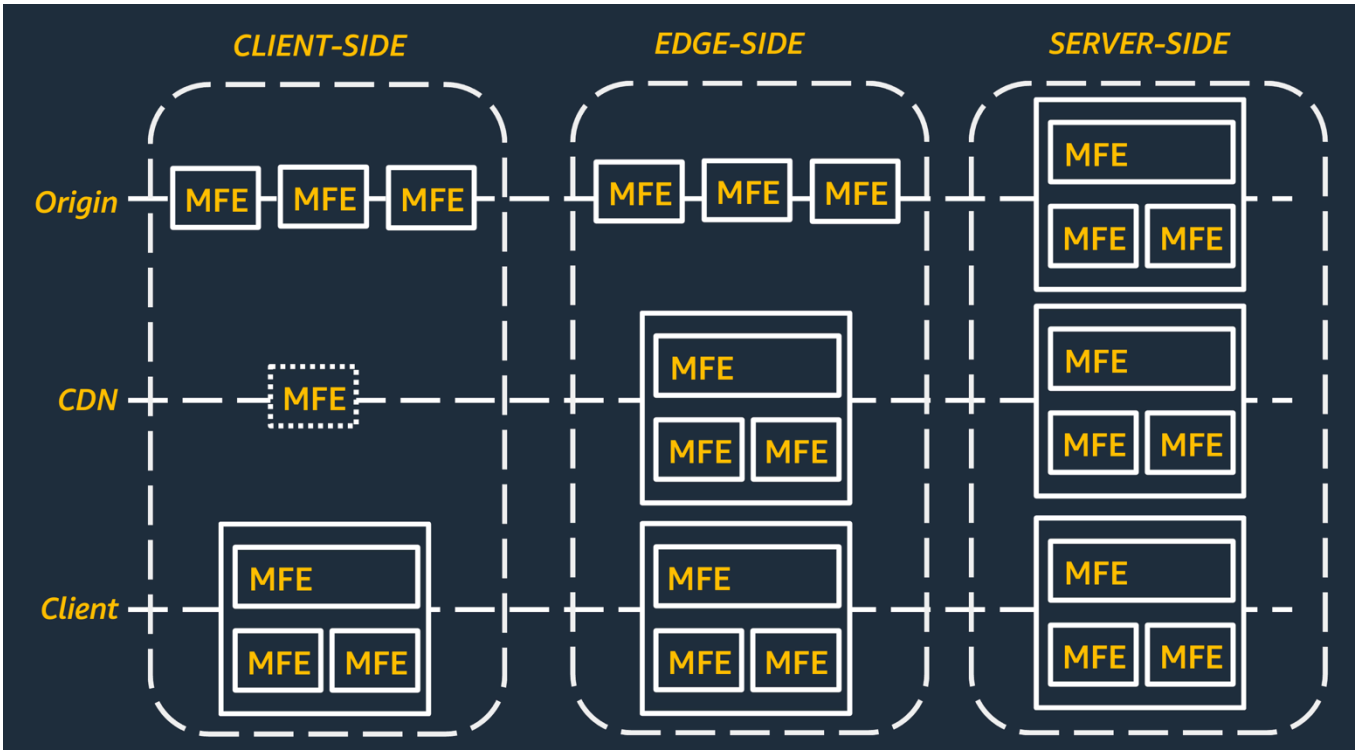
在图中，future 视图表示新团队为扩展团队控制面板和仪表板功能而开发的新功能。

门户和仪表板应用程序通常通过在用户界面中使用混合拆分来组合功能。微前端可通过定义明确的设置进行配置，包括位置和大小限制。

使用微前端撰写页面和视图

您可以使用客户端合成、边缘组合和服务器端合成来组合应用程序的视图。组合模式在必要的团队技能、容错能力、性能和缓存行为方面具有不同的特征。

下图显示了微前端架构的客户端、边缘端和服务器端层的组合是如何发生的。



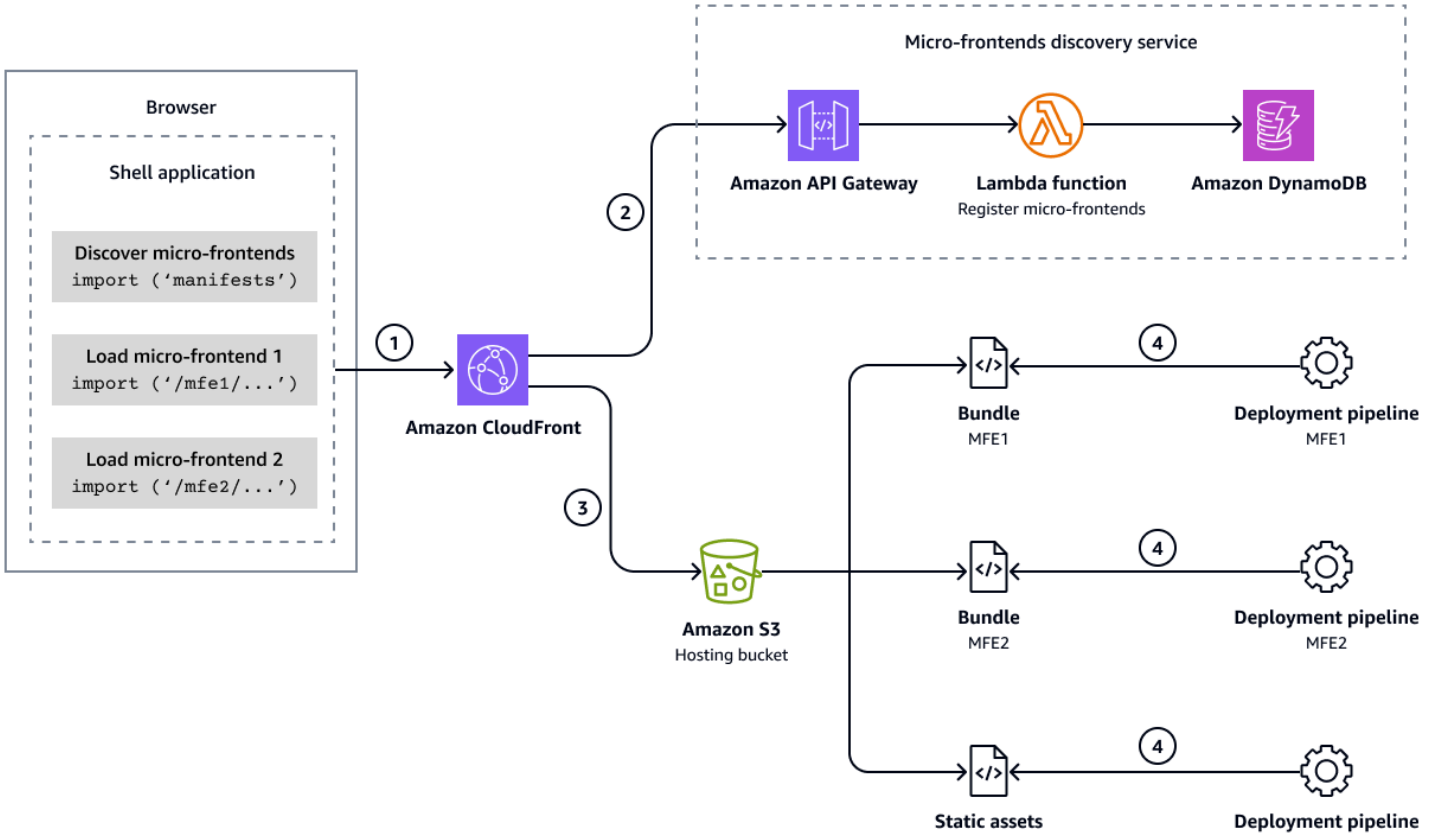
以下各节将讨论客户端、边缘端和服务器端层。

客户端合成

在客户端（浏览器或移动 Web 视图）上以文档对象模型 (DOM) 片段的形式动态加载和附加微前端。微前端工件（例如 JavaScript 或 CSS 文件）可以从内容分发网络 (CDN) 加载，以减少延迟。客户端组合需要满足以下条件：

- 一个拥有和维护 shell 应用程序或微前端框架的团队，以便能够在浏览器中在运行时发现、加载和渲染微前端组件
- 前端技术（例如 HTML、CSS 和）方面的高技能水平以及 JavaScript对浏览器环境的深入理解
- 优化页面中的 JavaScript 加载量，以及避免全局命名空间冲突的纪律

下图显示了无服务器客户端 AWS 组合的示例架构。



客户端合成在浏览器环境中通过 shell 应用程序进行。该图显示了以下细节：

1. 加载外壳应用程序后，它会向 Amazon 发出初始请求 CloudFront，以发现要通过清单终端节点加载的微前端。
2. 清单包含有关每个微前端的信息（例如，名称、URL、版本和后备行为）。清单由微前端发现服务提供。在图中，该发现服务由 Amazon API Gateway（一个 AWS Lambda 函数）和亚马逊 DynamoDB 表示。shell 应用程序使用清单信息请求各个微前端在给定布局中撰写页面。
3. 每个微前端包都由静态文件（例如 JavaScript CSS 和 HTML）组成。这些文件托管在亚马逊简单存储服务 (Amazon S3) 存储桶中并通过它提供服务。 CloudFront
4. 团队可以使用他们拥有的部署管道部署其微前端的新版本并更新清单信息。

边缘构图

在通过网络将页面发送给客户端之前，使用某些CDN和代理支持的边缘端包含 (ESI) 或服务器端包含 (SSI) 等嵌入技术来撰写页面。ESI 要求满足以下条件：

- 具有 ESI 功能的 CDN，或者服务器端微前端前的代理部署。诸如 HAProxy、Varnish 和 NGINX 之类的代理实现支持 SSI。

- 了解 ESI 和 SSI 实现的使用和局限性。

开始使用新应用程序的团队通常不会为其构图模式选择边缘构图。但是，这种模式可能会为依赖嵌入的遗留应用程序提供一条途径。

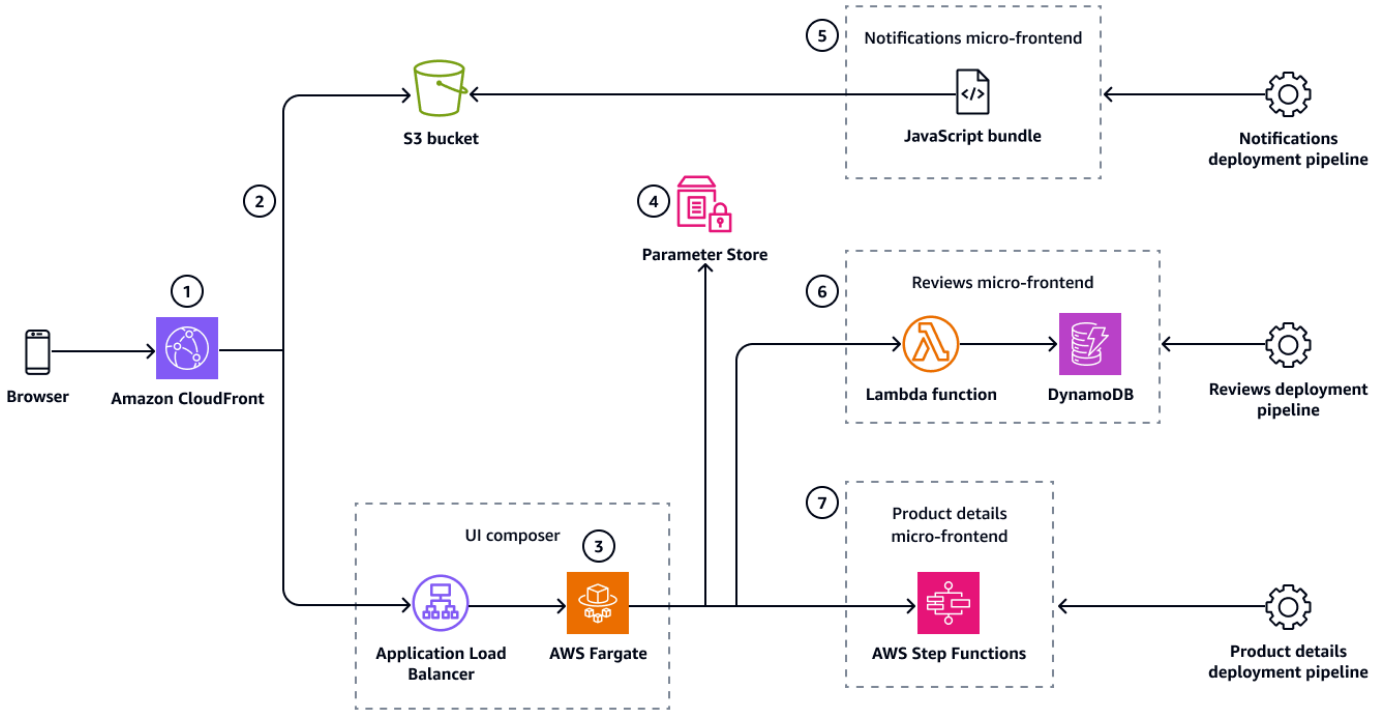
服务器端合成

在页面缓存到边缘之前，使用源服务器来撰写页面。这可以通过传统技术（例如 PHP、Jakarta Server Pages (JSP) 或模板库）来完成，通过包含来自微前端的片段来撰写页面。您还可以使用服务器上运行的 JavaScript 框架（例如 Next.js），在服务器上使用服务器端渲染 (SSR) 来撰写服务器上的页面。

在服务器上呈现页面后，可以将其缓存在CDN上以减少延迟。部署新版本的微前端时，必须重新渲染页面，并且必须更新缓存以向客户提供最新版本。

服务器端组合需要对服务器环境有深入的了解，以建立部署、发现服务器端微前端和缓存管理的模式。

下图显示了服务器端的构成。



该图包括以下组件和流程：

1. Amazon CloudFront 为该应用程序提供了一个独特的入口点。该发行版有两个来源：第一个用于静态文件，第二个用于用户界面编辑器。

2. 静态文件托管在 Amazon S3 存储桶中。它们由浏览器和界面编辑器用于 HTML 模板。
3. UI 编辑器在中的容器集群上运行AWS Fargate。借助容器化解决方案，您可以根据需要使用流媒体功能和多线程渲染。
4. Parameter Store（一种功能）被用作基本的微前端发现系统。 AWS Systems Manager此功能提供了 UI 编辑器使用的键值存储，用于检索要使用的微前端端点。
5. 通知微前端将优化的 JavaScript 捆绑包存储在 S3 存储桶中。这会在客户端上呈现，因为它必须对用户交互做出反应。
6. 评论微前端由 Lambda 函数组成，用户评论存储在 Dynam oDB 中。评论的微前端完全在服务器端呈现，并输出一个 HTML 片段。
7. 产品详细信息微前端是一个低代码的微前端，它使用. AWS Step Functions Express Workflow 可以同步调用，它包含渲染 HTML 片段的逻辑和缓存层。

有关服务器端组合的更多信息，请参阅博客文章服务器端渲染微前端——架构。

跨微前端的路由和通信

路由选项取决于组合方法。可以通过减少前端组件之间的耦合来优化通信。

路由

使用垂直分割的客户端组合的应用程序可以使用服务器端路由（多页应用程序）或客户端路由（单页应用程序）。如果他们使用混合拆分进行界面构成，则必须使用客户端路由来支持页面上更深层次的微前端路由层次结构。

使用边缘组合和服务器端组合的应用程序更适合服务器端路由，或者使用边缘计算进行路由，例如带有 Amazon 的 Lambda @Edge。 CloudFront

微前端之间的通信

对于微前端架构，我们建议减少前端组件之间的耦合。减少耦合的一种方法是从同步函数调用转向异步消息传递。

浏览器运行时和用户交互本质上是异步的。生产者和消费者之间可以通过消息交换事件。这些事件为微前端之间的通信提供了一个定义明确的接口。

如果您按照 DDD 实践来识别微前端的边界上下文，那么下一步就是确定必须跨界通信的事件。

事件的消息传递机制可以是平台团队提供的原生 DOM JavaScript 事件 (`CustomEvents`)、事件发射器或响应式流库。微前端发布事件并订阅与其边界上下文相关的事件。使用这种方法，发布者和订阅者无需相互了解。合同是事件的定义。有关此内容的直观表示，请参阅 “带有事件架构的边界上下文” 图的 “与事件沟通” 部分。

管理跨领域问题的依赖关系

有意识的依赖管理对于微前端等分布式架构的成功至关重要。依赖管理是微前端开发中最具挑战性的部分之一。

在微前端架构中，依赖关系管理的两个重要方面是将大型代码工件传输到客户端会降低性能，以及计算资源开销。理想情况下，您的组织需要规定如何维护分布式前端架构中的依赖关系。

强制维护依赖关系的三种可行策略是使用诸如导入地图和模块联合之类的 Web 标准。其他方法是反模式，因为它们违反了分布式架构的基本原理。

尽可能不分享

nothine-nothine 方法假设独立软件工件之间根本不应共享任何依赖关系，或者至少在集成或运行时不应该共享。这意味着，如果两个微前端依赖于同一个库，则每个微前端都必须在构建时在库中烘焙并单独发货。此外，每个微前端都必须验证该库不会污染全局命名空间和共享资源。

这会导致裁员，但这是一种有意识的权衡，具有最大的灵活性。由于没有共享运行时依赖关系，因此团队可以最大限度地灵活地以他们认为有用的任何方式发展软件，前提是他们在解决方案范围内这样做，并且不违反任何接口合同。

在微前端遵循无共享原则的平台上，尽可能保持微前端的轻量化非常重要。它要求开发人员熟练而勤奋地优化微前端以提高性能，并且不会为了开发者体验而牺牲用户体验。

当你共享代码时

当您决定共享某些代码时，可以将其作为库或运行时模块共享。例如，前端核心团队通过 CDN 提供供微前端使用的库。业务价值团队可以在运行时加载库，也可以使用软件包存储库来发布他们的库。Micro-Frontend 团队可以在构建时针对打包库的特定版本进行开发，类似于使用混合框架的移动应用程序。

第三种选择是使用私有包注册表来支持公共库的构建时集成。这降低了库合约中的重大更改在运行时引发错误的风险。但是，这种更为保守的方法需要更多的管理，才能将所有微前端与较新的库版本同步。

为了缩短页面加载时间，微前端可以将库依赖项外部化，以便从 CDN（例如 Amazon）的缓存区块中加载。 CloudFront

为了管理运行时依赖关系，微前端可以使用导入映射（或诸如之类的库`System.js`）来指定每个模块在运行时的加载位置。webpack Module Federation 是另一种指向远程模块托管版本并解决独立微前端之间常见依赖关系的方法。

另一种方法是通过向发现端点发出初始请求来促进导入映射的动态加载。

共享状态

为了减少微前端的耦合，重要的是要避免在同一个视图下从所有微前端访问的全局状态管理，类似于单片架构。例如，让所有微前端都能访问全球 Redux 商店可以增加耦合。

消除共享状态的一种模式是将其封装在微前端中，并如前所述，与异步消息进行通信。

在绝对必要时，为全局状态引入定义明确的接口，并选择只读共享以避免意外行为：

- 当存在垂直分割时，您可以使用 URL 组件和浏览器存储来访问主机环境中的信息。
- 当你进行混合拆分时，你还可以使用 DOM 标准的自定义事件或 JavaScript 库，例如事件发射器或双向流，将信息传递给微前端。

如果您需要跨微前端共享多条信息，我们建议您重新审视微前端边界。共享需求可能是由于业务发展或初始设计不合标准造成的。

也可以使用服务器端会话，其中每个微前端都使用会话标识符获取所需的数据。为了减少耦合，必须消除共享状态并将微前端特定的会话数据分开。

框架和工具

不乏前端框架，比如 Angular 和 Next.js，但它们中的大多数并不是在创建时考虑微前端的。因此，它们有时缺少应对微前端架构挑战的机制。

一般框架注意事项

本指南的目的不是推荐或比较各个框架。由于多个微前端通常在同一个Web应用程序页面上运行，因此加载和运行时性能是主要关注的问题。选择一个尽可能减少开销的框架很重要。

框架是根据渲染层划分的：

- 客户端渲染 (CSR)
- 服务器端渲染 (SSR)

前端架构包括其他功能，例如静态站点生成 (SSG)。但是，SSG 只能执行一次。微前端主要是在运行时组成的，因此 CSR 和 SSR 是主要选项。

客户端渲染

对于企业社会责任，有两种流行的选择：

- 单个 SPA 框架
- 模块联合

Single SPA 是构成微前端的轻量级选择。它解决了微前端架构中最常见的挑战，例如在同一页面中组合多个微前端并避免依赖冲突。

Module Federation 最初是一个插件，由 webpack 5 提供，它解决了微前端架构中的绝大多数挑战，包括跨不同工件的依赖关系管理。Module Federation 2.0 原生可与 Rspack、webpack、esbuild 配合使用，现在也可以使用。 JavaScript

考虑完全不使用框架。根据 caniuse.com 的数据，现代浏览器的总体市场份额为98%，它们本身就提供诸如自定义元素之类的功能，它们足以满足微前端应用程序的需求。必要时，将自定义元素与轻量级库结合使用，以解决事件传播、国际化或其他特定问题。

服务器端渲染

在SSR方面，两个主要选项更为复杂：

- 采用诸如 Next.js 之类的现有框架，并应用使用模块联合的微前端原则。
- 使用 HTML-交换代表微前端的 HTML 片段，并在运行时将这些片段组合over-the-wire 到模板中。这种方法的一个例子是 Podium。

API 集成 – 前端的后端

前端后端 (BFF) 模式通常用于微服务环境。在微前端的背景下，BFF 是一种属于微前端的服务器端服务。并非所有的微前端都需要有 BFF。但是，如果您使用的是 BFF，则它必须在相同的有界上下文中运行，并且不能在其他有界上下文之间共享。

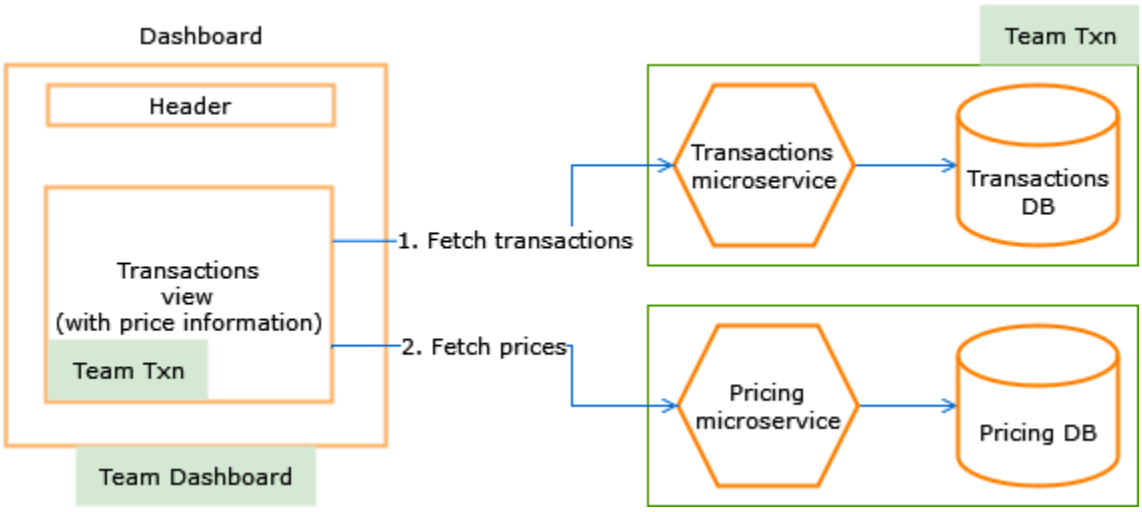
与传统服务不同，BFF 不遵循域名模型。相反，它是微前端的 API 层，用于在数据到达客户端之前对其进行预处理。这有用的领域包括：

- 对私有 API 的授权
- 汇总来自不同来源的数据
- 转换数据以减少网络负载并简化客户端对数据的消耗

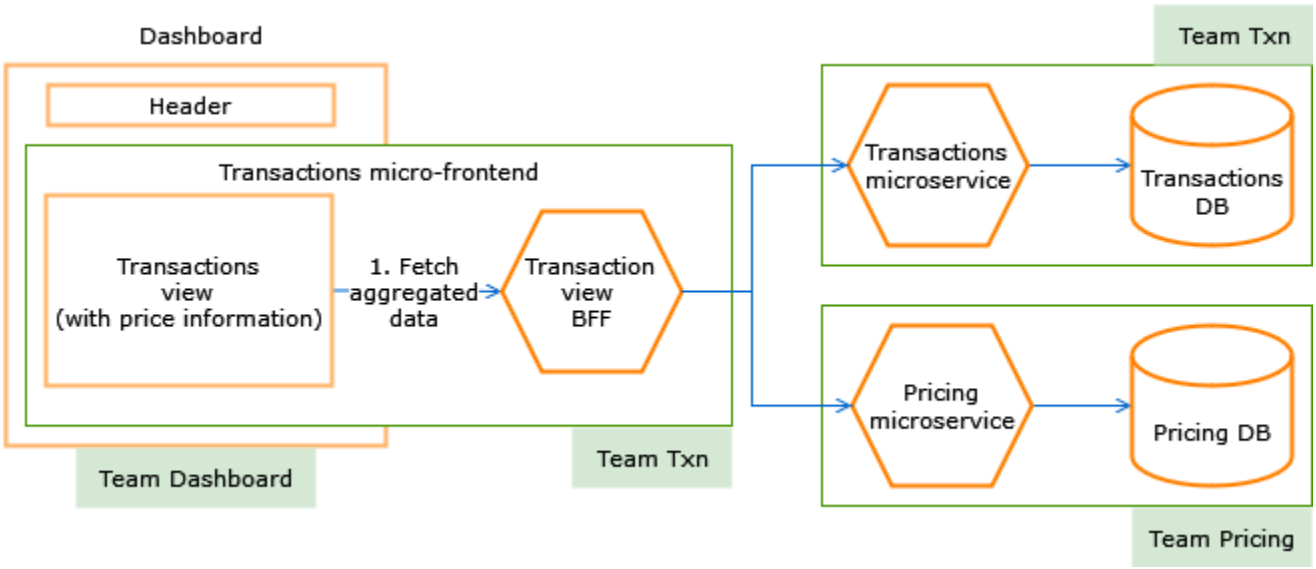
因此，BFF 归微前端所有，而不是域服务层所有。可以使用以下方法部署 BFF：

- AWS AppSync GraphQL API
- 一组 AWS Lambda 函数
- 作为在 Amazon ECS、Amazon EKS 或 AWS 上运行的容器 AppRunner

下图显示，如果没有 BFF 模式，微前端必须连接到各个微服务 API 端点才能获取和聚合数据。



取而代之的是，使用下图中的 BFF 模式，微前端可以与自己的后端通信并获取聚合数据。



团队可以为不同的渠道（例如移动、网络或特定视图）开发 BFF，并要求通过减少闲聊来优化后端互动。

样式和 CSS

层叠样式表 (CSS) 是一种用于集中确定文档呈现方式的语言，而不是对文本和对象进行硬编码格式化。该语言的级联功能旨在通过使用继承来控制样式之间的优先级。当你在微前端上工作并创建管理依赖关系的策略时，该语言的级联功能可能是一个挑战。

例如，两个微前端共存于同一页面上，每个微前端都为 HTML 元素定义了自己的样式。`body`如果每个文件都获取自己的 CSS 文件并使用`style`标签将其附加到 DOM，则如果它们都定义了常见的 HTML 元素、类名或元素 ID，则该 CSS 文件将覆盖第一个文件。有不同的策略可以处理这些问题，具体取决于你为管理样式而选择的依赖策略。

当前，平衡性能、一致性和开发者体验的最流行的方法是开发和维护设计系统。

设计系统 — 一种共享的方法

这种方法使用系统在适当时共享样式，同时支持偶尔出现的差异，以平衡一致性、性能和开发者体验。设计系统是以明确标准为指导的可重复使用的组件的集合。设计系统开发通常由一个团队推动，由多个团队提供意见和贡献。实际上，设计系统是一种共享可以导出为 JavaScript 库的低级元素的方法。Micro-Frontend 开发人员可以将该库用作依赖项，通过组合预制的可用资源来构建简单的接口，也可以作为创建新接口的起点。

以需要表单的微前端为例。典型的开发者体验包括使用设计系统中可用的预制组件来撰写文本框、按钮、下拉列表和其他用户界面元素。开发人员不需要为实际组件编写任何样式，只需为它们的外观编写样式即可。要构建和发布的系统可以使用 webpack Module Federation 或类似的方法将设计系统声明为外部依赖项，这样就可以在不包括设计系统的情况下打包表单的逻辑。

然后，多个微前端可以做同样的事情来解决共同的问题。当团队开发可在多个微前端之间共享的新组件时，这些组件将在成熟后添加到设计系统中。

设计系统方法的一个主要优点是高度的一致性。虽然微前端可以编写样式，偶尔还会覆盖设计系统中的样式，但几乎没有必要这样做。主要的低级元素不经常更改，它们提供了默认情况下可扩展的基本功能。另一个优势是性能。有了好的构建和发布策略，您就可以生成最少的共享包，由应用程序 shell 进行检测。当按需异步加载多个特定于微前端的捆绑包时，您可以进一步改进，而网络带宽占用空间最小。最后但并非最不重要的一点是，开发者体验非常理想，因为人们可以专注于构建丰富的界面，而无需重新发明轮子（例如每次需要在页面中添加按钮时编写 JavaScript 和CSS）。

缺点是，任何类型的设计系统都是依赖关系，因此必须对其进行维护，有时还必须更新。如果多个微前端需要共享依赖项的新版本，则可以使用以下任一方法：

- 一种编排机制，可以偶尔获取该共享依赖项的多个版本而不会发生冲突
- 将所有受抚养人转移到使用新版本的共享策略

例如，如果所有微前端都依赖于设计系统的 3.0 版本，并且有一个名为 3.1 的新版本可以共享使用，则可以为所有微前端实现功能标志，以便以最小的风险进行迁移。有关更多信息，请参阅 “功能标志” 部分。另一个潜在的缺点是，设计系统通常解决的不仅仅是造型。它们还包括 JavaScript 实践和工具。这些方面需要通过辩论和合作达成共识。

实施设计系统是一项不错的长期投资。这是一种流行的方法，任何从事复杂前端架构工作的人都应该考虑使用它。它通常需要前端工程师以及产品和设计团队进行协作并定义相互交互的机制。安排时间以达到所需状态很重要。获得领导层的赞助也很重要，这样人们才能长期建造可靠、维护良好且性能良好的东西。

完全封装的 CSS-一种不共享的方法

每个微前端都使用约定和工具来克服 CSS 的级联功能。例如，确保每个元素的样式始终与类名而不是元素的 ID 相关联，并且类名始终是唯一的。通过这种方式，所有内容都限于单个微前端，并且将不必要的冲突风险降至最低。尽管有些工具通过使用将样式捆绑在一起，但应用程序外壳通常负责在微前端的样式加载到DOM中。 JavaScript

不共享任何内容的主要优点是降低了在微前端之间引入冲突的风险。另一个优势是开发者的经验。每个微前端都与其他微前端没有任何共享。单独发布和测试更简单、更快捷。

不共享方法的一个主要缺点是可能缺乏一致性。目前尚无评估一致性的系统。即使目标是复制共享的内容，但在平衡发布速度和协作速度时也变得具有挑战性。常见的缓解措施是创建衡量一致性的工具。例如，你可以创建一个系统，使用无头浏览器自动截取页面中呈现的多个微前端的屏幕截图。然后，您可以在发布之前手动查看屏幕截图。但是，这需要纪律和治理。有关更多信息，请参阅 “平衡自主权与对齐方式” 部分。

根据用例的不同，另一个潜在的缺点是性能。如果所有微前端都使用大量的样式，则客户必须下载大量重复的代码。这将对用户体验产生负面影响。

这种不共享的方法应仅适用于仅涉及几个团队的微前端架构，或者可以容忍低一致性的微前端。在组织研究设计系统时，这也可能是自然而然的初始步骤。

共享全局 CSS — 一种共享的方法

通过这种方法，所有与样式相关的代码都存储在中央存储库中，贡献者通过处理 CSS 文件或使用 Sass 等预处理器为所有微前端编写 CSS。进行更改时，构建系统会创建一个 CSS 包，该包可以托管

在 CDN 中，并由应用程序 shell 包含在每个微前端中。Micro-Frontend 开发人员可以通过本地托管的应用程序 shell 运行其代码来设计和构建应用程序。

除了降低微前端之间冲突风险的明显优势外，这种方法的优势还在于一致性和性能。但是，将样式与标记和逻辑分开会使开发人员更难理解样式是如何使用的、它们如何演变以及如何弃用它们。例如，引入新的类名可能比了解现有类以及编辑其属性的后果要快。创建新类名的缺点是捆绑包大小的增长，这会影响性能，并且可能会在用户体验中引入不一致之处。

虽然共享的全局 CSS 可以作为 monolith-to-micro-frontends迁移的起点，但对于涉及一两个以上团队协作的微前端架构来说，它很少有好处。我们建议尽快投资设计系统，并在设计系统开发过程中实施不共享的方法。

组织和工作方式

与所有架构策略一样，微前端的影响远远超出了组织选择实施的技术。构建微前端应用程序的决定必须与业务、产品、组织、运营甚至文化保持一致（例如，赋予团队权力和去中心化决策）。作为回报，这种微前端架构支持真正敏捷、产品驱动的开发，因为它可以显著减少原本独立的团队之间的沟通开销。

敏捷开发

近年来，敏捷软件开发的想法已变得如此普遍，以至于几乎每个组织都声称要敏捷工作。虽然敏捷的确切定义超出了该策略的范围，但值得回顾一下与微前端开发相关的关键要素。

敏捷范式的基础是《敏捷宣言》（2001），它假设了四个主要原则（例如，“个人和互动胜于流程和工具”）和十二项原则。诸如Scrum和规模化敏捷框架（SaFe）之类的流程框架是围绕敏捷宣言出现的，并已进入日常实践。但是，它们背后的哲学在很大程度上被误解或忽视了。

在微前端架构的背景下，必须遵循以下敏捷原则：

- “经常交付可运行的软件，从几周到几个月不等，优先考虑较短的时间范围。”

  这一原则强调了分阶段工作并尽可能定期和频繁地将软件交付到生产环境是多么重要。从技术角度来看，这是指持续集成和持续交付 (CI/CD)。在 CI/CD 中，用于构建、测试和部署的工具和流程是每个软件项目不可或缺的一部分。该原则还意味着运行时基础设施和运营责任必须归团队所有。这种所有权在分布式系统中尤其重要，在分布式系统中，独立子系统对基础设施和运营的要求可能截然不同。
- “围绕积极进取的个人构建项目。为他们提供所需的环境和支持，并相信他们能把工作做好。”

  “最好的架构、要求和设计来自于自组织团队。”

  这两项原则都强调所有权、独立性和 end-to-end 责任感的好处。当（且仅当）团队真正拥有自己的微前端时，微前端架构才会成功。从构思到设计和实施，再到交付和运营，电子nd-to-end责任确保了团队能够真正行使所有权。无论是在技术上还是在组织上，团队都需要这种独立性，这样团队才能对战略方向拥有自主权。我们不建议在使用瀑布式开发模式的集中式组织中使用微前端平台。

团队组成和规模

软件团队要行使所有权，就必须在组织规定的界限内进行自我管理，包括团队交付的方式和内容。

为了提高效率，团队必须能够独立交付软件，并有权决定交付软件的最佳方式。从外部产品经理那里获得功能要求或从外部设计师那里获得用户界面设计但不参与这些项目的规划的团队，不能被视为自主团

队。这些功能可能违反现有合同或功能。此类违规行为将需要进一步的讨论和谈判，有可能延迟交付，并在团队之间引入不必要的冲突。

同时，团队不应变得太大。虽然规模更大的团队拥有更多的资源并且可以容纳个人缺席，但每增加一个新成员，沟通的复杂性就会呈指数级增长。无法说出一个普遍有效的最大队伍规模。项目所需人员数量取决于团队成熟度、技术复杂性、创新速度和基础设施等因素。例如，亚马逊遵循双披萨规则：一支规模太大而无法吃两个披萨的队伍应该分成较小的队伍。这可能是一个挑战。分裂应沿着自然界限进行，并应赋予每个团队对其工作的自主权和所有权。

DevOps 文化

DevOps 指一种软件工程实践，其中开发生命周期的各个步骤从组织和技术角度紧密整合。与普遍的看法相反， DevOps 这在很大程度上与文化和思维方式有关，而很少涉及角色和工具。

传统上，软件组织会有专家团队，例如设计、实施、测试、部署和运营。每当一个团队完成工作时，他们就会将项目移交给下一个团队。但是，通过专业团队的孤岛交付软件会导致移交过程中的摩擦。同时，当专家被迫以狭隘的重点工作时，他们缺乏邻近领域的知识，对产品也没有系统的看法。这些缺陷可能导致软件产品的一致性降低。

例如，当软件架构师设计的解决方案将由不同团队中的某人实施时，他们可能会忽略实现的固有方面（例如依赖关系不匹配）。然后，开发人员走捷径（例如猴子补丁），或者在架构师和开发团队之间启动正式 back-and-forth 的补丁。由于管理这些流程的开销，开发不再是敏捷的（从灵活、自适应、增量和非正式的意义上讲）。

尽管该术语 DevOps 主要涉及文化，但它意味着在实践中使之成为 DevOps 可能的技术和过程。DevOps 与 CI/CD 密切相关。当开发者完成软件增量实现后，他们会将其提交到版本控制系统（例如 Git）。传统上，构建系统随后会构建和集成软件，然后在或多或少的统一和集中化流程中对其进行测试和发布。借助 CI/CD，软件的构建、集成、测试和发布是固有的、自动化的。理想情况下，通过专门为给定项目量身定制的配置文件，该过程是软件项目本身的一部分。

尽可能多的步骤是自动化的。例如，应减少手动测试做法，因为几乎所有类型的测试都可以实现自动化。以这种方式设置项目后，每天可以充满信心地多次交付软件产品的更新。另一种支持的技术 DevOps 是基础设施即代码 (IaC)。

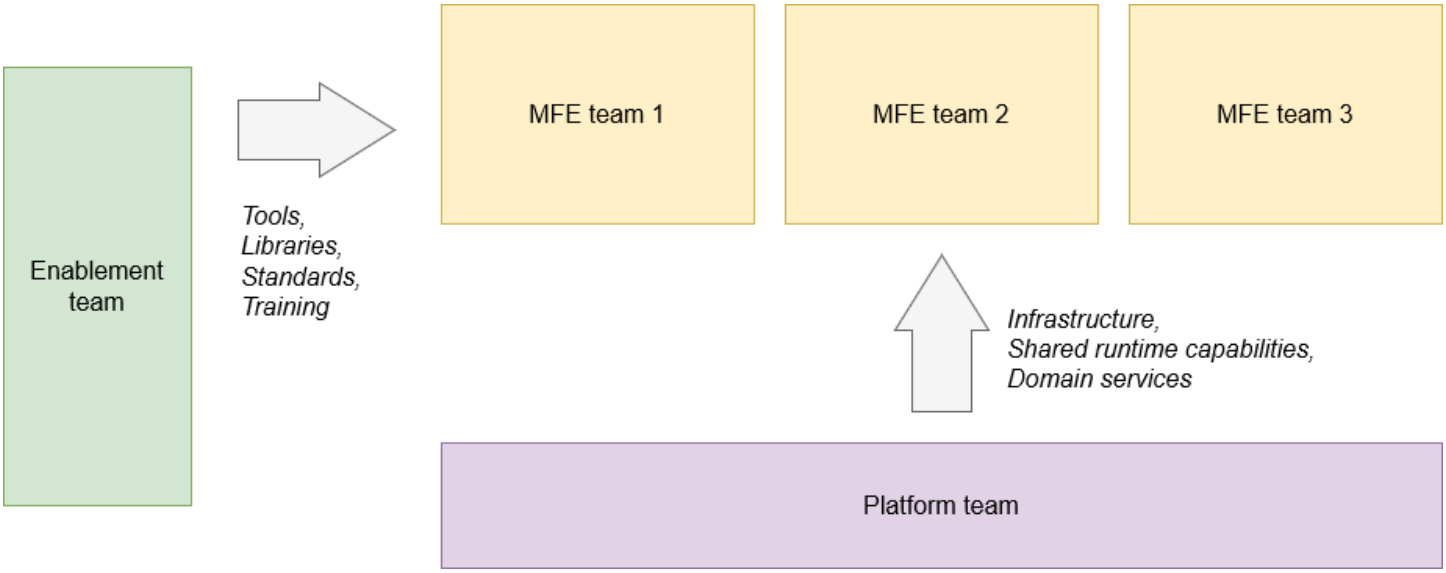
传统上，设置和维护 IT 基础架构需要手动安装和维护硬件（在数据中心设置电缆和服务器）和操作软件。这是必要的，但它有很多缺点。安装既耗时又容易出错。硬件往往过度配置或配置不足，导致开支过高或性能降低。通过使用 IaC，您可以通过配置文件描述 IT 系统的基础架构要求，通过该文件可以自动部署和更新云服务。

所有这些与微前端有什么关系？ DevOps、CI/CD 和 IaC 是微前端架构的理想补充。微前端的好处取决于快速无摩擦的交付流程。只有在团队 end-to-end 负责任地拥有软件项目的环境中， DevOps 文化才能蓬勃发展。

跨多个团队协调微前端开发

在跨多个跨职能团队扩展微前端开发时，会出现两个问题：首先，团队开始对范式进行自己的解释，做出框架和库选择，并创建自己的工具和帮助程序库。其次，完全自主的团队必须对诸如低级基础设施管理之类的通用功能负责。因此，在多团队的微前端组织中再引入两个团队是有意义的：支持团队和平台团队。这些概念在具有分布式系统的现代 IT 组织中被广泛采用，并且在 Tea m Topologies 中有详细记录。

下图显示了支持团队向三个微前端团队提供工具、库、标准和测试。平台团队为这三个微前端团队提供基础架构、共享运行时功能和域服务。



平台团队通过将微前端团队从无差别的繁重工作中解放出来，为他们提供支持。这种支持可能包括基础设施服务，例如容器运行时、CI/CD 管道、协作工具和监控。但是，成立平台团队不应导致组织将开发与运营分开。情况恰恰相反：平台团队提供工程产品，而微前端团队对平台上的服务拥有所有权和运行时责任。

支持团队通过专注于治理和确保微前端团队之间的一致性来提供支持。（平台团队不应参与其中。）支持团队维护共享资源，例如用户界面库，并创建框架选择、性能预算和互操作性惯例等标准。同时，它为新团队或团队成员提供应用治理所定义的标准和工具的培训。

部署

微前端团队自主权的北极星是拥有一个自动化管道，其生产路径独立于其他微前端团队。遵循不共享原则的团队可以实现独立的管道。共享库或依赖平台团队的团队必须决定如何管理部署管道中的依赖关系。

通常，每个管道都执行以下操作：

- 构建前端资产
- 将资产部署到主机以供使用
- 确保更新注册表和缓存，以便向客户交付新版本

实际的管道步骤因技术堆栈和页面合成方法而异。

对于客户端组合，这意味着将应用程序包上传到托管存储桶，然后通过在 CDN 上缓存来发布以供使用。在服务工作线程中使用浏览器缓存的应用程序还应实现更新服务工作线程缓存的方法。

对于服务器端组合，这通常意味着部署服务器组件的新版本并更新微前端注册表以使新版本可被发现。您可以使用蓝/绿或灰色部署模式来逐步推出新版本。

治理

多个角色通常适用于微前端，每个角色在不同的限制下工作，以实现共同的业务目标。虽然人与人之间的沟通和协作是成功的关键，但过度沟通和实施过于复杂的流程会减慢开发周期。这会导致士气低落，并降低质量门槛。

通过使用多个团队实现微前端的最成功的公司创建了平衡自主权与一致性的机制。它们使决策者能够在当地采取行动，并仅在需要时才按等级进行升级。机制包括以下内容：

- API 合同
- 使用事件进行交叉互动
- 在自治与协调之间取得平衡
- 功能标志
- 服务发现

API 合同

每个微前端都是一个能够封装观点、逻辑和复杂性的系统。跨领域问题通常包括以下几点：

- 设计系统-以库形式 UIs分布式开发的工具
- 构图 – 微前端需要与应用程序 shell 交互才能渲染和继承其上下文的方式
- 逻辑处理-与之交互 APIs 以处理持久状态
- 与其他微前端的交互 ——例如发布和使用事件或从一个微前端导航到另一个微前端的场景

为了加快使用和故障排除，通常需要投资标准化这些接口的声明和记录方式，包括微前端依赖关系。由人类策划的维基是一个良好的开端。一种更具可扩展性的方法是将这些信息作为结构化元数据存储在代码中。然后，您可以通过使用自动化来跟踪历史更改并提供全文搜索，将其集中起来以供使用。

当微前端涉及大量团队时，你需要一个策略来协调团队之间的关系。必须以统一的方式共享 API 合同，因为它可以减少通信开销并改善开发者体验。

OpenAPI 是一种 HTTP 的规范语言 APIs ，它支持以统一的方式定义 API 接口和合约。你可以在 Amazon API Gate APIs way 中使用 OpenAPI 来实现 R EST。您还可以使用各种开源框架，这些框架可以托管在容器或虚拟机中。一个显著的优势是 OpenAPI 可以自动生成格式一致的文档，因此多个团队可以用最少的初始投资共享知识。

当多个团队在微前端上工作时，他们通常会组成小组。在这些小组中，人们可以见面并互相学习，同时思考大局并为之做出贡献。这些举措通常定义和记录所有权界限，讨论跨部门问题，并尽早发现任何重复努力以解决常见问题。

使用事件进行交叉互动

在某些情况下，多个微前端可能需要相互交互才能对状态变化或用户操作做出反应。例如，页面上的多个微前端可以包含可折叠的菜单。当用户选择按钮时，会出现一个菜单。当用户点击其他任何地方（包括在不同的微前端中呈现的另一个菜单）时，该菜单会被隐藏。

从技术上讲，诸如 Redux 之类的共享状态库可以由多个微前端使用并由 shell 进行协调。但是，这会在应用程序之间造成显著的耦合，从而导致代码更难测试，并且可能会降低渲染过程中的性能。

一种常见、有效的方法是开发一个事件总线，该总线作为库分发，由应用程序 shell 编排，并由多个微前端使用。通过这种方式，每个微前端都异步发布和监听特定事件，其行为仅基于自己的内部状态。然后，多个团队可以维护一个共享的 wiki 页面，该页面描述了用户体验设计师同意的事件和文档行为。

在事件总线示例的实现中，下拉列表组件使用共享总线发布一个有效载荷为`drop-down-open-menu`的名为的事件。`{"id": "homepage-aboutus-button"}`该组件向`drop-down-open-menu`事件添加了一个侦听器，以确保在针对新 ID 触发事件时，会呈现下拉列表组件以隐藏其可折叠部分。通过这种方式，微前端可以通过提高性能和更好的封装来异步应对变化，从而使多个团队更容易设计和测试行为。

我们建议使用现代浏览器原生 APIs 实现的标准，以提高简单性和可维护性。MDN Event 参考提供了有关在客户端渲染的应用程序中使用事件的信息。

在自治与协调之间取得平衡

微前端架构强烈偏向于团队自主权。但是，重要的是要区分能够支持灵活性和多种方法来解决问题的领域，以及需要标准化才能实现一致的领域。高级领导和架构师必须尽早确定这些领域并确定投资的优先顺序，以平衡微前端的安全性、性能、卓越运营和可靠性。找到这种平衡涉及以下方面：微前端的创建、测试、发布和记录、监控和警报。

创建微前端

理想情况下，所有团队都要紧密合作，以最大限度地提高最终用户性能。实际上，这可能很难，而且可能需要付出更多的努力。我们建议从一些书面指导方针开始，多个团队可以通过公开透明的辩论为之做出贡献。然后，团队可以逐步采用 Cookiecutter 软件模式，该模式支持创建工具，为项目搭建提供统一的脚手架方式。

使用这种方法，你可以提出意见和约束。缺点是，这些工具需要大量投资来创建和维护，并确保在不影响开发人员工作效率的情况下快速解决障碍。

End-to-end 测试微前端

单元测试可以留给所有者。我们建议尽早实施策略，对在独特 shell 上运行的微前端进行交叉测试。该策略包括能够在生产版本之前和之后测试应用程序。我们建议为技术和非技术人员开发流程和文档，以便他们手动测试关键功能。

重要的是要确保更改不会降低功能性或非功能性的客户体验。理想的策略是逐步投资于自动测试，包括关键功能和安全性和性能等架构特征。

发布微前端

每个团队可能都有自己的方式来部署代码、发表意见和自己的基础架构。维护此类系统的复杂性通常会起到威慑作用。相反，我们建议尽早进行投资，以实施可通过共享工具强制执行的共享策略。

使用所选的 CI/CD 平台开发模板。然后，团队可以使用预先批准的模板和共享基础架构来发布对生产环境的更改。您可以尽早开始投资这项开发工作，因为经过最初的测试和整合后，这些系统很少需要重大更新。

日志记录和监控

每个团队可以有不同的业务和系统指标，他们想要跟踪这些指标以用于运营或分析目的。Cookiecutter 软件模式也可以在这里应用。事件的交付可以抽象出来，并作为一个库提供，供多个微前端使用。为了平衡灵活性和提供自主权，请开发用于记录自定义指标和创建自定义仪表板或报告的工具。该报告促进了与产品负责人的密切合作，减少了最终客户的反馈循环。

通过标准化交付，多个团队可以协作跟踪指标。例如，电子商务网站可以跟踪用户旅程，从 “产品详情” 微前端到 “购物车” 微前端，再到 “购买” 微前端，以衡量参与度、流失率和问题。如果每个微前端都使用单个库来记录事件，则可以整体使用这些数据，对其进行全面探索，并确定有洞察力的趋势。

警报

与日志和监控类似，警报也受益于标准化以及一定程度的灵活性。不同的团队对功能警报和非功能警报的反应可能有所不同。但是，如果所有团队都有一种统一的方法来根据在共享平台上收集和分析的指标来启动警报，那么企业就可以识别跨团队的问题。此功能在事件管理事件期间非常有用。例如，可以通过以下方式启动警报：

- 特定浏览器版本的 JavaScript 客户端异常数量增加
- 在给定阈值内，渲染时间明显降低
- 使用特定 API 时增加了 5xx 状态码的数量

根据系统的成熟度，您可以平衡基础架构不同部分的工作，如下表所示。

收养	研究和开发	Ascent	成熟度
创建微前端。	实验、记录和分享学习成果。	投资购买工具来搭建新的微前端。宣传收养。	整合脚手架工具。推动采用。
端到端测试微前端。	实现手动测试所有相关微前端的机制。	投资购买用于自动安全和性能测试的工具。调查功能标志和服务发现。	整合用于服务发现、生产环境测试和自动 end-to-end测试的工具。
发布微前端。	投资共享的 CI/CD 基础架构和自动化的多环境发布。宣传收养。	整合 CI/CD 基础架构的工具实施手动回滚机制。推动采用。	创建机制，根据系统和业务指标及警报启动自动回滚。
观察微前端性能。	投资共享的监控基础设施和库，以实现系统和业务事件的持续记录。	整合用于监控和警报的工具。实施跨团队仪表板，以监控总体运行状况并改善事件管理。	标准化日志架构。针对成本进行优化。根据复杂的业务指标实施警报。

功能标志

可以在微前端中实现功能标志，以便在多个环境中协调测试和发布功能。功能标记技术包括将决策集中在基于 Boolean 的商店中，并以此为基础驱动行为。它通常用于静默传播更改，这些更改可以一直隐藏到特定时刻，同时解锁新版本以获取原本会被屏蔽的新功能，从而降低团队速度。

以团队开发将在特定日期推出的微前端功能为例。该功能已准备就绪，但需要与独立发布的另一个微前端的更改一起发布。阻止两个微前端的发布将被视为一种反模式，部署后会增加风险。

取而代之的是，团队可以在数据库中创建一个布尔特征标志，供他们在渲染期间使用（可能是通过对共享功能标志 API 的 HTTP 调用）。团队甚至可以在测试环境中发布更改，在该环境中，布尔值设置为，`True`以便在启动生产之前验证跨项目的功能和非功能需求。

使用功能标志的另一个示例是实现一种机制，通过`QueryString`参数设置特定值或将特定的测试字符串存储在 cookie 中，从而覆盖标志的值。在发布日期之前，产品所有者可以对功能进行迭代，而不会阻止其他功能的发布或错误修复。在给定日期，更改数据库上的标志值会立即使更改在生产中可见，而无需跨团队协调发布。功能发布后，开发团队会清理代码以删除旧行为。

其他用例包括发布基于上下文的功能标记系统。例如，如果一个网站以多种语言为客户提供服务，则某项功能可能仅适用于特定国家/地区的访问者。功能标志系统可能取决于发送国家/地区上下文的消费者（例如，通过使用 `Accept-Language` HTTP 标头），并且根据该上下文，可能会有不同的行为。

虽然功能标志是促进开发人员和产品所有者之间协作的有力工具，但它们依靠人们的勤奋来避免代码库的严重退化。在多个功能上保持标记处于活动状态可能会增加故障排除时的复杂性，增加 JavaScript 捆绑包的大小，并最终积累技术债务。常见的缓解活动包括以下内容：

- 在标记后面对每个功能进行单元测试以降低出现错误的可能性，这可能会在运行测试的自动 CI/CD 管道中引入更长的反馈循环
- 创建工具来衡量代码更改期间捆绑包大小的增加，在代码审查期间可以缓解这种情况

AWS 提供了一系列解决方案，用于使用亚马逊 CloudFront 函数或 Lambda @Edge 优化边缘的 A/B 测试。这些方法有助于降低集成解决方案或您用来维护假设的现有 SaaS 产品的复杂性。有关更多信息，请参阅 A/B 测试。

服务发现

在开发、测试和交付微前端时，前端发现模式可以改善开发体验。该模式使用描述微前端入口点的可共享配置。可共享配置还包括其他元数据，这些元数据用于使用金丝雀版本在每个环境中进行安全部署。

现代前端开发需要在开发过程中使用各种各样的工具和库来支持模块化。传统上，此过程包括将代码捆绑到可以托管在 CDN 中的单个文件中，目标是在运行时将网络调用保持在最低限度，包括初始加载（当应用程序在浏览器中打开时）和使用情况（当客户执行诸如选择按钮或插入信息之类的操作时）。

拆分捆绑包

Micro-Frontend 架构解决了由于单独捆绑大量功能而生成的非常大的捆绑包所导致的性能问题。例如，可以将一个非常大的电子商务网站捆绑成一个 6 MB 的 JavaScript 文件。尽管进行了压缩，但该文件的大小可能会对用户加载应用程序和从边缘优化的 CDN 下载文件时的体验产生负面影响。

如果您将应用程序拆分为主页、产品详情和购物车微前端，则可以使用捆绑机制生成三个单独的 2 MB 捆绑包。当用户使用主页时，此更改可能会将首次加载的性能提高 300%。只有当用户访问某件商品的产品页面并决定购买时，才会异步加载产品或购物车微前端捆绑包。

许多框架和库都基于这种方法可用，这对客户和开发人员都有好处。要识别可能导致代码中依赖关系解耦的业务边界，您可以将不同的业务职能映射到多个团队。分布式所有权引入了独立性和敏捷性。

拆分构建包时，您可以使用配置来映射微前端并驱动初始加载和加载后导航的编排。然后，可以在运行时而不是在构建期间使用该配置。例如，客户端前端代码或服务器端后端代码可以对 API 进行初始网络调用，以动态获取微前端列表。它还会获取合成和集成所需的元数据。您可以配置故障转移策略和缓存以提高可靠性和性能。映射微前端有助于使先前部署的微前端能够被先前部署的、由 shell 应用程序编排的微前端发现。

Canary 发布

金丝雀版本是一种成熟且流行的微服务部署模式。Canary 版本将版本的目标用户分为多个组，发布会逐渐进行更改，而不是立即替换（也称为蓝/绿部署）。金丝雀发布策略的一个例子是向10%的目标用户推出新的更改，每分钟增加10%，总持续时间为10分钟才能达到100%。

金丝雀版本的目标是尽早获得有关变更的反馈，监控系统以减少任何问题的影响。实现自动化后，可以由内部系统监控业务或系统指标，该系统可以停止部署或开始回滚。

例如，更改可能会引入一个错误，该错误在发布的最初几分钟内会导致收入损失或性能下降。自动监控可以启动警报。使用服务发现模式，该警报可以停止部署并立即回滚，仅影响20%的用户，而不是100%的用户。企业受益于问题范围的缩小。

有关使用 DynamoDB 作为存储来实现 REST 管理 API 的架构示例，请参阅 AWS 上的前端服务发现解决方案。 GitHub使用 AWS CloudFormation 模板将架构集成到您自己的 CI/CD 管道中。该解决方案包括一个 REST Consumer API，用于将该解决方案与您的前端应用程序集成。

你需要平台团队吗？

一些公司的团队负责拥有和维护其他团队在微前端上工作时采用的代码、基础架构和流程。常见职责包括：

- 创建和维护可用于包含微前端的存储库的 CI/CD 管道。生成和测试代码更改，然后将其发布到多个环境中。
- 创建和维护与可观测性相关的工具，例如共享仪表板、警报机制和系统以应对问题。
- 创建和维护用于事件处理、共享服务使用和第三方依赖项的共享库。
- 创建和维护能够持续监控非功能性质的工具，例如系统的性能、安全性和可靠性。
- 创建和维护设计系统。
- 创建、维护和支持微前端系统的应用程序 shell。

根据项目的规模，您可以使用以下方法之一来管理这些职责：

- 创建一个专门的平台团队，其唯一职责是开发共享工具。
- 创建一个由来自多个团队的成员组成的小组。小组成员将时间分散在处理微前端和开发共享工具之间。这也被称为老虎队。

虽然老虎团队方法是保持以客户为中心的有效方法，但如果项目获得吸引力和责任，老虎团队通常会演变为平台团队。对于平台团队和老虎团队来说，从事微前端工作的最成功的公司组成了这些团队，因此具有多种背景和技能的多个人可以做出贡献。团队成员可能包括后端工程师、前端工程师、用户体验 (UX) 设计师和技术产品经理。这种多样性促使人们持续参与健康的辩论，并以简单为设计理念。

后续步骤

本指南涵盖了架构和组织模式、关键决策的权衡以及与微前端相关的治理问题。这些表格从以下几个方面总结了本文档中讨论的做法的权衡取舍：

- A@@ ut onomy – 每个微前端团队能够独立改进其实现并向最终用户发布。
- 一致性 – 应用程序的整体体验，其中每个微前端的行为都符合预期。高一致性意味着微前端与应用程序的其余部分保持一致，并且不会损害整个应用程序的用户体验。
- 复杂性 – 实施和测试微前端、整个应用程序和治理控制所需的基础架构、代码和工作量。

练习	自治	一致性	复杂性
使用微前端而不是单片应用程序进行构建	高	中	高

代码共享实践	自治	一致性	复杂性
什么都不分享	高	低	低
分享跨领域问题	中	高	中
共享业务逻辑	低	高	中
在构建时通过库共享	中	高	低

代码共享实践	自治	一致性	复杂性
在运行时共享	高	高	高

微前端发现实践	自治	一致性	复杂性
在应用程序构建期间进行配置	低	高	低
服务器端发现	高	高	中
客户端（运行时）发现	高	高	中

查看作文练习	自治	一致性	复杂性
服务器端合成	高	中	高
边缘构图	中	中	高
客户端合成	高	中	中

要详细了解本指南中引入的概念，请参阅 “资源” 部分。

资源

- 上下文中的微前端
- 领域驱动的设计
- EDA 视觉效果
- 前端发现
- 开启前端服务发现 AWS
- 敏捷宣言
- MDN 事件参考
- OpenAPI

贡献者

以下人员为本指南做出了贡献。

- 首席解决方案架构师 Matteo Figus， AWS
- 亚历山大·根舍，高级解决方案架构师， AWS
- Harun Hasdal，高级解决方案架构师， AWS
- Luca Mezzalira，英国Serverless首席上市专家解决方案架构师， AWS

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	—	2024年7月12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：将您的本地 Oracle 数据库迁移到兼容 Amazon Aurora PostgreSQL 的版本。
- 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：在中将您的本地 Oracle 数据库迁移到适用于 Oracle 的亚马逊关系数据库服务 (Amazon RDS) AWS Cloud。
- 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：将您的客户关系管理 (CRM) 系统迁移到 Salesforce.com。
- 更换主机（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：在中的 EC2 实例上将您的本地 Oracle 数据库迁移到 Oracle AWS Cloud。
- 重新定位（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。您可以将服务器从本地平台迁移到同一平台的云服务。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

参见托管服务。

ACID

参见原子性、一致性、隔离性、耐久性。

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。与主动-被动迁移相比，它更灵活，但需要更多的工作。

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

一个 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

参见人工智能。

AIOps

参见人工智能操作。

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

坏机器人

旨在破坏个人或组织或对其造成伤害的机器人。

BCP

参见业务连续性计划。

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。另请参见字节顺序。

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的应用程序版本（蓝色），在另一个环境中运行新的应用程序版本（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互动的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的网络爬虫。其他一些被称为恶意机器人的机器人旨在破坏个人或组织或对其造成伤害。

僵尸网络

被恶意软件感染并受单方（称为机器人牧民或机器人操作员）控制的机器人网络。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）。

破碎的玻璃通道

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 Well -Architected 指南中的 “实施破碎玻璃程序” 指示 AWS 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

参见AWS 云采用框架。

金丝雀部署

向最终用户缓慢而渐进地发布版本。当你有信心时，你可以部署新版本并全部替换当前版本。

CCoE

参见云卓越中心。

CDC

请参阅变更数据捕获。

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弹性。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

查看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。云计算通常与边缘计算技术相关。

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

组织迁移到以下阶段时通常会经历四个阶段 AWS Cloud：

- 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

参见配置管理数据库。

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 管道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)

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

配置偏差

对于工作负载，配置会从预期状态发生变化。这可能会导致工作负载变得不合规，而且通常是渐进的，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 is commonly described as a pipeline. 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

参见计算机视觉。

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

参见数据库定义语言。

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。

委托管理员

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。

后

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

参见环境。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出警报。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

在星型架构中，一种较小的表，其中包含事实表中有关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)

您用来最大限度地减少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

参见数据库操作语言。

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

参见灾难恢复。

漂移检测

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

参见开发价值流映射。

E

EDA

参见探索性数据分析。

EDI

参见电子数据交换。

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。与云计算相比，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)

组织之间自动交换业务文档。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密文的计算过程。

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

参见服务端点。

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

参见企业资源规划。

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

星形架构中的中心表。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。

失败得很快

一种使用频繁和增量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

参见分支。

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。

少量提示

在要求法学硕士执行类似任务之前，向其提供少量示例，以演示该任务和所需的输出。这种技术是情境学习的应用，模型可以从提示中嵌入的示例（镜头）中学习。对于需要特定格式、推理或领域知识的任务，Few-shot 提示可能非常有效。另请参见零镜头提示。

FGAC

请参阅精细的访问控制。

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，它使用连续的数据复制，通过更改数据捕获在尽可能短的时间内迁移数据，而不是使用分阶段的方法。目标是将停机时间降至最低。

FM

参见基础模型。

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

人工智能模型的子集，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工件，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 AI。

地理封锁

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，而基于主干的工作流程是现代的首选方法。

金色影像

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配置软件，并有助于提高设备制造运营的速度、可扩展性和生产力。

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

一项高级规则，可帮助管理各组织单位的资源、策略和合规性 (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。侦测性防护机制会检测策略违规和合规性问题，并生成警报以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

参见高可用性。

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。

抵制数据

从用于训练机器学习模型的数据集中扣留的一部分带有标签的历史数据。通过将模型预测与抵制数据进行比较，您可以使用抵制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

参见基础设施即代码。

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

参见工业物联网。

不可变的基础架构

一种为生产工作负载部署新基础架构，而不是更新、修补或修改现有基础架构的模型。不可变基础架构本质上比可变基础架构更一致、更可靠、更可预测。有关更多信息，请参阅 Well-Architected Framework 中的使用不可变基础架构 AWS 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

该术语由克劳斯·施瓦布（Klaus Schwab）于2016年推出，指的是通过连接、实时数据、自动化、分析和人工智能/机器学习的进步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置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。

IoT

参见物联网。

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。

大型语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 AI 模型。法学硕士可以执行多项任务，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

见 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参见字节顺序。

LLM

参见大型语言模型。

下层环境

参见环境。

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

参见分支。

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。恶意软件的示例包括病毒、蠕虫、勒索软件、特洛伊木马、间谍软件和键盘记录器。

托管服务

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。亚马逊简单存储服务 (Amazon S3) Service 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

参见迁移加速计划。

机制

一个完整的过程，在此过程中，您可以创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运行过程中自我增强和改进的循环。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

参见制造执行系统。

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 DevOps 人员和冲刺专业人员。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： EC2 使用 AWS 应用程序迁移服务重新托管向 Amazon 的迁移。

迁移组合评测（MPA）

一种在线工具，可提供信息，用于验证迁移到的业务案例。 AWS Cloud 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

用于将工作负载迁移到的方法 AWS Cloud。有关更多信息，请参阅此词汇表中的 7 R 条目和动员组织以加快大规模迁移。

ML

参见机器学习。

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，请参阅中的应用程序现代化策略。 AWS Cloud

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，请参阅中的评估应用程序的现代化准备情况 AWS Cloud。

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

参见迁移组合评估。

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？”

可变基础架构

一种用于更新和修改现有生产工作负载基础架构的模型。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

请参阅源站访问控制。

OAI

参见源访问身份。

OCM

参见组织变更管理。

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

参见运营集成。

OLA

参见运营层协议。

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

参见开放流程通信-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)

一份问题清单和相关的最佳实践，可帮助您理解、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 Well-Architecte AWS d Frame work 中的运营准备情况评估 (ORR)。

操作技术 (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

由此创建的跟踪 AWS CloudTrail ，用于记录组织 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

参见运营准备情况审查。

OT

参见运营技术。

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

查看个人身份信息。

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

参见可编程逻辑控制器。

PLM

参见产品生命周期管理。

policy

一个对象，可以在中定义权限（参见基于身份的策略）、指定访问条件（参见基于资源的策略）或定义组织中所有账户的最大权限 AWS Organizations （参见服务控制策略）。

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。有关更多信息，请参阅在微服务中实现数据持久性。

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`或的查询条件`false`，通常位于子`WHERE`句中。

谓词下推

一种数据库查询优化技术，可在传输前筛选查询中的数据。这减少了必须从关系数据库检索和处理的数据量，并提高了查询性能。

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。

通过设计保护隐私

一种在整个开发过程中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

一种安全控制措施，旨在防止部署不合规的资源。这些控件会在资源配置之前对其进行扫描。如果资源与控件不兼容，则不会对其进行配置。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)

在产品的整个生命周期中，从设计、开发和上市，到成长和成熟，再到衰落和移除，对产品进行数据和流程的管理。

生产环境

参见环境。

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。

提示链接

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，以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

一系列步骤，例如指令，用于访问 SQL 关系数据库系统中的数据。

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

参见 “负责任、负责、咨询、知情”（RACI）。

RAG

请参见检索增强生成。

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

参见 “负责任、负责、咨询、知情”（RACI）。

RCAC

请参阅行和列访问控制。

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。

重新架构师

见 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

见 7 R。

区域

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 AWS 区域 您的账户可以使用的账户。

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

见 7 R。

版本

在部署过程中，推动生产环境变更的行为。

搬迁

见 7 R。

更换平台

见 7 R。

回购

见 7 R。

故障恢复能力

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

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

见 7 R。

退休

见 7 R。

检索增强生成（RAG）

一种生成式人工智能技术，其中法学硕士在生成响应之前引用其训练数据源之外的权威数据源。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据的过程。

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

参见恢复点目标。

RTO

参见恢复时间目标。

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 Management Console 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

参见监督控制和数据采集。

SCP

参见服务控制政策。

secret

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 Secret s Manager 密钥中有什么？ 在 Secrets Manager 文档中。

安全性源于设计

一种在整个开发过程中考虑安全性的系统工程方法。

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。安全控制主要有四种类型：预防性、侦测、响应式和主动式。

安全加固

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

一种预定义和编程的操作，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 实例或轮换证书。

服务器端加密

在目的地对数据进行加密，由接收方 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。

服务级别指示器 (SLI)

对服务性能方面的衡量，例如其错误率、可用性或吞吐量。

服务级别目标 (SLO)

代表服务运行状况的目标指标，由服务级别指标衡量。

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

参见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

参见服务级别协议。

SLI

参见服务级别指标。

SLO

参见服务级别目标。

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，请参阅中的分阶段实现应用程序现代化的方法。 AWS Cloud

恶作剧

参见单点故障。

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交易数据或测量数据，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。

监控和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控有形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。

系统提示符

一种向法学硕士提供上下文、说明或指导方针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

tags

键值对，充当用于组织资源的元数据。 AWS 标签可帮助您管理、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

参见环境。

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

参见环境。

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

一个 SQL 函数，用于对一组以某种方式与当前记录相关的行进行计算。窗口函数对于处理任务很有用，例如计算移动平均线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。

蠕虫

参见一次写入，多读。

WQF

参见AWS 工作负载资格框架。

一次写入，多次读取 (WORM)

一种存储模型，它可以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，但他们无法对其进行更改。这种数据存储基础架构被认为是不可变的。

Z

零日漏洞利用

一种利用未修补漏洞的攻击，通常是恶意软件。

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。

零镜头提示

向法学硕士提供执行任务的说明，但没有示例（镜头）可以帮助指导任务。法学硕士必须使用其预先训练的知识来处理任务。零镜头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 few-shot 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。