AWS 白皮书

在上实现微服务 AWS

在上实现微服务 AWS: AWS 白皮书

Table of Contents

在上实现微服务 AWS

发布日期：2023 年 7 月 31 日 (文档历史记录)

微服务提供了一种简化的软件开发方法，可加快部署、鼓励创新、增强可维护性并提高可扩展性。这种方法依赖于通过定义明确的小型松散耦合服务 APIs，这些服务由自主团队管理。采用微服务可以带来诸多好处，例如增强的可扩展性、弹性、灵活性和更快的开发周期。

本白皮书探讨了三种流行的微服务模式：API 驱动、事件驱动和数据流。 我们概述了每种方法，概述了微服务的主要功能，解决了微服务开发中的挑战，并说明了 Amazon Web Services (AWS) 如何帮助应用程序团队克服这些障碍。

考虑到数据存储、异步通信和服务发现等主题的复杂性，我们鼓励您在做出架构决策时权衡应用程序的特定需求和用例以及提供的指导。

简介

微服务架构结合了来自各个领域的成功且久经考验的概念，例如：

- 敏捷软件开发
- 面向服务的架构
- API 优先的设计
- 连续Integration/Continuous Delivery (CI/CD)

通常，微服务会整合 T welve- Factor 应用程序中的设计模式。

虽然微服务具有许多好处，但评估用例的独特要求和相关成本至关重要。在某些情况下，整体架构或替代方法可能更合适。应 case-by-case根据规模、复杂性和具体用例等因素，在微服务还是单体之间做出决定。

我们首先探索一种高度可扩展、容错的微服务架构（用户界面、微服务实现、数据存储），并演示如何使用容器技术构建该架构。 AWS 然后，我们建议 AWS 服务来实现典型的无服务器微服务架构，从而降低运营复杂性。

无服务器的特点是以下原则：

- 无需配置或管理基础架构

- 按消耗单位自动缩放
- “按价值付费” 计费模式
- 内置可用性和容错能力
- 事件驱动架构 (EDA)

最后，我们将研究整个系统并讨论微服务架构的跨服务方面，例如分布式监控、日志、跟踪、审计、数据一致性和异步通信。

本文档重点介绍在中运行的工作负载 AWS 云，不包括混合场景和迁移策略。 有关迁移策略的信息，请参阅容器迁移方法白皮书。

您使用 Well-Architected 了吗？

当您在云端构建系统时，AWS Well-Architected Framework 可帮助您了解所做决策的利弊。利用此框架的六个支柱，您可以了解到设计和运行可靠、安全、高效、经济有效且可持续的系统的架构最佳实践。您可以使用AWS 管理控制台免费提供的 AWS Well-Architected Tool，回答与每个支柱相关的一组问题，即可根据这些最佳实践检查自己的工作负载。

在无服务器应用程序视角中，我们重点介绍在上架构无服务器应用程序的最佳实践。 AWS

有关云架构的更多专家指导和最佳实践（参考架构部署、图表和白皮书），请参阅 AWS 架构中心。

向微服务进行现代化改造

微服务本质上是构成应用程序的小型、独立的单元。从传统的整体结构过渡到微服务可以遵循多种策略。

这种过渡还会影响贵组织的运作方式：

- 它鼓励敏捷开发，即团队在快速周期内工作。
- 团队通常很小，有时被描述为两支披萨队，足够小，两个披萨可以养活整个团队。
- 从创建到部署和维护，团队对其服务承担全部责任。

开启了简单的微服务架构 AWS

典型的单片应用程序由不同的层组成：表示层、应用层和数据层。 另一方面，微服务架构根据特定领域而不是技术层将功能划分为有凝聚力的垂直市场。图 1 说明了上 AWS典型微服务应用程序的参考架构。

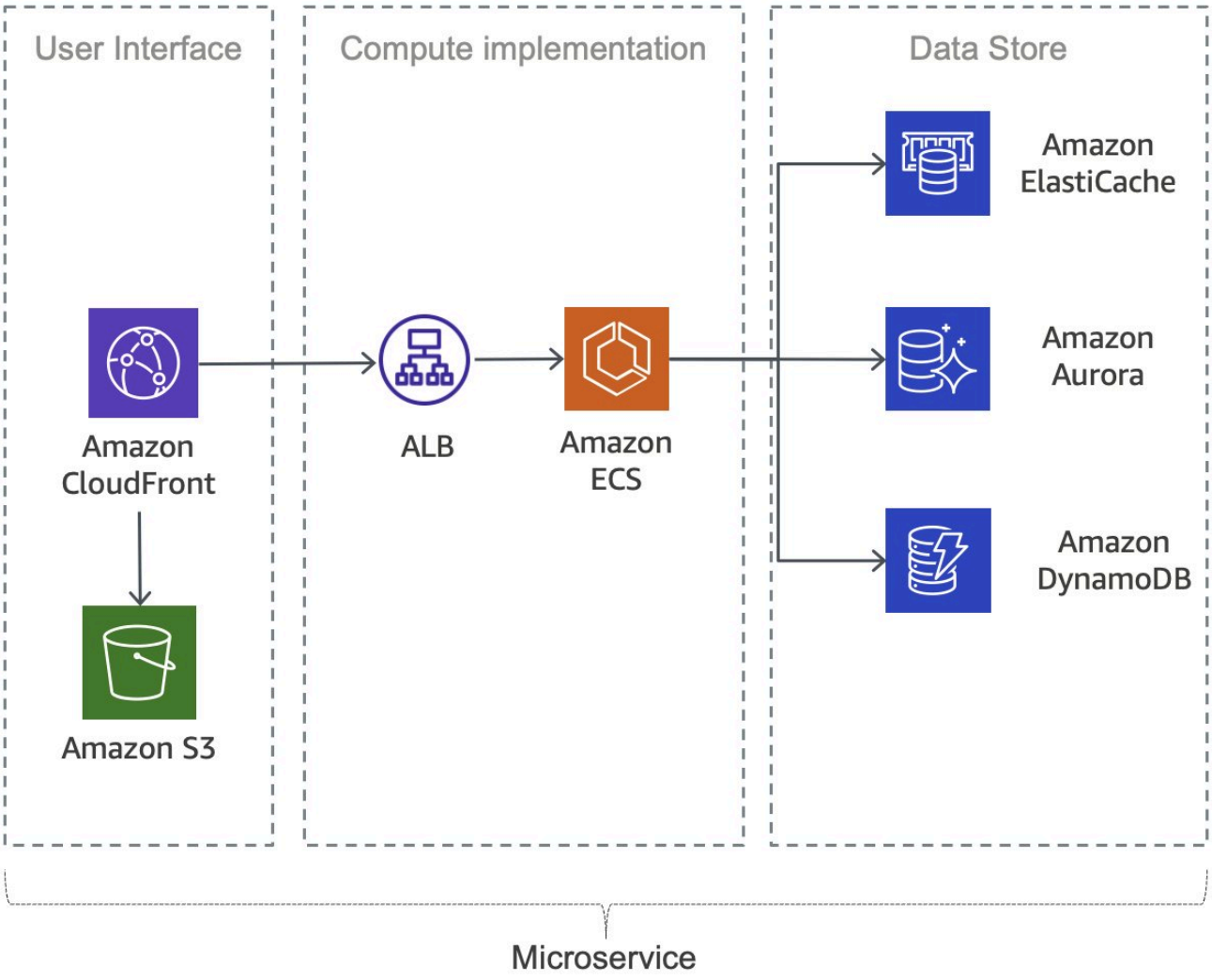

图 1：上的典型微服务应用程序 AWS

用户界面

现代 Web 应用程序通常使用 JavaScript 框架来开发与后端 APIs通信的单页应用程序。 APIs 它们通常使用表述性状态转移 (REST) 或或 RESTful APIs GraphQL APIs 构建。静态网页内容可以使用亚马逊简单存储服务 (A mazon S3) 和亚马逊提供 CloudFront。

微服务

APIs 被认为是微服务的前门，因为它们是应用程序逻辑的入口点。通常 RESTful 使用网络服务 API 或 GraphQL APIs 。它们 APIs 管理和处理客户端呼叫，处理流量管理、请求过滤、路由、缓存、身份验证和授权等功能。

微服务实现

AWS 为开发微服务提供了构建块，包括作为容器编排引擎选择的 Amazon ECS 和 Amazon EKS AWS Fargate 以及作为托管选项的 EC2。 AWS Lambda 是在上面构建微服务的另一种无服务器方式。AWS这些托管选项之间的选择取决于客户管理底层基础架构的要求。

AWS Lambda 允许您上传代码，自动缩放并以高可用性管理其执行。这消除了对基础架构管理的需求，因此您可以快速行动并专注于业务逻辑。Lambda 支持多种编程语言，可以由其他 AWS 服务触发，也可以直接从 Web 或移动应用程序调用。

由于便携性、生产力和效率，基于容器的应用程序越来越受欢迎。AWS 为构建、部署和管理容器提供了多种服务。

- App2Container，一种命令行工具，用于将 Java 和.NET Web 应用程序迁移到容器格式并对其进行现代化改造。 AWS A2C 分析并生成在裸机、虚拟机、Amazon Elastic Compute Cloud (EC2) 实例或云中运行的应用程序清单。

- 亚马逊弹性容器服务 (Amazon ECS) 和亚马逊弹性 Kubernetes Service (Amazon EKS) 管理您的容器基础设施，从而更轻松地启动和维护容器化应用程序。
  - Amazon EKS 是一项托管 Kubernetes 服务，用于在 AWS 云端和本地数据中心（亚马逊 EKS Anywhere）中运行 Kubernetes。这将云服务扩展到本地环境，以满足低延迟、本地数据处理、高数据传输成本或数据驻留要求（参见 “使用 Amazon EKS Anywhere 运行混合容器工作负载” 的白皮书）。您可以将来自 Kubernetes 社区的所有现有插件和工具与 EKS 一起使用。
  - Amazon Elastic Container Service (Amazon ECS) 是一项完全托管的容器编排服务，可简化容器化应用程序的部署、管理和扩展。客户之所以选择 ECS，是因为它既简单又能与 AWS 服务深度集成。

如需进一步阅读，请参阅博客 Amazon ECS vs Amazon EKS：理解 AWS 容器服务。

- AWS App Runner是一项完全托管的容器应用程序服务，允许您构建、部署和运行容器化 Web 应用程序和 API 服务，无需事先具备基础架构或容器经验。

- AWS Fargate是一款无服务器计算引擎，可与 Amazon ECS 和 Amazon EKS 配合使用，自动管理容器应用程序的计算资源。
- Amazon ECR 是一个完全托管的容器注册表，提供高性能托管，因此您可以可靠地在任何地方部署应用程序映像和工件。

持续集成和持续部署 (CI/CD)

持续集成和持续交付 (CI/CD) 是快速软件变更 DevOps计划的关键部分。 AWS CI/CD 为微服务提供了要实现的服务，但详细讨论超出了本文档的范围。有关更多信息，请参阅《实践持续集成和持续交付》AWS白皮书。

私有联网

AWS PrivateLink 是一种通过允许在您的虚拟私有云 (VPC) 和支持的服务之间建立私有连接来增强微服务安全的技术。 AWS 它有助于隔离和保护微服务流量，确保其永远不会通过公共互联网。这对于遵守PCI或HIPAA等法规特别有用。

数据存储

数据存储用于保存微服务所需的数据。会话数据的常用存储是内存缓存，例如 Memcached 或 Redis。AWS 在 Amazon 托管 ElastiCache服务中提供了这两种技术。

在应用程序服务器和数据库之间放置缓存是减少数据库读取负载的常用机制，这反过来又可以允许使用资源来支持更多的写入。缓存还可以改善延迟。

关系数据库在存储结构化数据和业务对象方面仍然非常流行。 AWS 通过亚马逊关系数据库服务（亚马逊 RDS）提供六种数据库引擎（微软 SQL Server、甲骨文、MySQL、MariaDB、PostgreSQL 和亚马逊 Aur ora）作为托管服务。

但是，关系数据库并不是为无限扩展而设计的，这会使应用技术来支持大量查询变得困难和耗时。

NoSQL 数据库的设计有利于可扩展性、性能和可用性，而不是关系数据库的一致性。NoSQL 数据库的一个重要元素是，它们通常不强制执行严格的架构。数据分布在可以水平扩展的分区上，并使用分区键进行检索。

由于单个微服务旨在很好地完成一件事，因此它们通常具有非常适合 NoSQL 持久性的简化数据模型。重要的是要明白，NoSQL 数据库的访问模式与关系数据库不同。例如，无法联接表。如果有必要，则

必须在应用程序中实现逻辑。您可以使用 Amazon Dy namoDB 创建数据库表，该表可以存储和检索任意数量的数据并处理任何级别的请求流量。DynamoDB 提供个位数的毫秒性能，但是，某些用例需要以微秒为单位的响应时间。 DynamoDB 加速器 (DAX) 提供用于访问数据的缓存功能。

DynamoDB 还提供自动扩展功能，可根据实际流量动态调整吞吐容量。但是，在某些情况下，由于应用程序中出现了持续时间较短的大型活动峰值，因此容量规划很困难或不可能。对于此类情况，DynamoDB 提供了按需选项，其定价非常简单。 pay-per-requestDynamoDB 按需能够即时处理每秒数千个请求，无需进行容量规划。

有关更多信息，请参阅分布式数据管理和如何选择数据库。

简化操作

为了进一步简化运行、维护和监控微服务所需的运营工作，我们可以使用完全无服务器的架构。

部署基于 Lambda 的应用程序

您可以通过上传`zip`文件存档来部署 Lambda 代码，也可以使用有效的 Amazon ECR 映像 URI 通过控制台用户界面创建和上传容器映像。但是，当 Lambda 函数变得复杂（这意味着它具有层、依赖关系和权限）时，通过用户界面上传可能会因为代码更改而变得笨拙。

使用AWS CloudFormation 和 AWS Serverless Application Model (AWS SAM) AWS Cloud Development Kit (AWS CDK)、或 Terraform 可以简化定义无服务器应用程序的过程。 AWS SAM 原生支持 CloudFormation，它提供了一种用于指定无服务器资源的简化语法。AWS Lambda 层可帮助管理多个 Lambda 函数之间的共享库，最大限度地减少函数占用空间，集中租户感知库并改善开发者体验。Lambda f SnapStart or Java 增强了延迟敏感型应用程序的启动性能。

要进行部署，请在 CloudFormation 模板中指定资源和权限策略，打包部署对象，然后部署模板。SAM Local 是一种 AWS CLI 工具，允许在上传到 Lambda 之前对无服务器应用程序进行本地开发、测试和分析。

与 AWS Cloud9 IDE、、等工具集成 AWS CodeBuild AWS CodeDeploy， AWS CodePipeline 简化了基于 SAM 的应用程序的创作、测试、调试和部署。

下图显示了使用 CloudFormation 和 C AWS I/CD 工具部署 AWS Serverless Application Model 资源。

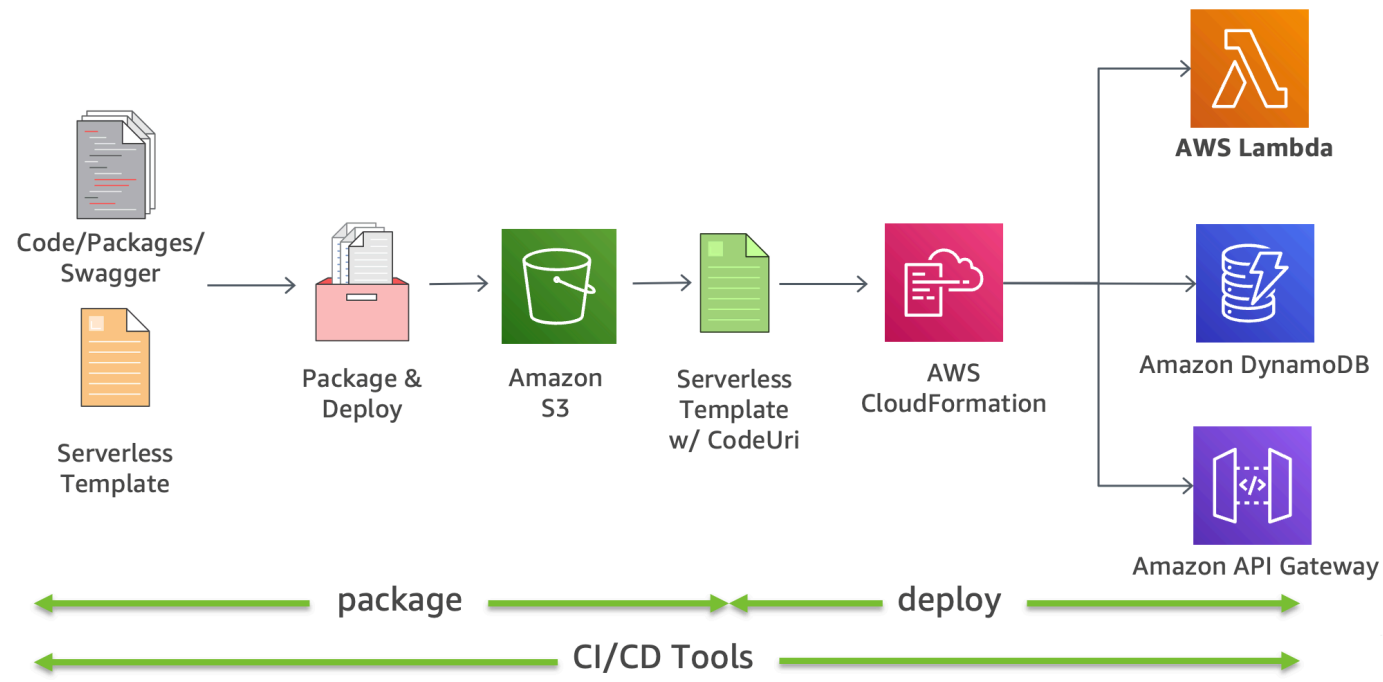

图 2: AWS Serverless Application Model (AWS SAM)

抽象多租户的复杂性

在像 SaaS 平台这样的多租户环境中，简化与多租户相关的复杂性，让开发人员腾出时间专注于功能和功能开发至关重要。这可以通过诸如L AWS Lambda ay ers之类的工具来实现，这些工具提供了用于解决跨领域问题的共享库。 这种方法背后的基本原理是，如果使用得当，共享库和工具可以有效地管理租户上下文。

但是，由于它们可能带来的复杂性和风险，它们不应扩展到封装业务逻辑。共享库的一个基本问题是更新的复杂性增加，与标准代码复制相比，更新更难管理。因此，在寻求最有效的抽象时，必须在使用共享库和重复之间取得平衡。

API 管理

管理 APIs 可能很耗时，尤其是在考虑多个版本、开发周期的各个阶段、授权以及其他功能（例如限制和缓存）时。除了 API Gatew ay 之外，一些客户还使用 ALB（应用程序负载均衡器）或 NLB（网络负载均衡器）进行 API 管理。Amazon API Gateway 有助于降低创建和维护的操作复杂性 RESTful APIs。它允许您以 APIs 编程方式创建，充当访问后端服务、授权和访问控制、速率限制、缓存、监控和流量管理中的数据、业务逻辑或功能的“前门”，并且 APIs 无需管理服务器即可运行。

图 3 说明了 API Gateway 如何处理 API 调用以及如何与其他组件交互。来自移动设备、网站或其他后端服务的请求会被路由到最近的接入 CloudFront 点 (PoP)，以减少延迟并提供最佳的用户体验。

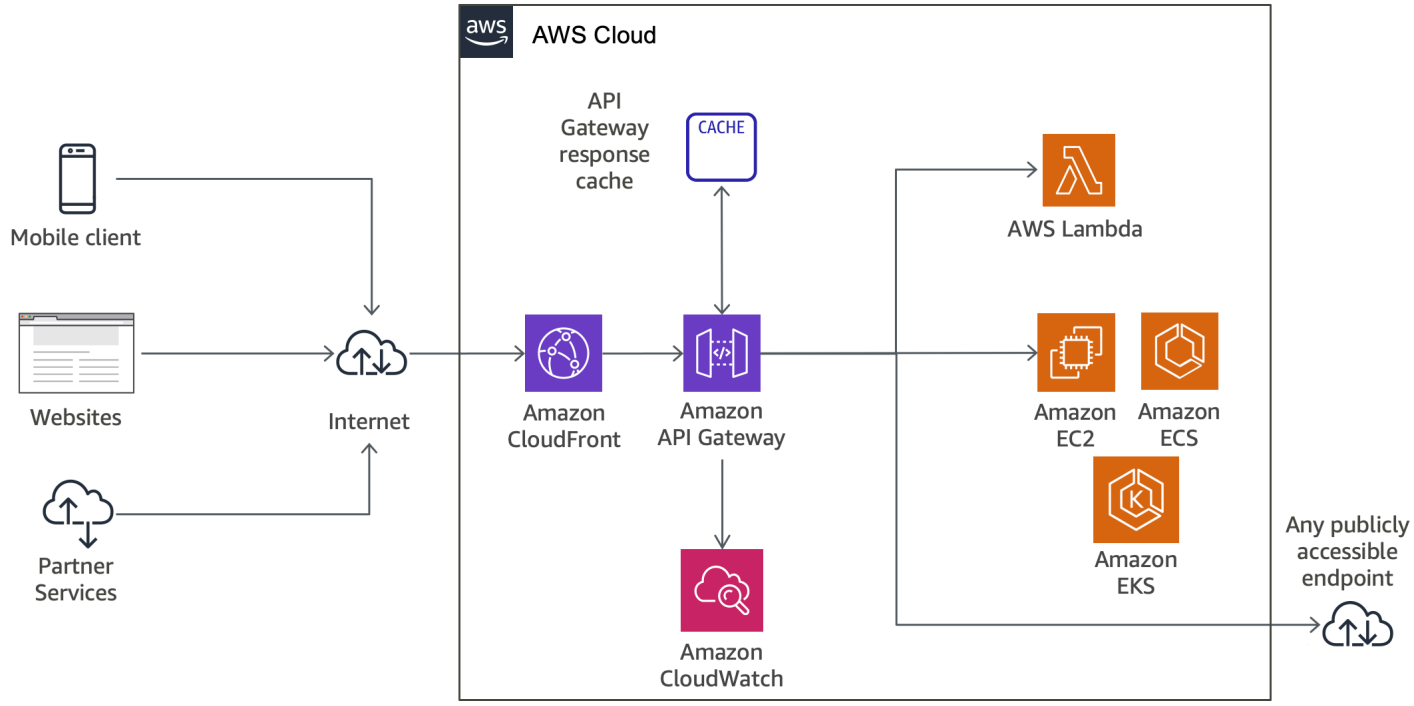

图 3：API Gateway 调用流程

无服务器技术上的微服务

将微服务与无服务器技术结合使用可以大大降低运营复杂性。 AWS Lambda 并且 AWS Fargate，与 API Gateway 集成，允许创建完全无服务器的应用程序。自 2023 年 4 月 7 日起，Lambda 函数可以逐步将响应负载流回客户端，从而提高 Web 和移动应用程序的性能。在此之前，使用传统请求-响应调用模型的基于 Lambda 的应用程序必须在将响应返回给客户端之前生成并缓冲响应，这可能会延迟第一个字节的时间。通过响应流，函数可以在客户端准备就绪时将部分响应发送回客户端，从而显著缩短第一个字节的时间，而网络和移动应用程序对此特别敏感。

图 4 演示了使用 AWS Lambda 托管服务的无服务器微服务架构。这种无服务器架构减少了设计规模和高可用性的需求，并减少了运行和监控底层基础设施所需的工作量。

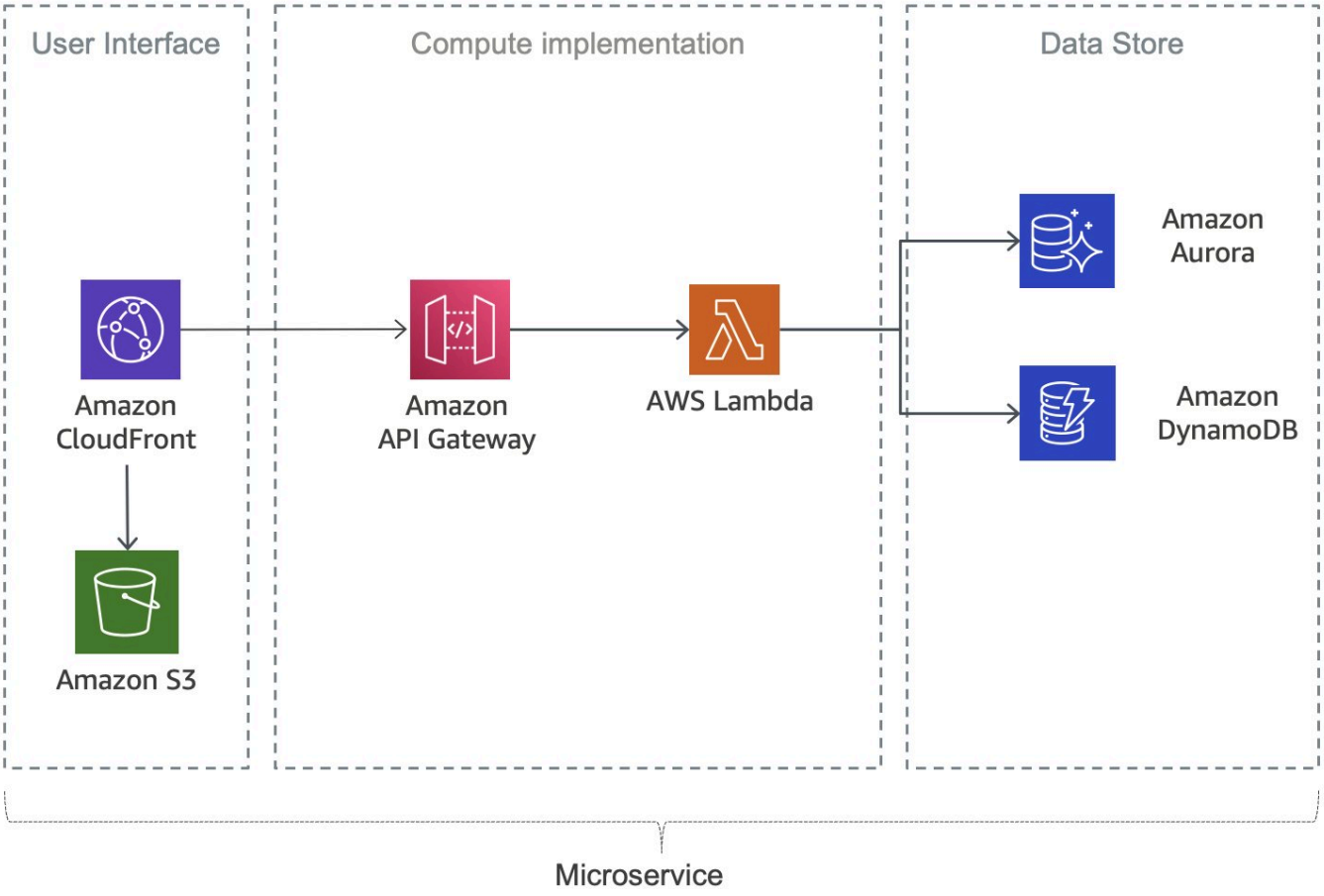

图 4：使用无服务器微服务 AWS Lambda

图 5 显示了使用带容器的类似无服务器实现 AWS Fargate，消除了对底层基础设施的担忧。它还具有 Amazon Aurora Serverless，这是一种按需自动缩放的数据库，可根据应用程序的要求自动调整容量。

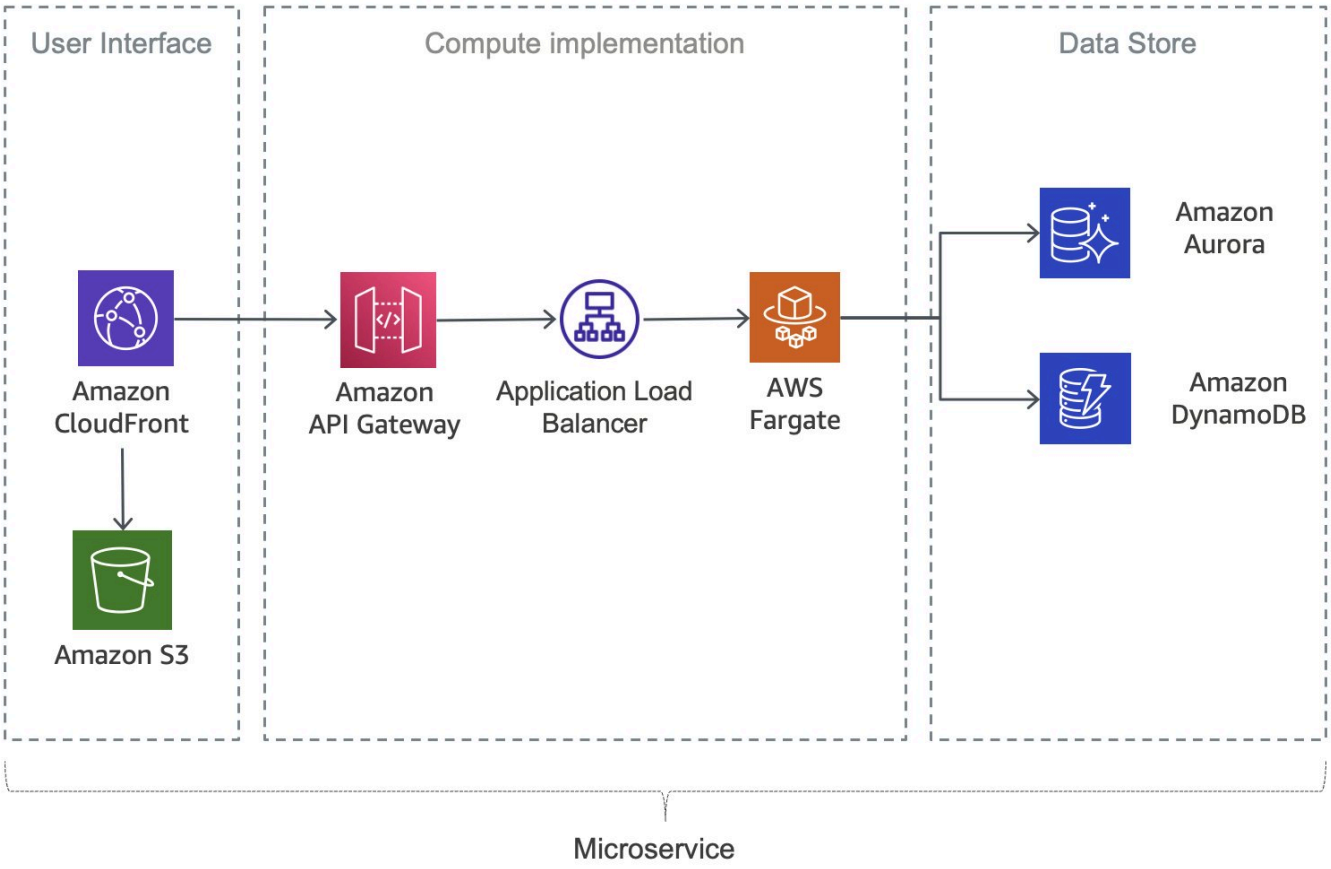

图 5：使用无服务器微服务 AWS Fargate

弹性且高效的系统

灾难恢复 (DR)

微服务应用程序通常遵循十二因子应用程序模式，即进程是无状态的，持久数据存储在数据库等有状态的支持服务中。这简化了灾难恢复 (DR)，因为如果服务出现故障，可以轻松启动新实例来恢复功能。

微服务的灾难恢复策略应侧重于维护应用程序状态的下游服务，例如文件系统、数据库或队列。Organizations 应规划恢复时间目标 (RTO) 和恢复点目标 (RPO)。RTO 是服务中断和恢复之间的最大可接受延迟，而 RPO 是自上次数据恢复点以来的最长时间。

有关灾难恢复策略的更多信息，请参阅《工作负载灾难恢复 AWS：云端恢复》白皮书。

高可用性 (HA)

我们将研究微服务架构各个组件的高可用性 (HA)。

Amazon EKS 通过跨多个可用区运行 Kubernetes 控制和数据平面实例来提供高可用性。它会自动检测和替换运行状况不佳的控制平面实例，并提供自动版本升级和修补。

Amazon ECR 使用亚马逊简单存储服务 (Amazon S3) Simple Service 进行存储，使您的容器镜像具有高度可用性和可访问性。它可与 Amazon EKS、Amazon ECS 配合使用 AWS Lambda，简化了从开发到生产的工作流程。

Amazon ECS 是一项区域性服务，它以高度可用的方式简化在一个区域内的多个可用区中运行容器，它提供了多种调度策略，可以根据资源需求和可用性要求放置容器。

AWS Lambda 在多个可用区中运行，确保单个区域服务中断期间的可用性。如果将您的函数连接到 VPC，请在多个可用区中指定子网以实现高可用性。

分布式系统组件

在微服务架构中，服务发现是指动态定位和识别分布式系统中各个微服务的网络位置（IP 地址和端口）的过程。

在选择方法时 AWS，请考虑以下因素：

- 代码修改：不修改代码就能获得好处吗？
- 跨VPC或跨账户流量：如果需要，您的系统是否需要高效管理不同 VPCs 或跨账户的通信？ AWS 账户
- 部署策略：您的系统是否使用或计划使用高级部署策略，例如蓝绿色或金丝雀部署？
- 性能注意事项：如果您的架构经常与外部服务通信，会对整体性能产生什么影响？

AWS 提供了几种在微服务架构中实现服务发现的方法：

- Amazon ECS 服务发现：Amazon ECS 使用其基于 DNS 的方法或通过与集成 AWS Cloud Map（请参阅 ECS 服务发现）来支持服务发现。ECS Service Connect 进一步改进了连接管理，这对于具有多个交互服务的大型应用程序尤其有益。
- Amazon Route 53：Route 53 与 ECS 和其他 AWS 服务（例如 EKS）集成，以促进服务发现。在 ECS 环境中，Route 53 可以使用 ECS 服务发现功能，该功能利用自动命名 API 自动注册和取消注册服务。
- AWS Cloud Map：此选项提供基于 API 的动态服务发现，可在您的服务中传播更改。

为了满足更高级的通信需求，Amazon VPC Lattice 是一项应用程序联网服务，可持续连接、监控和保护服务之间的通信，有助于提高工作效率，使您的开发人员可以专注于构建对您的业务至关重要的功能。您可以为网络流量管理、访问和监控定义策略，以简化且一致的方式跨实例、容器和无服务器应用程序连接计算服务。

如果您已经在使用第三方软件（例如 HashiCorp Consul 或 Netflix Eureka）进行服务发现，则可能希望在迁移到时继续使用这些软件 AWS，从而实现更顺畅的过渡。

这些选项之间的选择应符合您的特定需求。对于更简单的要求，基于 DNS 的解决方案（例如 Amazon ECS 或） AWS Cloud Map 可能就足够了。对于更复杂或更大的系统，像 Amazon VPC Lattice 这样的服务网格可能更合适。

总而言之，设计微服务架构 AWS 就是要选择合适的工具来满足您的特定需求。通过牢记所讨论的注意事项，您可以确保做出明智的决策，以优化系统的服务发现和服务间通信。

分布式数据管理

在传统应用程序中，所有组件通常共享一个数据库。相比之下，基于微服务的应用程序的每个组件都维护自己的数据，从而促进了独立性和去中心化。这种被称为分布式数据管理的方法带来了新的挑战。

其中一个挑战来自分布式系统中一致性和性能之间的权衡。接受数据更新中的轻微延迟（最终一致性）通常比坚持即时更新（即时一致性）更为实用。

有时，业务运营需要多个微服务才能协同工作。如果一个部件出现故障，则可能需要撤消一些已完成的任务。S aga模式通过协调一系列补偿措施来帮助管理这种情况。

为了帮助微服务保持同步，可以使用集中式数据存储。这家商店使用 AWS Lambda AWS Step Functions、和 Amazon 等工具进行管理 EventBridge，可以帮助清理和删除重复数据。

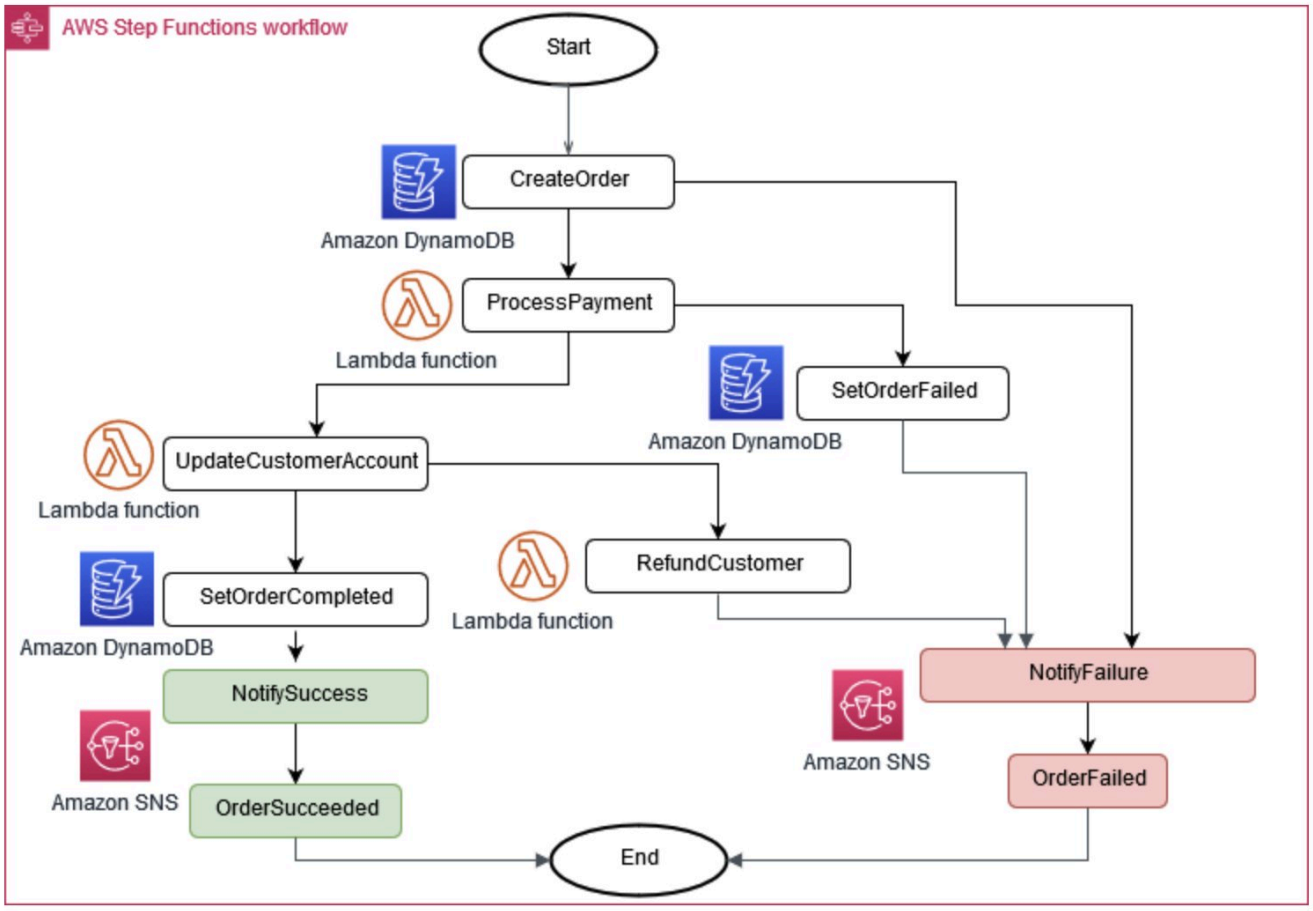

图 6：Saga 执行协调器

管理微服务变更的一种常见方法是事件来源。应用程序中的每一次更改都记录为事件，从而创建系统状态的时间表。这种方法不仅有助于调试和审计，还允许应用程序的不同部分对相同的事件做出反应。

事件来源通常 hand-in-hand使用命令查询责任分离 (CQRS) 模式，该模式将数据修改和数据查询分成不同的模块，以提高性能和安全性。

在上 AWS，您可以使用服务组合来实现这些模式。如图 7 所示，Amazon Kinesis Data Streams 可以用作您的中央事件存储，而 Amazon S3 可为所有事件记录提供持久存储。 AWS Lambda、Amazon DynamoDB 和 Amazon API Gateway 共同处理和处理这些事件。

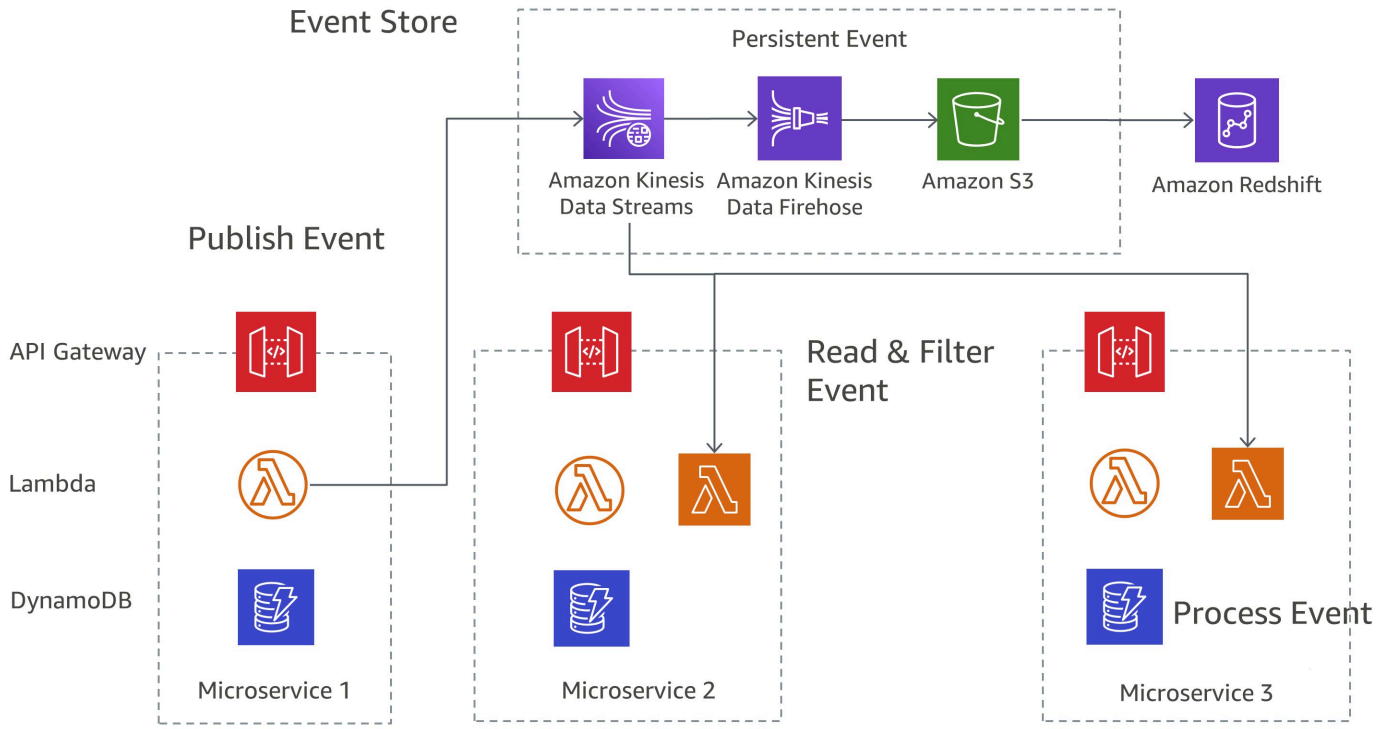

图 7：活动来源模式开启 AWS

请记住，在分布式系统中，由于重试，事件可能会被多次传送，因此设计应用程序来处理这个问题非常重要。

配置管理

在微服务架构中，每项服务都与数据库、队列和其他服务等各种资源进行交互。以一致的方式配置每项服务的连接和操作环境至关重要。理想情况下，应用程序无需重启即可适应新的配置。这种方法是 Twelve-Factor App 原则的一部分，该原则建议将配置存储在环境变量中。

另一种方法是使用 AWS App Config。这是 S AWS ystems Manager 的一项功能，它使客户可以轻松快速、安全地配置、验证和部署功能标志和应用程序配置。您的功能标志和配置数据可以在部署前阶段通过语法或语义进行验证，并且可以在触发您配置的警报时进行监控和自动回滚。 AppConfig 可以使用 AWS AppConfig 代理与 Amazon ECS 和 Amazon EKS 集成。该代理充当 sidecar 容器，与您的 Amazon ECS 和 Amazon EKS 容器应用程序一起运行。如果您在 Lambda 函数中使用 AWS AppConfig 功能标志或其他动态配置数据，那么我们建议您将 Lambda 扩展作为一个层添加到您的 AWS AppConfig Lambda 函数中。

GitOps是一种创新的配置管理方法，它使用 Git 作为所有配置更改的真实来源。这意味着对配置文件所做的任何更改都将通过 Git 自动跟踪、版本控制和审计。

密钥管理

安全性至关重要，因此不应以纯文本形式传递凭证。 AWS 为此提供安全的服务，例如 S AWS ystems Manager 参数存储和 AWS Secrets Manager。这些工具可以将机密作为卷发送到 Amazon EKS 中的容器，或者作为环境变量发送到 Amazon ECS。在中 AWS Lambda，环境变量会自动提供给您的代码。 对于 Kubernetes 工作流程，外部机密操作员直接从服务中获取机密，例如 AWS Secrets Manager创建相应的 Kubernetes 密钥。这可以实现与 Kubernetes 原生配置的无缝集成。

成本优化和可持续性

微服务架构可以增强成本优化和可持续性。通过将应用程序分成较小的部分，您可以只扩展需要更多资源的服务，从而减少成本和浪费。这在处理可变流量时特别有用。微服务是独立开发的。因此，开发人员可以进行较小的更新，减少端到端测试所花费的资源。在更新时，他们只需要测试部分功能，而不是整体功能。

您的架构中的无状态组件（将状态存储在外部数据存储而不是本地数据存储中的服务）可以使用 Amazon EC2 Spot 实例，这些实例在 AWS 云中提供未使用的 EC2容量。这些实例比按需实例更具成本效益，非常适合可以处理中断的工作负载。这可以在保持高可用性的同时进一步削减成本。

借助隔离服务，您可以为每个自动缩放组使用成本优化的计算选项。例如， AWS Graviton 为适合基于 ARM 的实例的工作负载提供了经济实惠的高性能计算选项。

优化成本和资源使用还有助于最大限度地减少对环境的影响，这与 Well-Architected Framework 的可持续发展支柱保持一致。您可以使用 AWS 客户碳足迹工具监控您在减少碳排放方面的进展。该工具可让您深入了解您的 AWS 使用对环境的影响。

沟通机制

在微服务模式中，应用程序的各个组件必须通过网络进行通信。实现此目的的常用方法包括基于 REST、基于 GraphQL、基于 GRPC 和异步消息传递。

基于休息的沟通

该 HTTP/S 协议广泛用于微服务之间的同步通信，通常通过 RESTful APIs它运行。API Gateway 提供了一种简化的方法来构建 API，该API充当后端服务的集中接入点，处理流量管理、授权、监控和版本控制等任务。

基于 GraphQL 的通信

同样，GraphQL 是一种广泛使用的同步通信方法，它使用与 REST 相同的协议，但限制了对单个端点的暴露。借助 AWS AppSync，您可以创建和发布直接与 AWS 服务和数据存储交互的 GraphQL 应用程序，或者为业务逻辑整合 Lambda 函数。

基于 GRPC 的通信

gRPC 是一种同步、轻量、高性能、开源 RPC 通信协议。gRPC 通过使用 HTTP/2 并启用更多功能（例如压缩和流优先级）对其底层协议进行了改进。它使用二进制编码的 Protobuf 接口定义语言 (IDL)，因此利用了 HTTP/2 二进制框架。

异步消息和事件传递

异步消息允许服务通过队列发送和接收消息进行通信。这使服务能够保持松散耦合并促进服务发现。

消息可以定义为以下三种类型：

- 消息队列：消息队列充当缓冲区，用于分离消息的发送者（生产者）和接收者（使用者）。生产者将消息排入队列，而消费者则将消息出队并进行处理。这种模式对于异步通信、负载均衡和处理突发流量很有用。
- 发布-订阅：在发布-订阅模式中，一条消息发布到一个主题，多个感兴趣的订阅者会收到该消息。这种模式允许异步向多个使用者广播事件或消息。
- 事件驱动的消息传递：事件驱动的消息传递涉及捕获系统中发生的事件并做出反应。事件发布到消息代理，感兴趣的服务会订阅特定的事件类型。这种模式支持松散耦合，并允许服务在没有直接依赖关系的情况下对事件做出反应。

为了实现每种消息类型， AWS 提供了各种托管服务，例如亚马逊 SQS、亚马逊 SNS、亚马逊 MQ 和 EventBridge亚马逊 MSK。这些服务具有针对特定需求量身定制的独特功能：

- 亚马逊简单队列服务 (Amazon SQS) 和亚马逊简单通知服务 (Amazon SNS) Simple Notification：如图 8 所示，这两种服务相辅相成，亚马逊 SQS 提供了存储消息的空间，Amazon SNS 允许向多个订阅者传送消息。当需要将同一封邮件传送到多个目的地时，它们会生效。

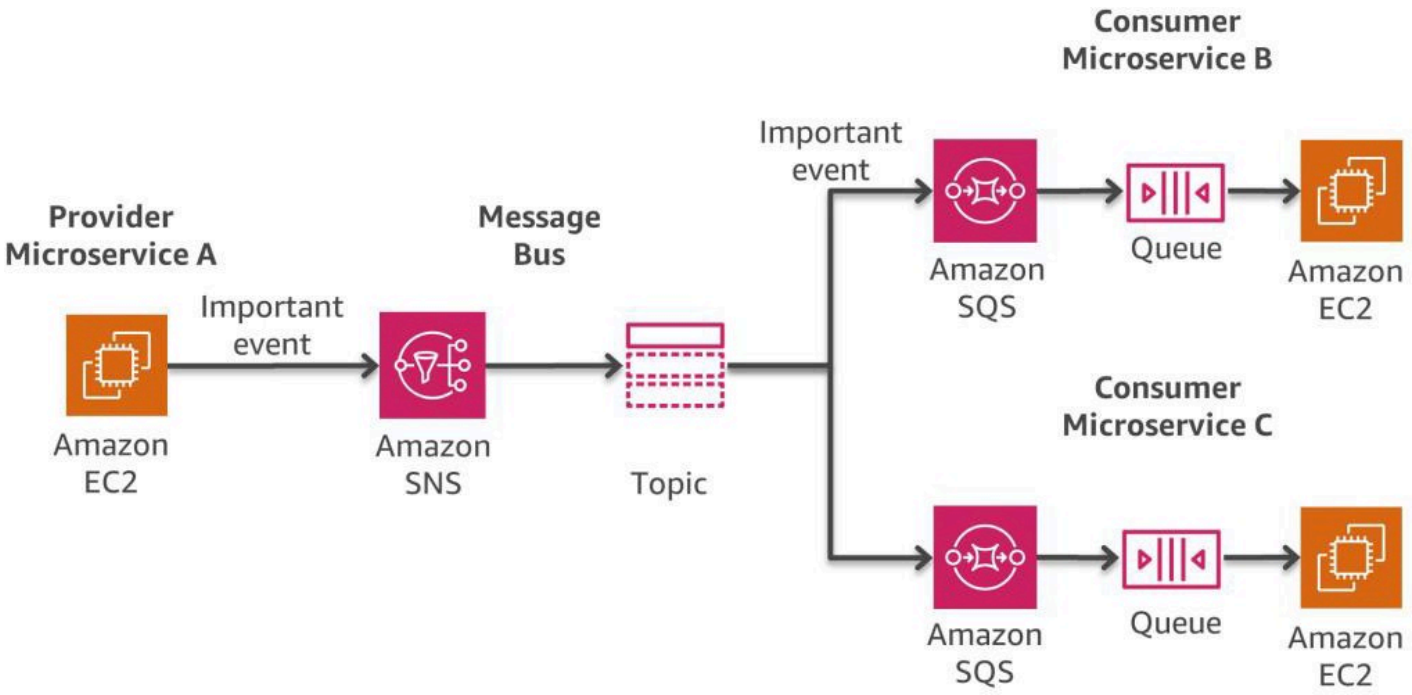

图 8：消息总线模式开启 AWS

- Amazon EventBridge：一种无服务器服务，它使用事件将应用程序组件连接在一起，使您可以更轻松地构建可扩展的事件驱动应用程序。使用它可以将来自本土应用程序、 AWS 服务和第三方软件等来源的事件路由到组织中的消费者应用程序。 EventBridge 提供了一种简单而一致的方法来提取、筛选、转换和交付事件，因此您可以快速构建新应用程序。 EventBridge 事件总线非常适合在事件驱动的服务之间 many-to-many路由事件。
- Amazon MQ：如果您的现有消息系统使用 JMS、AMQP 或类似协议等标准协议，那么这是一个不错的选择。此托管服务可在不中断操作的情况下替换您的系统。
- Amazon MSK（托管 Kafka）：一种用于存储和读取消息的消息系统，适用于必须多次处理消息的情况。它还支持实时消息流。
- Amazon Kinesis：实时处理和分析流数据。这允许开发实时应用程序，并提供与 AWS 生态系统的无缝集成。

请记住，最适合您的服务取决于您的特定需求，因此了解每项服务提供的服务以及它们如何与您的要求保持一致非常重要。

编排和状态管理

微服务编排是指一种集中式方法，在这种方法中，一个名为协调器的中央组件负责管理和协调微服务之间的交互。跨多个微服务协调工作流程可能具有挑战性。不鼓励将编排代码直接嵌入到服务中，因为它会引入更紧密的耦合并阻碍替换单个服务。

Step Functions 提供了一个工作流引擎，用于管理服务编排的复杂性，例如错误处理和序列化。这使您无需添加协调代码即可快速扩展和更改应用程序。Step Functions 是 AWS 无服务器平台的一部分，支持 Lambda 函数、亚马逊 EC2、亚马逊 EKS、亚马逊 ECS SageMaker AWS Glue、人工智能等。

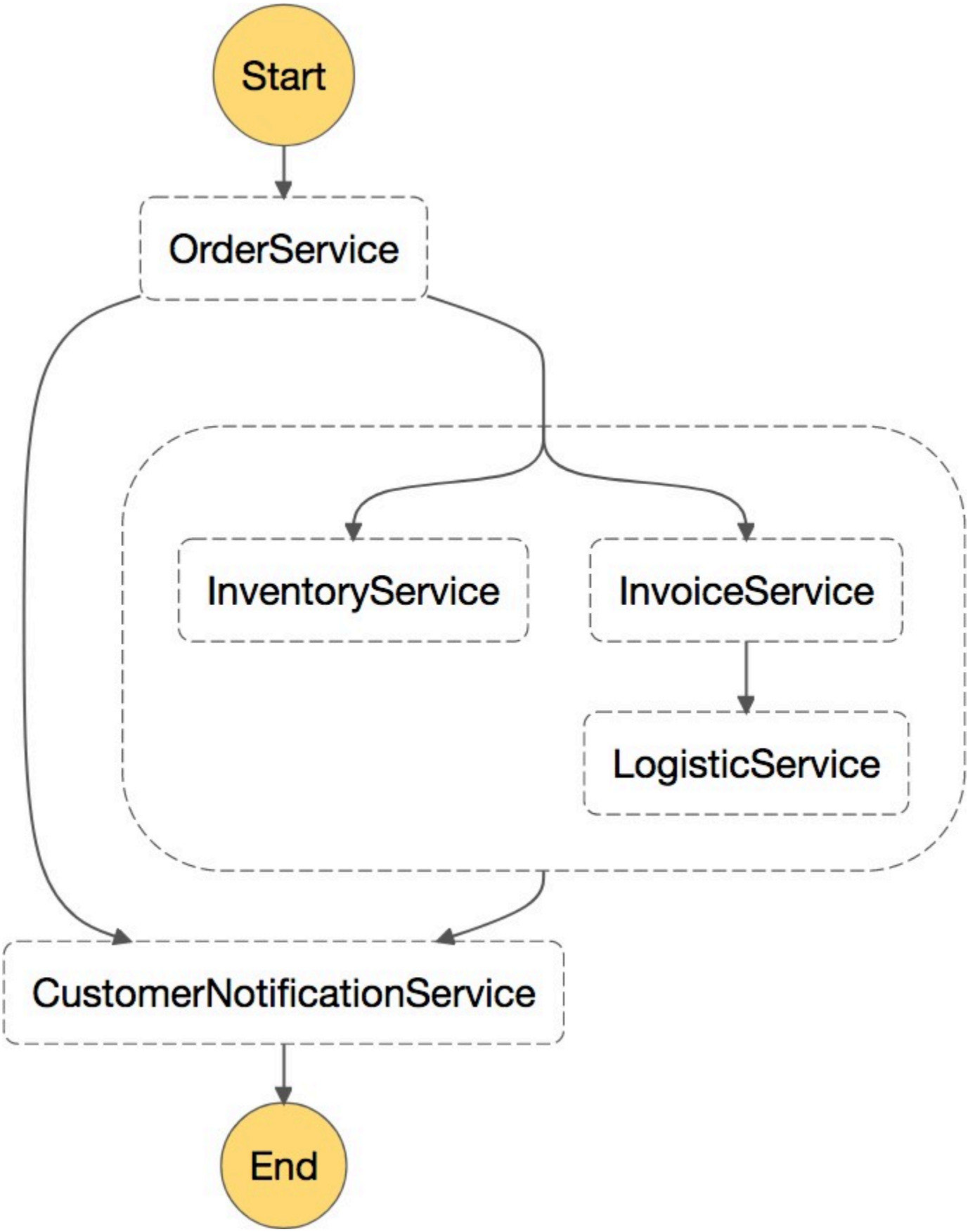
Start
OrderService
InventoryService
InvoiceService
LogisticService
CustomerNotificationService
End

图 9：调用并行和顺序步骤的微服务工作流程示例 AWS Step Functions

适用于 Apache Airflow 的亚马逊托管工作流程（亚马逊 MWAA）是 Step Functions 的替代方案。 如果您优先考虑开源和便携性，则应使用 Amazon MWAA。Airflow拥有一个庞大而活跃的开源社区，该社区定期提供新功能和集成。

可观测性

由于微服务架构本质上是由许多分布式组件组成的，因此所有这些组件的可观察性变得至关重要。Amazon CloudWatch 通过收集和跟踪指标、监控日志文件以及对 AWS 环境变化做出反应来实现这一点。它可以监控您的应用程序和服务生成的 AWS 资源和自定义指标。

主题

监控

CloudWatch 提供对资源利用率、应用程序性能和运行状况的全系统可见性。在微服务架构中，通过自定义指标进行监控 CloudWatch 是有益的，因为开发人员可以选择要收集的指标。动态扩展也可以基于这些自定义指标。

CloudWatch Container Insights 扩展了此功能，自动收集诸如 CPU、内存、磁盘和网络等许多资源的指标。它有助于诊断与容器相关的问题，简化解决方案。

对于 Amazon EKS 来说，通常首选的是 Prometheus，这是一个提供全面监控和警报功能的开源平台。它通常与 Grafana 配合使用，以实现直观的指标可视化。Amazon Prometheus 托管服务 (AMP) 提供与 Prometheus 完全兼容的监控服务，让您可以毫不费力地监督容器化应用程序。此外，Amazon Managed Grafana (AMG) 简化了指标的分析和可视化，无需管理底层基础设施。

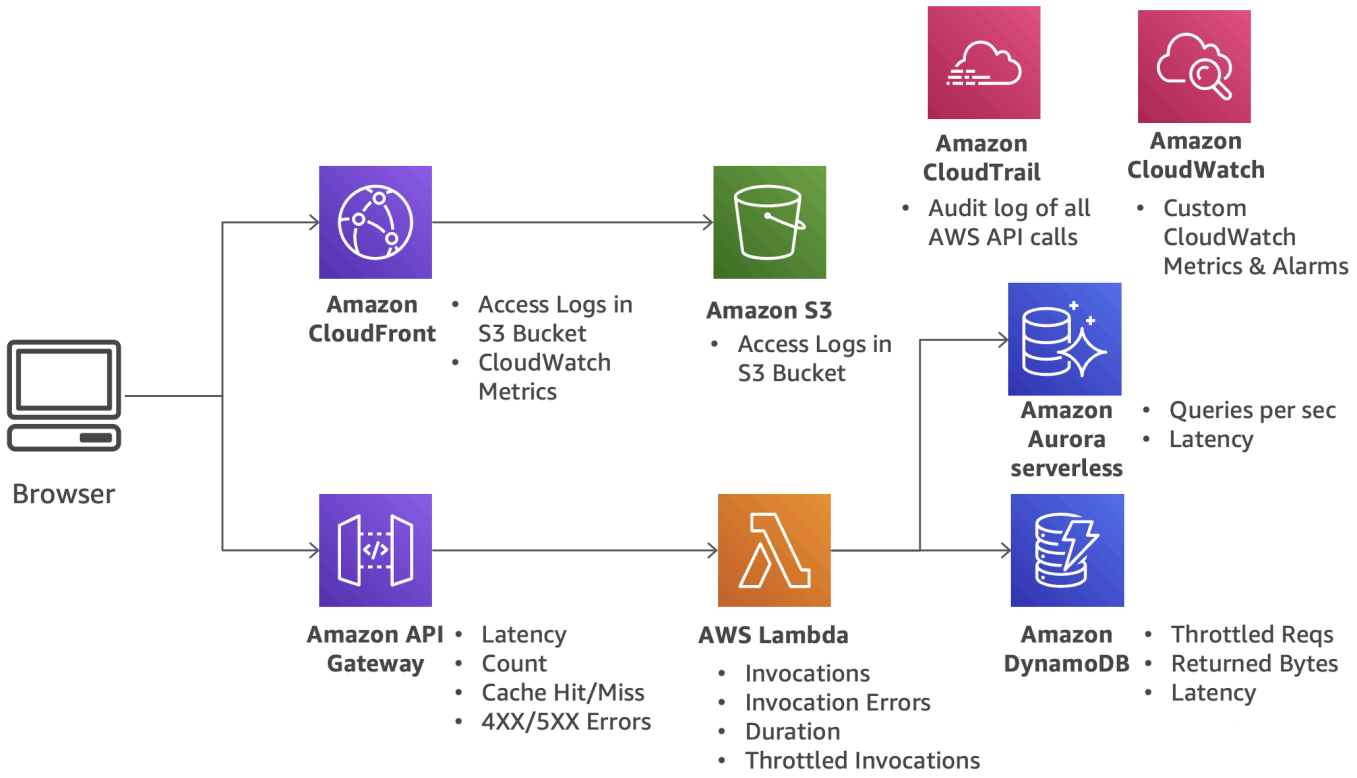

图 10：带有监控组件的无服务器架构

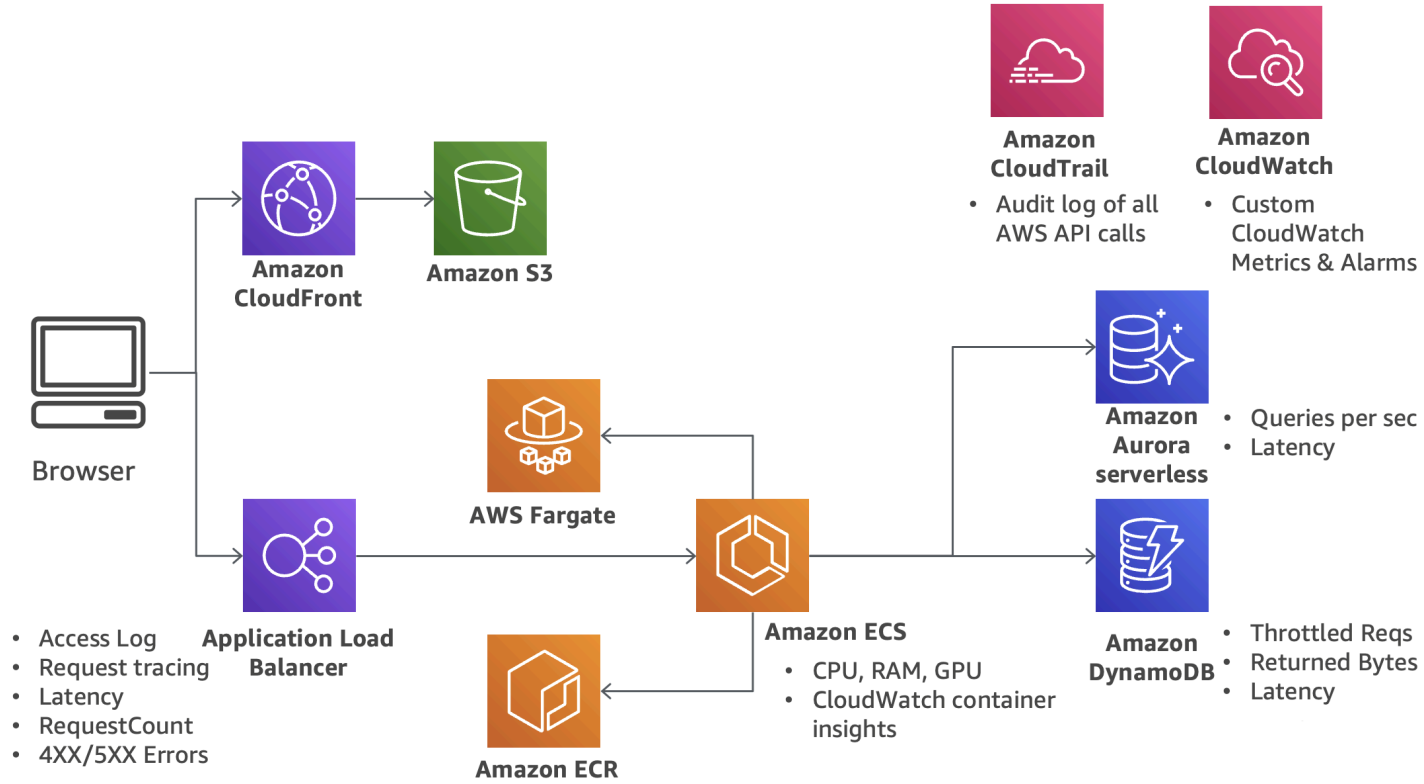

图 11：带有监控组件的基于容器的架构

集中日志

记录是查明和解决问题的关键。借助微服务，您可以更频繁地发布并尝试新功能。 AWS 提供 Amazon S3、CloudWatch 日志和亚马逊 OpenSearch 服务等服务来集中管理日志文件。Amazon EC2 使用守护程序向发送日志 CloudWatch，而 Lambda 和 Amazon ECS 则在本地将其日志输出发送到那里。对于 Amazon EKS，可以使用 Fluent Bit 或 Fluentd 将日志转发到，以便使用 OpenSearch 和 Kibana CloudWatch 进行报告。但是，由于占地面积较小且具有性能优势，因此建议使用 Fluent Bit 而不是 Fluentd。

图 12 说明了如何将来自各种 AWS 服务的日志定向到 Amazon S3 和 CloudWatch。可以使用亚马逊 OpenSearch 服务（包括用于数据可视化的 Kibana）对这些集中式日志进行进一步分析。此外，还可以使用 Amazon Athena 对存储在 Amazon S3 中的日志进行临时查询。

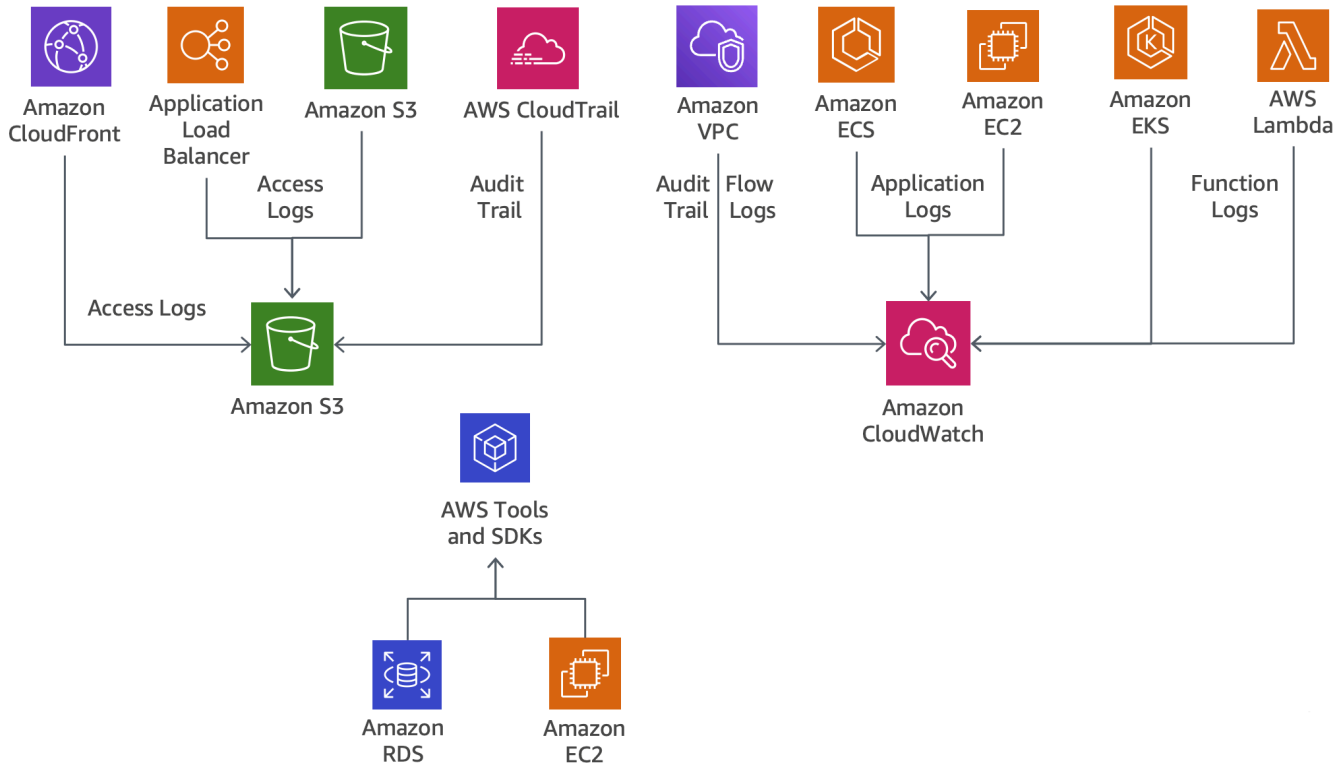

图 12： AWS 服务的日志功能

分布式跟踪

微服务通常协同工作来处理请求。 AWS X-Ray 使用关联 ID 来跟踪跨这些服务的请求。X-Ray 可与亚马逊 EC2、亚马逊 ECS、Lambda 和 Elastic Beanstalk 配合使用。

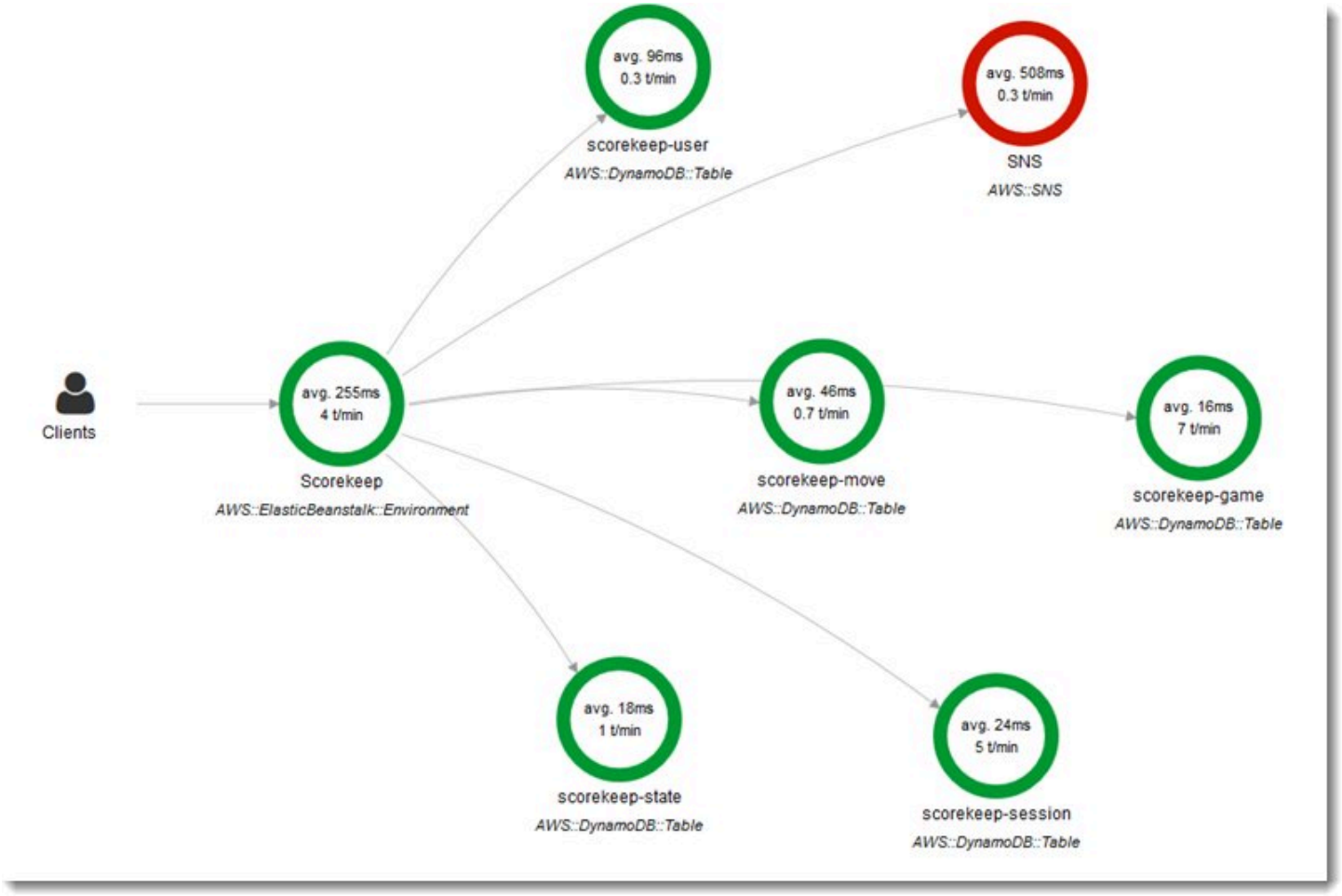

图 13： AWS X-Ray 服务地图

AWS Distro for OpenTelemetry 是该 OpenTelemetry 项目的一部分，它提供开源 APIs 和代理来收集分布式跟踪和指标，从而改善您的应用程序监控。 它向多个合作伙伴监控解决方案发送指标 AWS 和跟踪。通过从您的 AWS 资源中收集元数据，它可以使应用程序性能与底层基础架构数据保持一致，从而加快问题的解决。另外，它与各种 AWS 服务兼容，可以在本地使用。

开启日志分析 AWS

Amazon Lo CloudWatch gs Insights 允许进行实时日志探索、分析和可视化。为了进一步分析日志文件，包括 Kibana 在内的亚马逊 OpenSearch 服务是一款功能强大的工具。 CloudWatch 日志可以将日

志条目实时流式传输到 OpenSearch 服务。Kibana 与之无缝集成 OpenSearch，可视化这些数据，并提供直观的搜索界面。

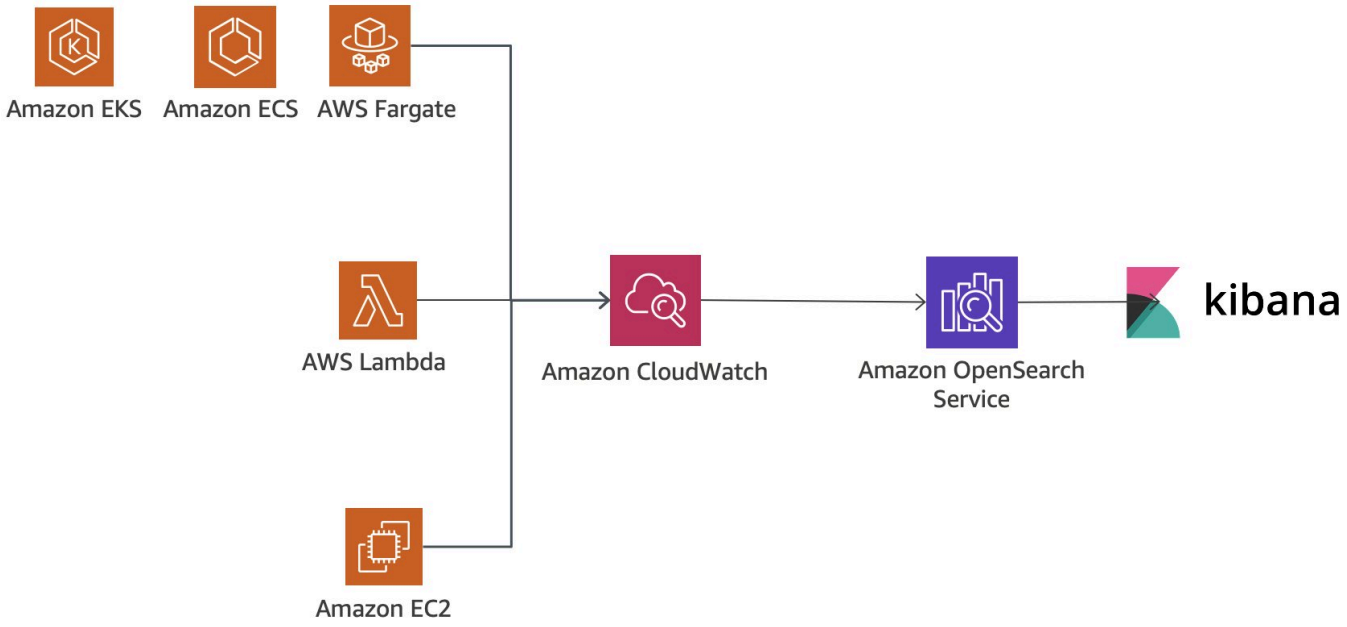

图 14：使用亚马逊 OpenSearch 服务进行日志分析

其他分析选项

为了进一步进行日志分析，完全托管的数据仓库服务 Amazon Redshift 和可扩展的商业智能服务 Quick 提供了有效的解决方案。 QuickSight 可轻松连接各种 AWS 数据服务，例如 Redshift、RDS、Aurora、EMR、DynamoDB、Amazon S3 和 Kinesis，从而简化了数据访问。

CloudWatch 日志可以将日志条目流式传输到 Amazon Data Firehose，这是一项用于提供实时流数据的服务。 QuickSight 然后使用存储在 Redshift 中的数据进行全面分析、报告和可视化。

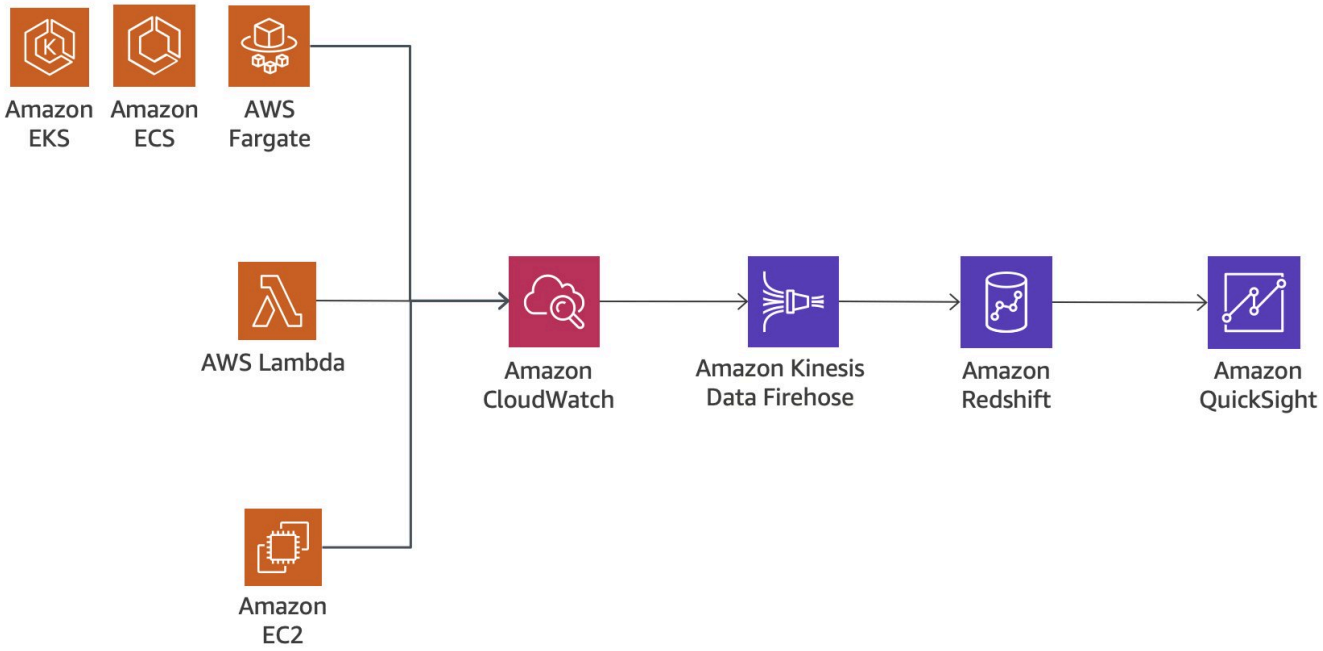

图 15：使用 Amazon Redshift 和 Quick 进行日志分析

此外，当日志存储在对象存储服务 S3 存储桶中时，可以将数据加载到 Redshift 或 EMR（基于云的大数据平台）等服务中，从而可以对存储的日志数据进行全面分析。

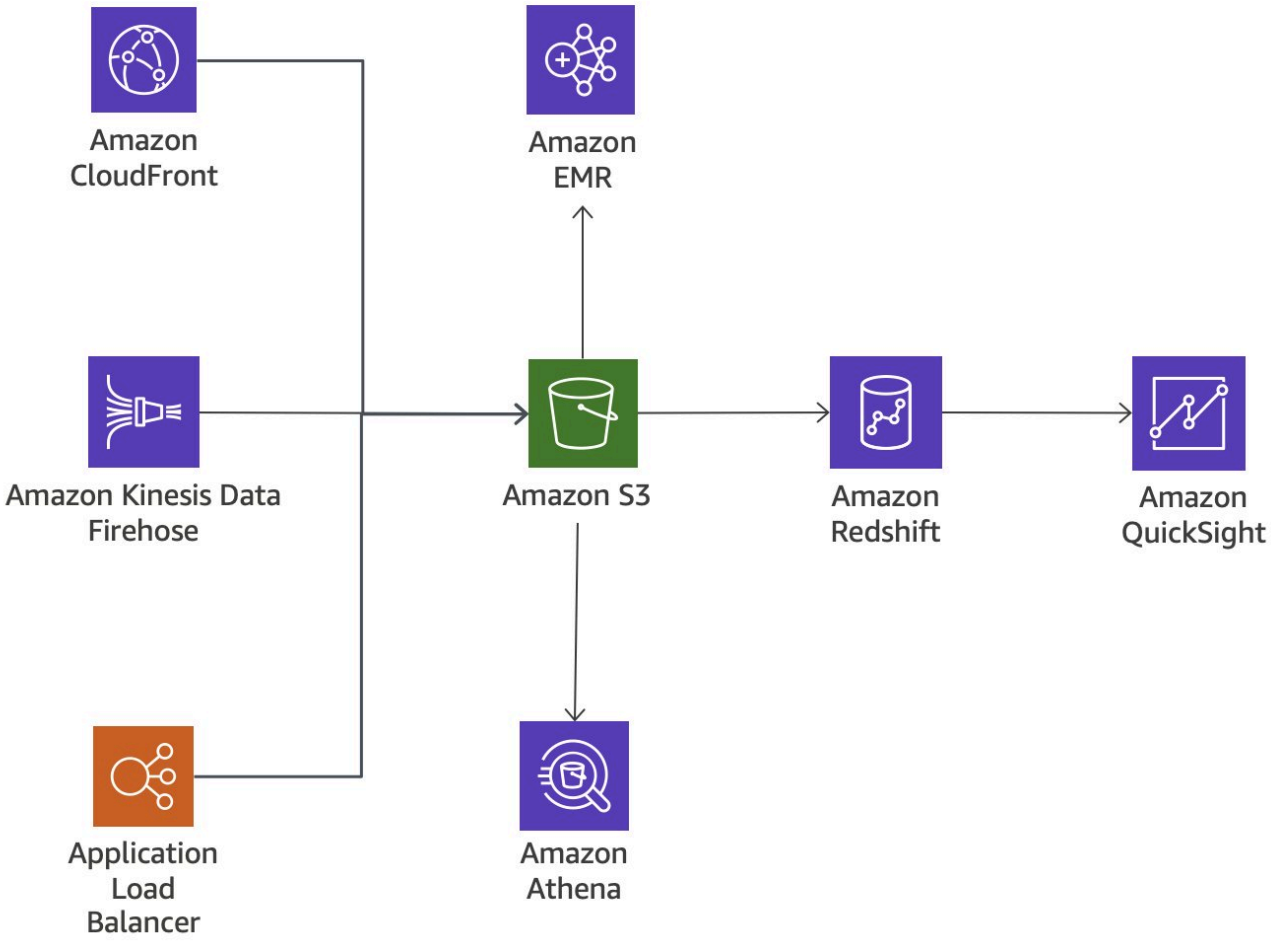

图 16：简化日志分析：从 AWS 服务到 QuickSight

管理微服务通信中的闲聊

闲聊是指微服务之间的过度通信，由于网络延迟增加，这可能会导致效率低下。对于一个运转良好的系统来说，有效管理聊天是至关重要的。

管理闲聊的一些关键工具是 REST APIs、HTTP 和 APIs gRPC。 APIsREST APIs 提供了一系列高级功能，例如 API 密钥、按客户端限制、请求验证、 AWS WAF 集成或私有 API 端点。HTTP APIs 的设计功能很少，因此价格更低。有关本主题的更多详细信息以及 REST 和 HTTP 中可用的核心功能列表 APIs，请参阅在 RES T APIs APIs 和 HTTP 之间进行选择 APIs。

由于其广泛使用，微服务通常使用基于 HTTP 的 REST 进行通信。但是在高容量情况下，REST 的开销可能会导致性能问题。这是因为通信使用 TCP 握手，这是每个新请求所必需的。在这种情况下，gRPC API 是更好的选择。gRPC 可以减少延迟，因为它允许通过单个 TCP 连接发送多个请求。gRPC 还支持双向流式传输，允许客户端和服务器同时发送和接收消息。这可以提高通信效率，特别是对于大型或实时数据传输。

如果尽管选择了正确的 API 类型，但仍然存在闲聊，则可能需要重新评估您的微服务架构。整合服务或修改域名模型可以减少闲聊并提高效率。

使用协议和缓存

微服务通常使用 gRPC 和 REST 等协议进行通信（参见上一节沟通机制。）gRPC 使用 HTTP/2 进行传输，而 REST 通常使用 HTTP/1.1。gRPC 使用协议缓冲区进行序列化，而 REST 通常使用 JSON 或 XML。为了减少延迟和通信开销，可以应用缓存。诸如 Amazon ElastiCache 或 API Gateway 中的缓存层之类的服务可以帮助减少微服务之间的调用次数。

审核

在微服务架构中，了解所有服务的用户操作至关重要。 AWS 提供了诸如 AWS CloudTrail用于记录所有 API 调用的工具 AWS AWS CloudWatch，以及用于捕获应用程序日志的工具。这使您可以跟踪变化并分析微服务中的行为。Amazon EventBridge 可以对系统变化做出快速反应，通知合适的人员，甚至可以自动启动工作流程来解决问题。

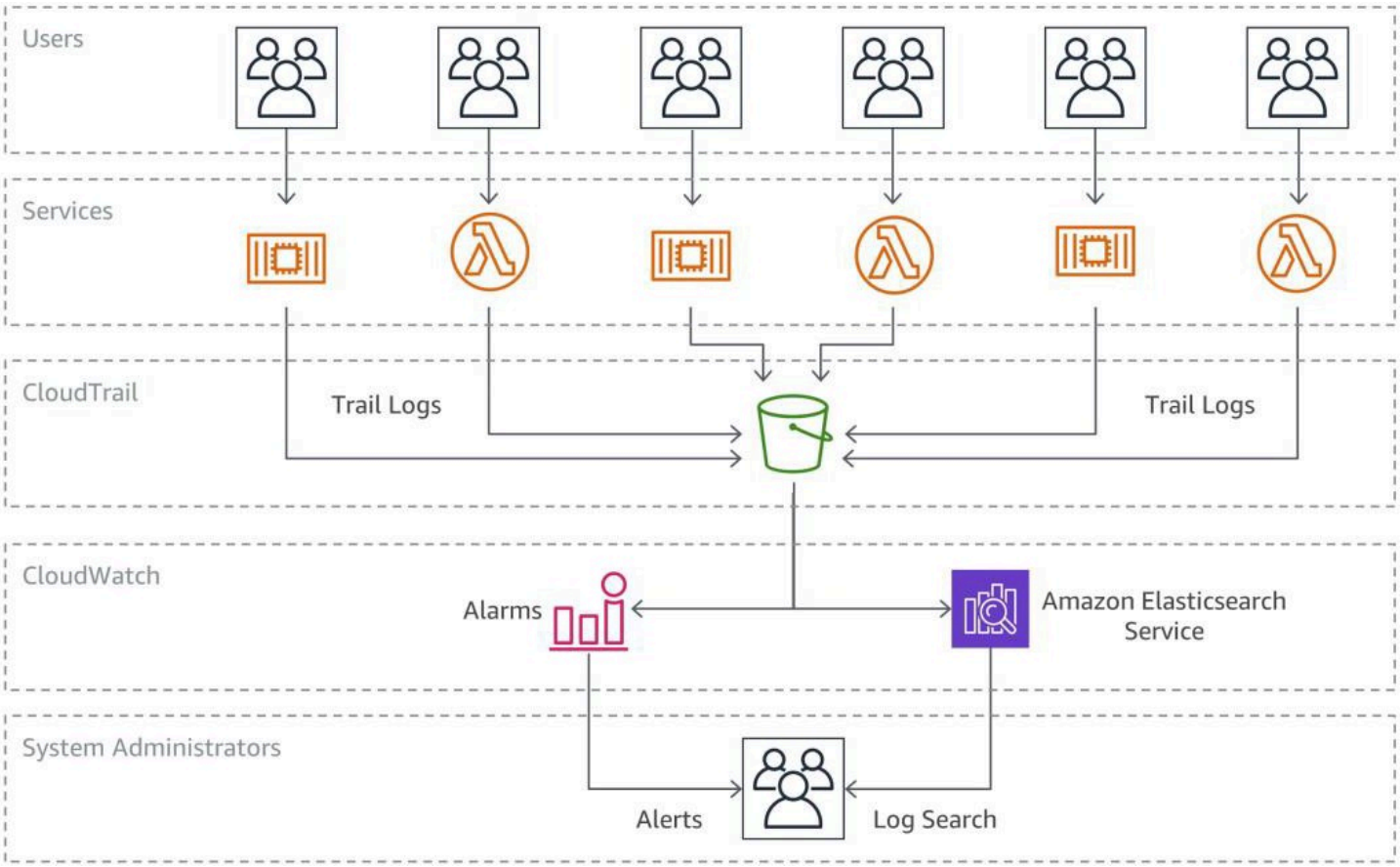

图 17：对您的微服务进行审计和修复

资源库存和变更管理

在基础设施配置快速演变的敏捷开发环境中，自动审计和控制至关重要。 AWS Config 规则 提供一种托管方法来监控跨微服务的这些变化。它们允许定义特定的安全策略，这些策略可以自动检测、跟踪和发送违反策略的警报。

例如，如果将微服务中的 API Gateway 配置更改为接受入站 HTTP 流量，而不仅仅是 HTTPS 请求，AWS Config 则预定义的规则可以检测到这种安全违规行为。它会记录更改以供审计，并触发 SNS 通知，从而恢复合规状态。

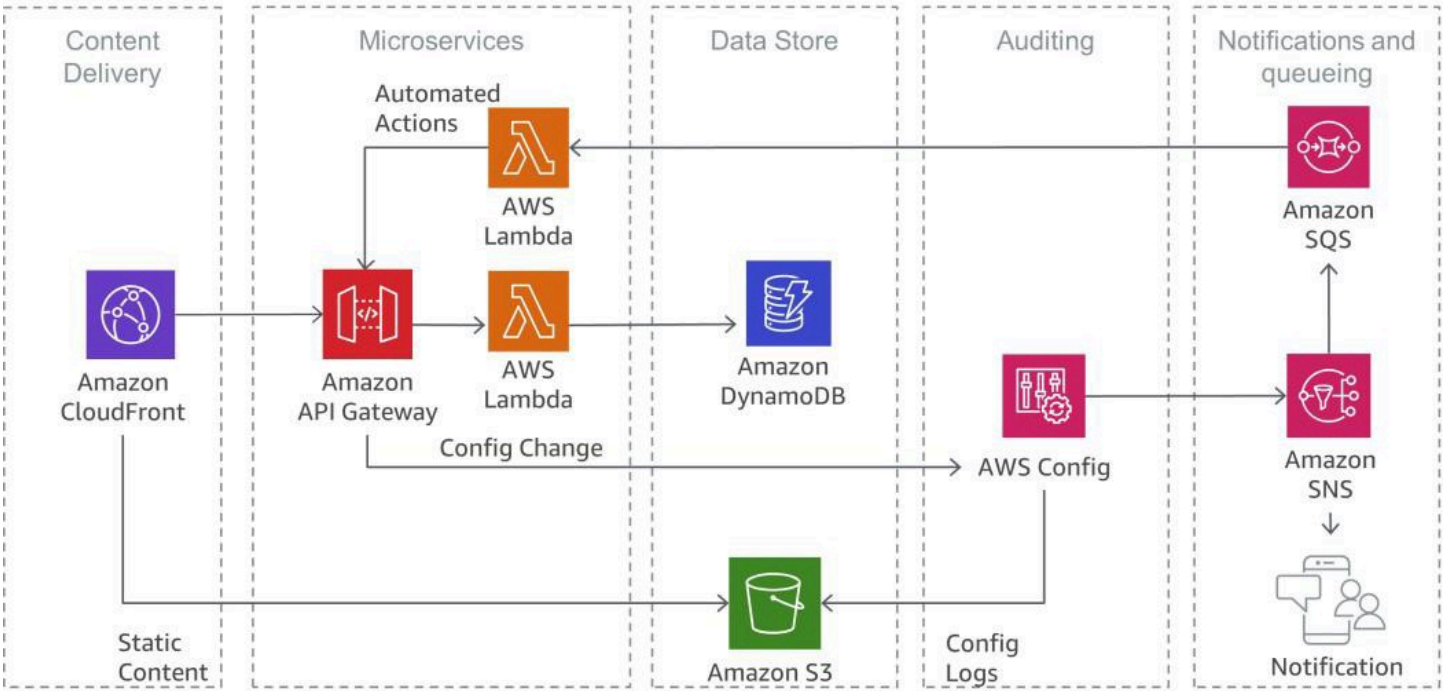

图 18：使用以下方法检测安全违规行为 AWS Config

结论

微服务架构是一种多功能的设计方法，可替代传统的单片系统，它有助于扩展应用程序、提高开发速度和促进组织发展。凭借其适应性，它可以使用容器、无服务器方法或两者的混合来实现，根据特定需求量身定制。

但是，这不是 one-size-fits-all解决方案。考虑到架构复杂性和运营需求的潜在增加，每个用例都需要进行细致的评估。但是，从战略角度来看，微服务的好处可以大大超过这些挑战。关键在于主动规划，尤其是在可观察性、安全性和变更管理领域。

同样重要的是要注意，除了微服务之外，还有完全不同的架构框架，例如生成式人工智能架构，例如检索增强生成 (RAG)，它提供了一系列最能满足您需求的选项。

AWS，凭借其强大的托管服务套件，使团队能够构建高效的微服务架构并有效地最大限度地降低复杂性。本白皮书旨在指导您完成相关 AWS 服务和关键模式的实施。目标是为您提供利用微服务的强大功能所需的知识 AWS，使您能够利用微服务的优势并改变您的应用程序开发之旅。

贡献者

以下个人和组织参与了本文档的编撰：

- Sascha Möllering，解决方案架构，亚马逊 Web Services
- Christian Müller，解决方案架构，亚马逊 Web Services
- Matthias Jung，解决方案架构，亚马逊 Web Services
- Peter Dalbhanjan，解决方案架构，亚马逊 Web Services
- 彼得·查普曼，解决方案架构，亚马逊 Web Services
- Christoph Kassen，解决方案架构，亚马逊 Web Services
- Umair Ishaq，解决方案架构，亚马逊 Web Services
- Rajiv Kumar，解决方案架构，亚马逊 Web Services
- Ramesh Dwarakanath，解决方案架构，亚马逊 Web Services
- 安德鲁·沃特金斯，解决方案架构，亚马逊 Web Services
- Yann Stoneman，解决方案架构，亚马逊 Web Services
- Mainak Chaudhuri，解决方案架构，亚马逊 Web Services
- Gaurav Acharya，解决方案架构，亚马逊 Web Services

文档历史记录

如需获取有关该白皮书更新的通知，请订阅 RSS 信息源。

变更	说明	日期
主要更新	添加了有关 AWS 客户碳足迹工具、亚马逊 AWS AppSync（GraphQL）EventBridge、AWS Lambda 图层、Lambda 、大型语言模型（ ）、适用于 Apache Kafka SnapStart 的亚马逊托管流媒体 (MSK) LLMs、亚马逊 Apache Airflow 托管工作流程 (MWAA)、亚马逊 VPC Lattice 等的信息。AWS AppConfig增加了关于成本优化和可持续性的单独章节。	2023 年 7 月 31 日
次要更新	将 Well-Architected 添加到摘要中。	2022 年 4 月 13 日
已更新白皮书	整合了亚马逊 EventBridge、AWS、AMP OpenTelemetry、AMG、Container Insights、细微的文本更改。	2021 年 11 月 9 日
次要更新	调整了页面布局	2021 年 4 月 30 日
次要更新	细微的文字更改。	2019 年 8 月 1 日
已更新白皮书	整合亚马逊 EKS、AWS Fargate、亚马逊 MQ、AWS、PrivateLink AWS App Mesh、AWS Cloud Map	2019 年 6 月 1 日

已更新白皮书	集成 AWS Step Functions 、AWS X-Ray 和 ECS 事件流。	2017 年 9 月 1 日
初次发布	已发布在 AWS 上实现微服务。	2016 年 12 月 1 日

> **Note**
>
> 要订阅 RSS 更新，您必须为当前使用的浏览器启用 RSS 插件。

版权声明

客户有责任对本文档中的信息进行单独评测。本文件：(a) 仅供参考，(b) 代表当前 AWS 的产品供应和做法，如有更改，恕不另行通知，以及 (c) 不产生其关联公司、供应商或许可方的任何承诺或保证。AWS AWS 产品或服务 “按原样” 提供，不附带任何形式的担保、陈述或条件，无论是明示还是暗示。对客户的责任和责任由 AWS 协议控制，本文档不属于其客户之间的任何协议，也不会对其 AWS 进行修改。 AWS

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

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。