AWS 白皮书

在 AWS 上实施微服务

在 AWS 上实施微服务: AWS 白皮书

Table of Contents

在 AWS 上实施微服务

发布日期：2021 年 11 月 9 日（文档历史记录和贡献者）

摘要

微服务是一种用于软件开发的架构和组织方法，旨在加快部署周期、促进创新和所有权、提高软件应用程序的可维护性和可扩展性，以及通过使用敏捷方法（该方法可帮助团队独立工作）来扩大交付软件和服务的组织的规模。使用微服务方法时，软件由小型服务组成，这些服务通过明确定义且可独立部署的应用程序编程接口 (API) 进行通信。这些服务归小型自主团队所有。这种敏捷方法是成功扩大组织规模的关键。

当AWS客户构建微服务时已观察到三种常见模式：API 驱动、事件驱动和数据流。本白皮书介绍所有这三种方法、总结微服务的共同特性、讨论构建微服务的主要挑战，并描述产品团队如何使用 Amazon Web Services (AWS) 来克服这些挑战。

由于本白皮书中讨论的各种主题的性质（包括数据存储、异步通信和服务发现）相当复杂，因此，建议您在进行架构选择之前，除了考虑提供的指导之外，还考虑其应用程序的具体要求和使用场景。

介绍

微服务架构并不是一种全新的软件工程方法，而是各种经过验证的成功概念的组合，例如：

- 敏捷软件开发
- 面向服务的架构
- API 优先设计
- 持续集成/持续交付 (CI/CD)

在许多情况下，微服务使用十二要素应用程序设计模式。

本白皮书首先介绍高度可扩展且具有容错能力的微服务架构的不同方面（用户界面、微服务实施和数据存储），以及如何利用容器技术在AWS上构建该架构。然后，本白皮书建议使用相应的AWS服务来实施典型的无服务器微服务架构，以降低操作复杂性。

无服务器是指遵循以下原则的一种操作模型：

- 无需预置或管理基础设施

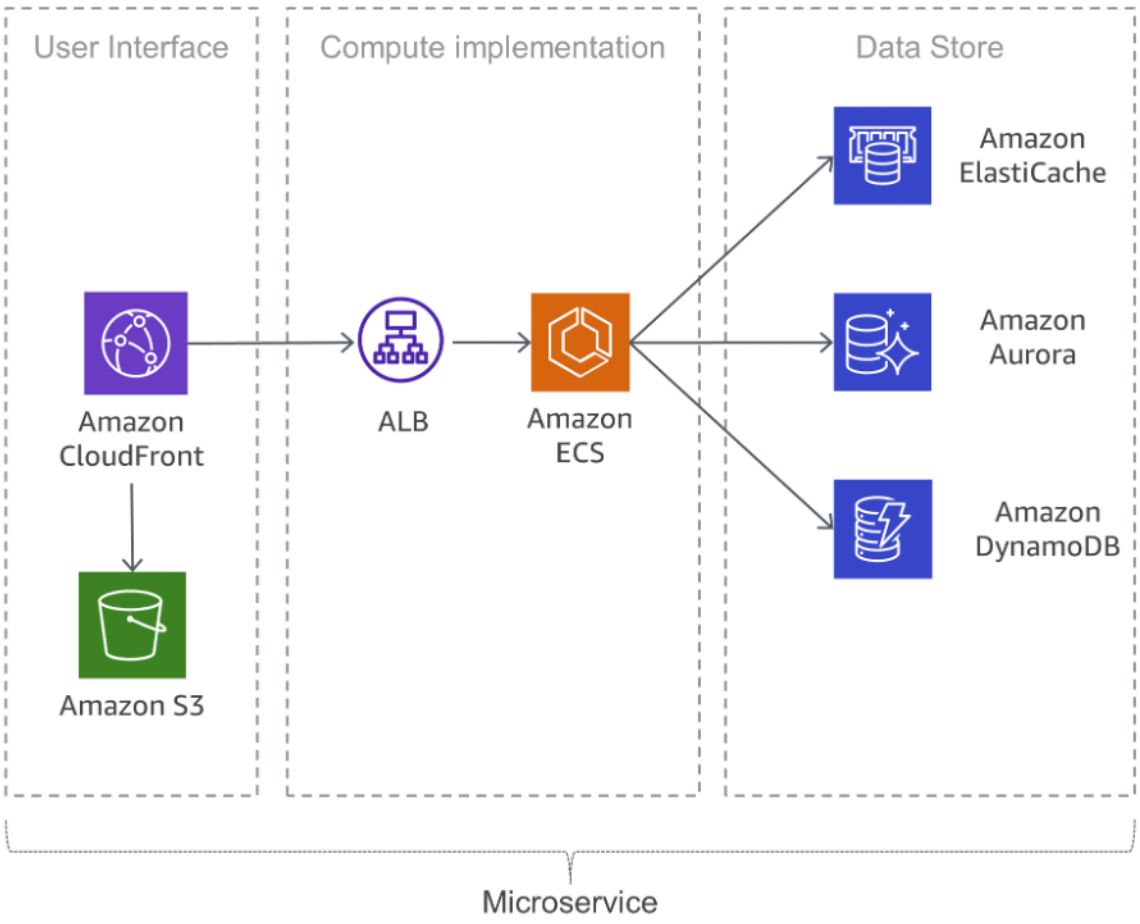
- 按消耗单位自动扩展
- 按价值付费计费模型
- 内置可用性和容错能力

最后，本白皮书介绍整个系统并讨论微服务架构的跨服务方面，如分布式监控和审计、数据一致性和异步通信。

本白皮书仅重点介绍在 AWS 云中运行的工作负载。它不包括混合方案或迁移策略。有关迁移的更多信息，请参阅容器迁移方法白皮书。

AWS 上的微服务架构

典型的整体式应用程序使用不同的层（用户界面 (UI) 层、业务层和持久层）构建而成。微服务架构的核心思想是将功能划分为紧密相关的垂直架构 – 不是通过技术层，而是通过实施特定域。下图描述了 AWS 上的典型微服务应用程序的参考架构。



AWS 上的典型微服务应用程序

主题

- 用户界面
- 微服务
- 数据存储

用户界面

现代 Web 应用程序通常使用 JavaScript 框架来实施与表述性状态转移 (REST) 或 RESTful API 通信的单页应用程序。静态 Web 内容可以使用 Amazon Simple Storage Service (S3) 和 Amazon CloudFront 提供。

由于是从最近的边缘站点向微服务的客户端提供内容，并且这些客户端从与源具有优化连接的缓存或代理服务器获得响应，因此延迟得以显著降低。但是，彼此相邻运行的微服务并不会受益于内容分发网络。在某些情况下，这种方法实际上可能会额外增加延迟。最佳实践是实施其他缓存机制，以减少干扰并最大程度地降低延迟。有关更多信息，请参阅the section called “干扰”主题。

微服务

API 是微服务的前门，这意味着 API 充当一组编程接口背后的应用程序逻辑的入口点，通常是 RESTful Web 服务 API。此 API 接受并处理来自客户端的调用，并可能实施流量管理、请求筛选、路由、缓存、身份验证和授权等功能。

微服务实施

AWS 集成了支持微服务开发的构建块。两种常见的方法是使用 AWS Lambda 和带有 AWS Fargate 的 Docker 容器。

使用 AWS Lambda，您上载代码，并让 Lambda 负责处理运行和扩展实施所需的所有事项，以满足您的实际需求曲线和高可用性。不需要管理基础设施。Lambda 支持多种编程语言，并且可以从其他 AWS 服务调用，或者直接从任何 Web 或移动应用程序调用。AWS Lambda 的最大优势之一是您可以快速转变：您可以专注于业务逻辑，因为安全性和可扩展性由 AWS 负责。Lambda 的坚持己见造就了可扩展的平台。

基于容器的部署是减少部署操作工作的常用方法。由于具备可移植性、高生产力和效率等优势，Docker 等容器技术在过去几年中越来越受欢迎。容器的学习曲线可能很陡，并且您必须考虑 Docker 镜像的安全修复和监控。Amazon Elastic Container Service (Amazon ECS) 和 Amazon Elastic Kubernetes Service (Amazon EKS) 使您不必自行安装、操作和扩展集群管理基础设施。通过 API 调用，您可以启动和停止支持 Docker 的应用程序、查询集群的完整状态，以及访问许多常见功能，如安全组、负载均衡、Amazon Elastic Block Store (Amazon EBS) 卷和 AWS Identity and Access Management (IAM) 角色。

AWS Fargate 是适用于容器的无服务器计算引擎，可与 Amazon ECS 和 Amazon EKS 配合使用。使用 Fargate，您不再需要担心如何为容器应用程序预置足够的计算资源。Fargate 可以启动数万个容器，并能轻松扩展，从而运行最关键的任务型应用程序。

Amazon ECS 支持容器置放策略和限制，以自定义 Amazon ECS 置放和结束任务的方式。任务置放限制是置放任务时会考虑的规则。您可以将属性（它们实际上是键/值对）与容器实例相关联，然后使用限制根据这些属性来置放任务。例如，您可以使用限制根据实例类型或实例功能（如支持 GPU 的实例）来置放某些微服务。

Amazon EKS 运行的是最新版本的开源 Kubernetes 软件，因此您可以使用 Kubernetes 社群中现有的所有插件和工具。运行在 Amazon EKS 上的应用程序与运行在所有标准 Kubernetes 环境（无论是本地数据中心还是公有云）上的应用程序完全兼容。Amazon EKS 集成了 IAM 和 Kubernetes，使您能够向 Kubernetes 中的原生身份验证系统注册 IAM 实体。无需手动设置凭证以向 Kubernetes 控制层面进行身份验证。IAM 集成使您能够使用 IAM 直接向控制层面本身进行身份验证，以精细访问 Kubernetes 控制层面的公有终端节点。

Amazon ECS 和 Amazon EKS 中使用的 Docker 镜像可以存储在 (Amazon Elastic Container Registry (Amazon ECR) 中。使用 Amazon ECR，您无需操作和扩展支持容器注册表所需的基础设施。

持续集成和持续交付 (CI/CD) 是最佳实践，也是 DevOps 计划的重要组成部分，该计划不仅支持快速更改软件，还维护系统的稳定性和安全性。但是，这超出了本白皮书的范围。有关更多信息，请参阅在 AWS 上实现持续集成和持续交付白皮书。

私有链接

AWS PrivateLink 是一项具有高可用性的可扩展技术，使您能够将您的 Virtual Private Cloud (VPC) 以私密方式连接到支持的 AWS 服务、由其他 AWS 账户托管的服务（VPC 终端节点服务）以及支持的 AWS Marketplace 合作伙伴服务。您不需要互联网网关、网络地址转换服务、公有 IP 地址、AWS Direct Connect 连接或 VPN 连接即可与该服务通信。您的 VPC 和服务之间的流量不会脱离 Amazon 网络。

私有链接是提高微服务架构的隔离性和安全性的好方法。例如，微服务可以部署在完全独立的 VPC 中，由负载均衡器作为前端，并通过 AWS PrivateLink 终端节点向其他微服务公开。通过此设置，使用 AWS PrivateLink 时，进出微服务的网络流量从不会穿过公有互联网。这种隔离的一个使用案例包括用于处理敏感数据的服务（例如 PCI、HIPPA 和欧盟/美国隐私护盾）的监管合规性。此外，AWS PrivateLink 允许跨不同账户和 Amazon VPC 连接微服务，无需防火墙规则、路径定义或路由表；简化了网络管理。借助 PrivateLink，软件即服务 (SaaS) 提供商和 ISV 也可以为其基于微服务的解决方案提供全面的操作隔离和安全访问。

数据存储

数据存储用于保存微服务所需的数据。常用的会话数据存储方式是内存缓存，如 Memcached 或 Redis。AWS 将这两种技术作为托管式 Amazon ElastiCache 服务的一部分提供。

在应用程序服务器和数据库之间置放缓存是减少数据库读取负载的一种常用机制，这一机制进而让资源可用于支持更多写入操作。缓存还可以改善延迟。

关系数据库仍非常广泛地用于存储结构化数据和业务对象。AWS 通过 Amazon Relational Database Service (Amazon RDS) 提供六个数据库引擎（Microsoft SQL Server、Oracle、MySQL、MariaDB、PostgreSQL 和 Amazon Aurora）作为托管式服务。

然而，关系数据库并不是为实现无限规模而设计的，这会使得应用技术来支持大量查询变得十分困难和耗时。

NoSQL 数据库更偏重于可扩展性、性能和可用性，而不是关系数据库的一致性。有一点很重要，NoSQL 数据库一般不会强制实施严格的 Schema。数据分布在可水平扩展的分区上，并使用分区键进行检索。

由于单个微服务只有一个既定用途，因此通常具有可能非常适合实现 NoSQL 持久性的简化数据模型。NoSQL 数据库的访问模式与关系数据库不同，理解这一点很重要。例如，无法联接表。如果必须进行联接，则必须在应用程序中实施相应逻辑。您可以使用 Amazon DynamoDB 创建一个数据库表来存储和检索任何大小的数据，并处理任何级别的请求流量。DynamoDB 可以提供响应时间在十毫秒内的性能，但是某些使用案例要求响应时间在微秒级。Amazon DynamoDB Accelerator (DAX) 为访问数据提供了缓存功能。

DynamoDB 还提供了自动扩展功能，以根据实际流量动态调整吞吐量。但是，在某些情况下，由于应用程序中存在持续时间很短的大型活动峰值，很难或无法进行容量规划。对于这些情况，DynamoDB 提供了按需选项，即简单的“按请求付费”定价。DynamoDB 按需选项能够在没有进行容量规划的情况下每秒即时处理数千个请求。

降低操作复杂性

本白皮书前面介绍的架构已在使用托管式服务，但仍需管理 Amazon Elastic Compute Cloud (Amazon EC2) 实例。通过使用完全无服务器的架构，可以进一步减少运行、维护和监控微服务所需完成的操作工作。

主题

- API 实施
- 无服务器微服务
- 灾难恢复
- 高可用性
- 部署基于 Lambda 的应用程序

API 实施

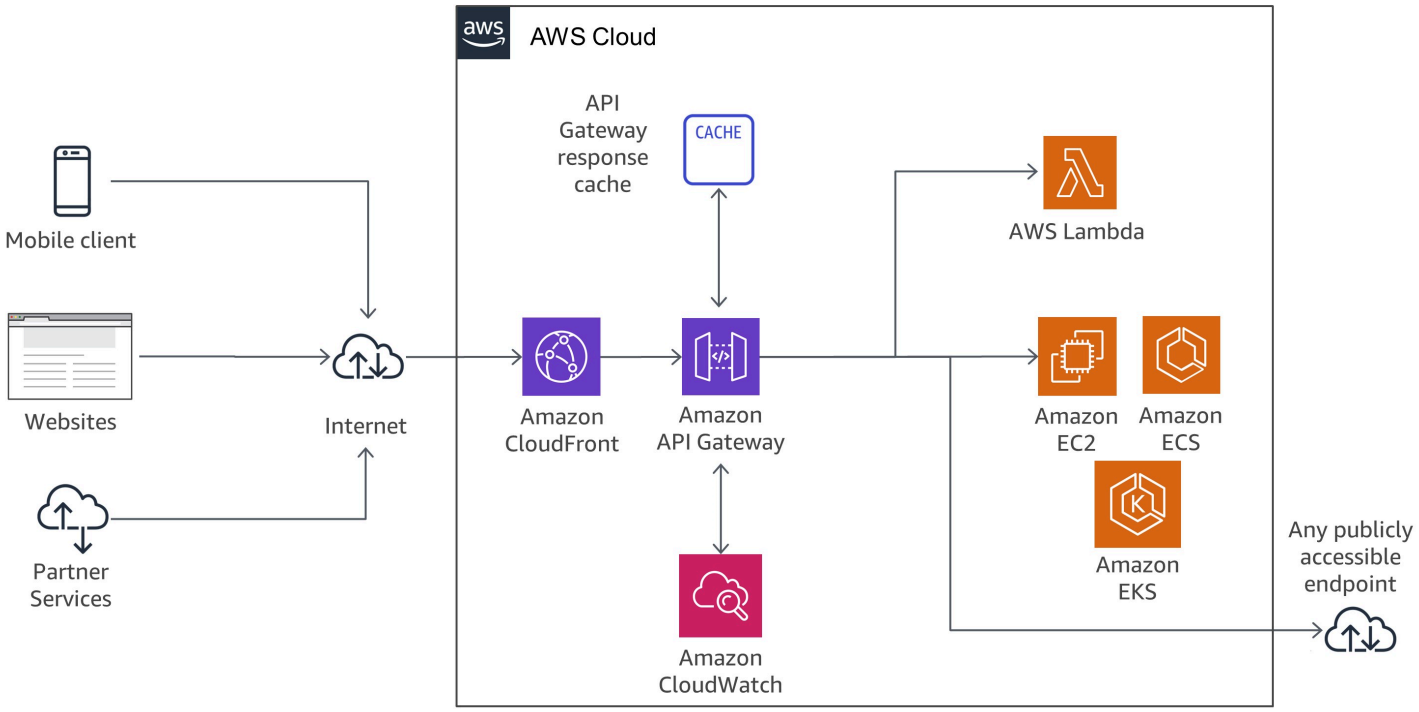
架构、部署、监控、持续改进和维护 API 是相当耗时的任务。有时，需要运行不同版本的 API 以确保所有客户端具有向后兼容性。开发周期的不同阶段（例如，开发、测试和生产）进一步增加了运营工作。

授权是所有 API 的一项重要功能，但通常其构建工作很复杂，而且需要重复性工作。成功发布 API 后，下一个挑战是使用该 API 管理和监控第三方开发人员的生态系统并从中获利。

其他重要功能和挑战包括限制请求以保护后端服务、缓存 API 响应、处理请求和响应转换以及使用 Swagger 等工具生成 API 定义和文档。

Amazon API Gateway 可以解决这些难题并降低创建和维护 RESTful API 的操作复杂性。借助 API Gateway，您可以通过 AWS API 或 AWS 管理控制台导入 Swagger 定义，以编程方式创建 API。API Gateway 充当在 Amazon EC2、Amazon ECS、AWS Lambda 或任何本地部署环境中运行的任何 Web 应用程序的前门。基本上，API Gateway 允许您运行 API 而无需管理服务器。

下图描述了 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将路由到最近的 CloudFront 接入点 (PoP)，以最大程度地降低延迟并提供最佳的用户体验。



API Gateway 调用流程

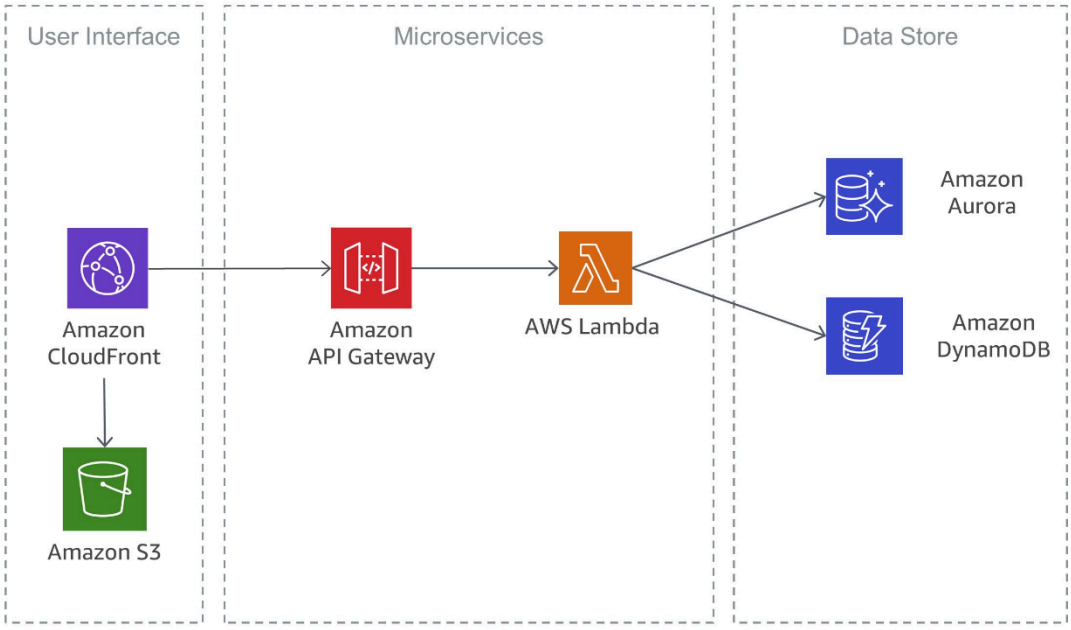
无服务器微服务

"没有任何服务器比无服务器更好管理"。

不使用服务器是消除操作复杂性的最好方法。

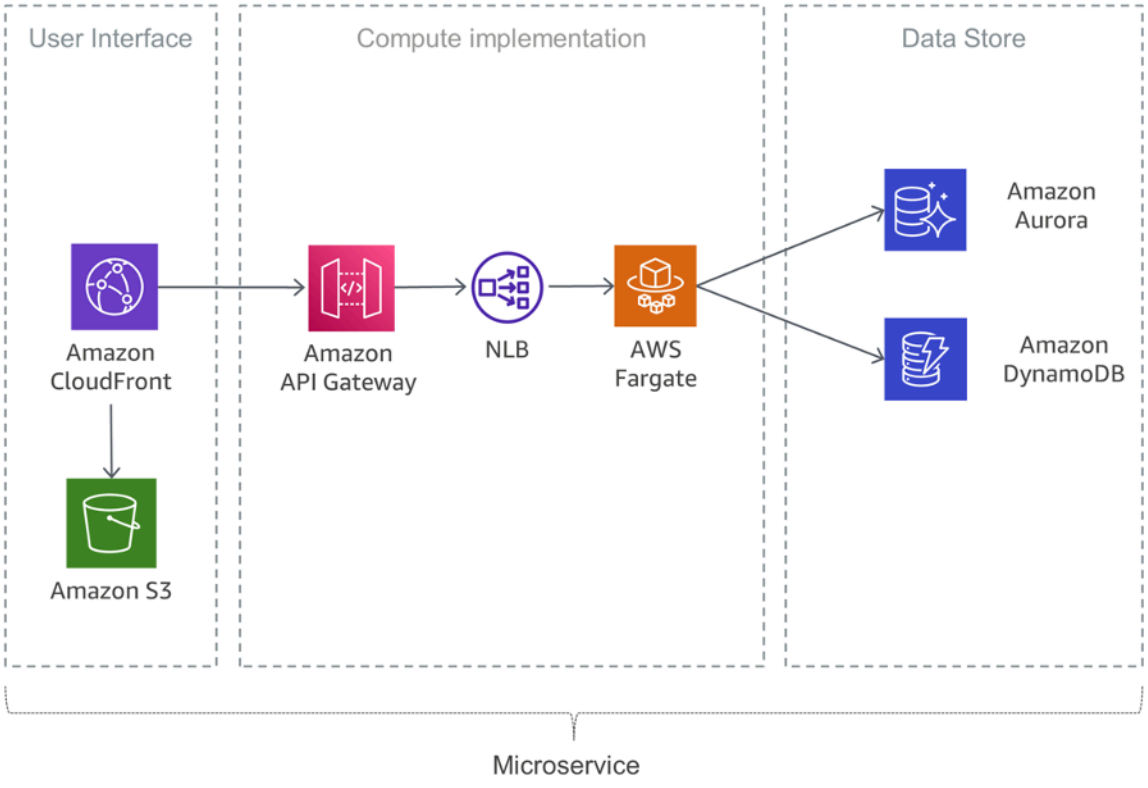
Lambda 与 API Gateway 紧密集成。能够从 API Gateway 向 Lambda 进行同步调用使得创建完全无服务器的应用程序成为可能，这一点将在 Amazon API Gateway 开发人员指南中详细介绍。

下图显示了使用 AWS Lambda 的无服务器微服务的架构，其中整个服务都是基于托管式服务构建的，这消除了实现规模和高可用性设计的架构负担，以及运行和监控微服务底层基础设施的运营工作。



使用 AWS Lambda 的无服务器微服务

下图显示了同样基于无服务器服务的类似实施情况。在本架构中，Docker 容器与 Fargate 结合使用，因此没有必要担心底层基础设施。除了 DynamoDB 之外，还使用了 Amazon Aurora Serverless，这是一种按需自动扩展的 Amazon Aurora 配置（MySQL 兼容版），其中数据库将根据应用程序的需求自动启动、关闭以及扩展或缩减容量。



使用 Fargate 的无服务器微服务

灾难恢复

正如前面在本白皮书的简介中所提到的，典型的微服务应用程序是使用十二要素应用程序模式实现的。"过程"部分指出："十二要素过程是无状态的，不共享任何内容。任何需要持久化的数据都必须存储在有状态的后备服务（通常是数据库）中。"

对于典型的微服务架构，这意味着灾难恢复的主要重点应放在维护应用程序状态的下游服务上。例如，这些可以是文件系统、数据库或队列等。在制定灾难恢复策略时，组织通常会根据恢复时间目标和恢复点目标进行规划。

恢复时间目标是指服务中断和服务恢复之间的最大可接受延迟。此目标确定当服务不可用时可接受的时间窗口，并由组织定义。

恢复点目标是指自上一个数据恢复点以来的最大可接受时间量。此目标确定在上一个恢复点和服务中断之间可接受的数据丢失程度，并由组织定义。

有关更多信息，请参阅 AWS 上工作负载的灾难恢复：云中的恢复白皮书。

高可用性

本节将详细介绍不同计算选项的高可用性。

Amazon EKS 跨多个可用区运行 Kubernetes 控制和数据层面实例以确保高可用性。Amazon EKS 可以自动检测和替换运行状况不佳的控制层面实例，并为它们提供自动版本升级和修补。该控制层面至少包含两个 API 服务器节点和三个 etcd 节点（这三个节点在一个区域内的三个可用区中运行）。Amazon EKS 使用 AWS 区域 架构来维护高可用性。

Amazon ECR 将您的镜像托管在高度可用且高性能的架构中，使您能够可靠地跨可用区为容器应用程序部署镜像。Amazon ECR 与 Amazon EKS、Amazon ECS 和 AWS Lambda 结合使用，可简化从开发到生产的工作流。

Amazon ECS 是一项区域服务，可跨 AWS 区域 内的多个可用区以高可用性方式简化正在运行的容器。Amazon ECS 包括多项计划策略，这些策略可根据您的资源需求（如 CPU 或 RAM）和可用性要求将容器放入各个集群。

AWS Lambda 在多个可用区中运行您的函数，以确保在单一区域中服务中断时能够处理事件。如果将函数配置为连接到账户中的 Virtual Private Cloud (VPC)，请在多个可用区中指定子网以确保高可用性。

部署基于 Lambda 的应用程序

您可以使用 AWS CloudFormation 定义、部署和配置无服务器应用程序。

AWS Serverless Application Model (AWS SAM) 是用于定义无服务器应用程序的简便方法。AWS SAM 本身受 CloudFormation 支持，并且为表达无服务器资源定义简化的语法。要部署应用程序，请指定您需要作为应用程序的一部分的资源及其在 CloudFormation 模板中的相关权限策略，打包部署构件，然后部署该模板。SAM Local 是以 AWS SAM 为基础的 AWS Command Line Interface (AWS CLI) 工具，在将无服务应用程序上载到 Lambda 运行时前，为您提供在本地开发、测试和分析它们的环境。您可以使用 AWS SAM Local 创建一种模拟 AWS 运行时环境的本地测试环境。

分布式系统组件

在了解 AWS 如何解决与各个微服务相关的挑战之后，重点转向解决服务发现、数据一致性、异步通信以及分布式监控和审计等跨服务挑战。

主题

服务发现

微服务架构的主要挑战之一是使各个服务能够彼此发现并相互交互。微服务架构的分布式特性不仅使服务更难通信，还带来了其他挑战，例如，检查这些系统的运行状况，以及宣布新应用程序何时可用。您还必须决定元存储信息（例如，应用程序可以使用的配置数据）的存储方式和位置。在本部分中，将介绍针对基于微服务的架构在 AWS 上执行服务发现的几种技术。

基于 DNS 的服务发现

Amazon ECS 现在包含集成式服务发现，这使您的容器化服务可以轻松地发现彼此并相互连接。

之前，为了确保服务能够发现彼此并相互连接，您必须基于 Amazon Route 53、AWS Lambda 和 ECS 事件流配置和运行自己的服务发现系统，或者将每个服务连接到负载均衡器。

Amazon ECS 使用 Route 53 自动命名 API 创建和管理服务名称注册表。名称将自动映射到一组 DNS 记录，以便您可以在代码中按名称引用服务，并编写 DNS 查询，从而在运行时将名称解析到服务的终端节点。您可以在服务的任务定义中指定运行状况检查条件，Amazon ECS 将确保服务查找工具只返回运行状况良好的服务终端节点。

此外，还可以对 Kubernetes 管理的服务使用统一服务发现。为了实现这种集成，AWS 参与了外部 DNS 项目，这是一个 Kubernetes 孵化器项目。

另一个选项是使用 AWS Cloud Map 的功能。AWS Cloud Map 扩展了自动命名 API 的功能，方法是为互联网协议 (IP)、统一资源定位符 (URL) 和 Amazon Resource Name (ARN) 等资源提供服务注册表，

以及提供基于 API 的服务发现机制，该机制可以更快传播更改并能够使用属性缩小已发现资源集。现有的 Route 53 自动命名资源将自动升级到 AWS Cloud Map。

第三方软件

实现服务发现的另一个方法是使用 HashiCorp Consul、etcd 或 Netflix Eureka 等第三方软件。这三个示例都是可靠的分布式键值存储。对于 HashiCorp Consul，有一个AWS快速入门，它设置了一种灵活且可扩展的 AWS 云环境，并将 HashiCorp Consul 自动启动到您选择的配置中。

服务网格

在高级微服务架构中，实际的应用程序可以由数百甚至数千个服务组成。应用程序中最复杂的部分通常不是实际的服务本身，而是这些服务之间的通信。服务网格是用于处理服务间通信的附加层，负责监控和控制微服务架构中的流量。这样一来，服务发现等任务可以完全由该层来处理。

通常，服务网格分为数据层面和控制层面。数据层面由一组智能代理组成，这些代理与应用程序代码一起部署，作为一个特殊的边车代理，用于拦截微服务之间的所有网络通信。控制层面负责与代理通信。

服务网格是透明的，这意味着应用程序开发人员不需要知道该附加层，也不需要更改现有的应用程序代码。AWS App Mesh 是一种服务网格，可提供应用程序级网络，使您的服务能够跨多种类型的计算基础设施彼此通信。App Mesh 对服务的通信方式进行了标准化，为您提供全面的可视性，并确保应用程序的高可用性。

您可以将 AWS App Mesh 与 AWS Fargate、Amazon ECS、Amazon EKS 上运行的现有或新微服务以及AWS上自行管理的 Kubernetes 一起使用。App Mesh 作为单个应用程序监控和控制跨集群、编排系统或 VPC 运行的微服务的通信，而不需要更改任何代码。

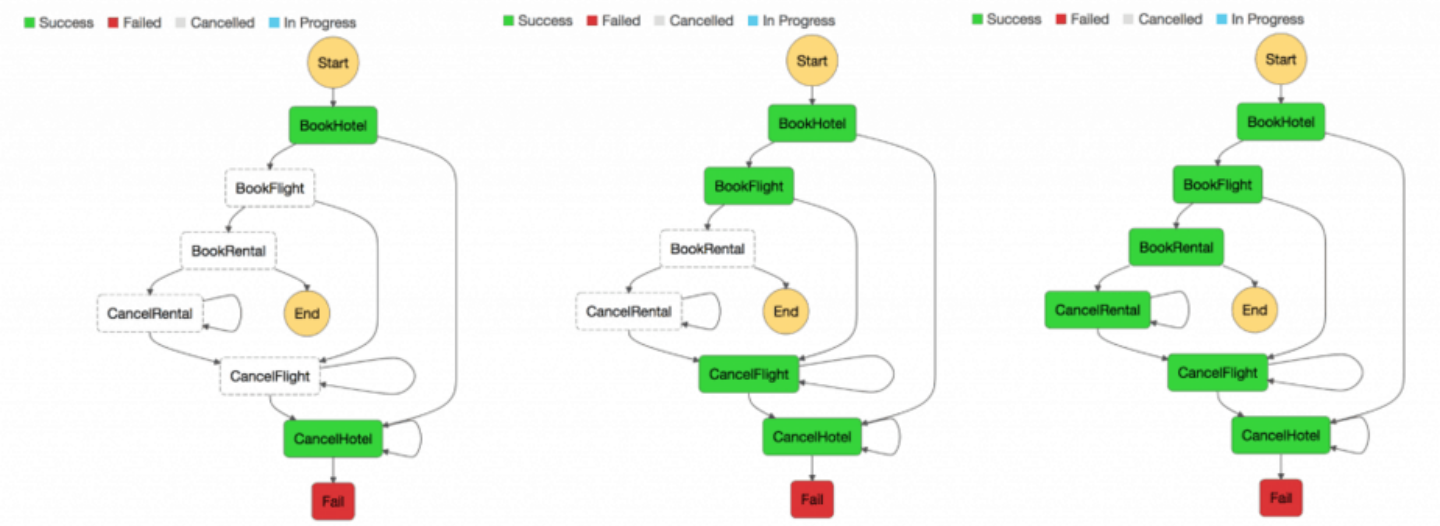
分布式数据管理

整体式应用程序通常由大型关系数据库支持，该数据库定义了所有应用程序组件通用的单个数据模型。在微服务方法中，这样一个中央数据库将阻止构建分散和独立组件的目标。每个微服务组件都应具有自己的数据持久层。

然而，分布式数据管理带来了新的挑战。根据 CAP 法则可知，分布式微服务架构本质上以一致性为代价来提高性能，因此需要实现最终一致性。

在分布式系统中，业务事务可以跨多个微服务。由于它们不能利用单个 ACID 事务，因此最终可能是部分执行。在这种情况下，我们需要一些控制逻辑来重做已处理的事务。为此，通常会使用分布式 Saga

模式。在业务事务失败的情况下，Saga 会编排一系列补偿事务，以撤消前面事务所做的更改。AWS Step Functions 使您能够轻松实施 Saga 执行协调器，如下图所示。



Saga 执行协调器

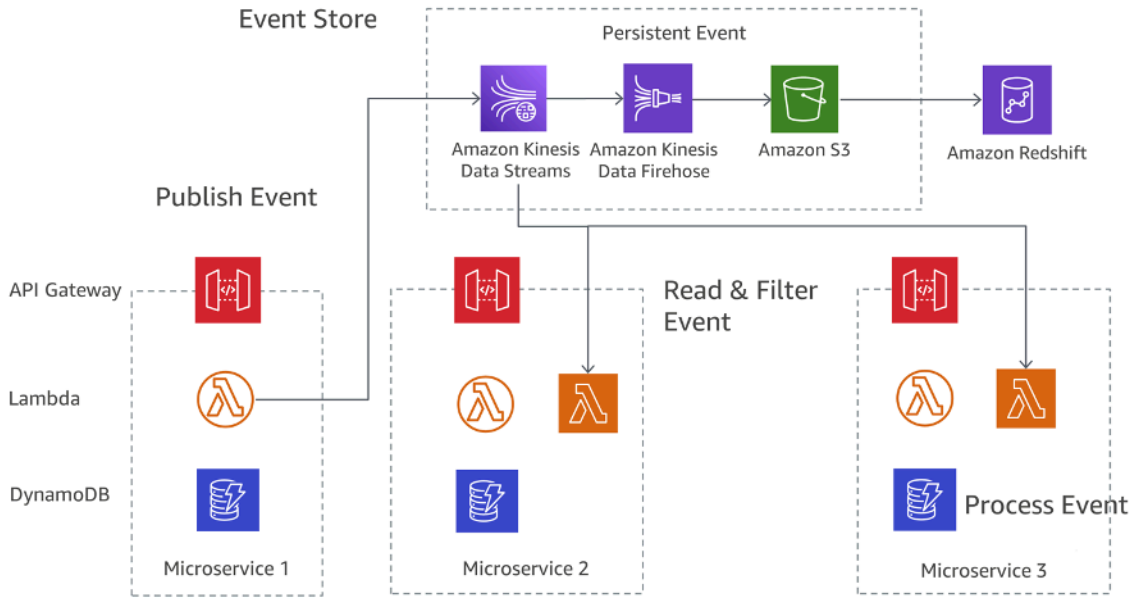
构建经核心数据管理工具和程序策管的关键参考数据集中存储，为微服务提供了一种同步其关键数据并可能回滚状态的方法。将 Lambda 与计划的 Amazon CloudWatch Events 结合使用，您可以构建简单的清理和重复数据删除机制。

状态变化影响多个微服务是很常见的。在这些情况下，事实证明，事件溯源是一种有用的模式。事件溯源背后的核心思想是，将每个应用程序更改表示并保存为事件记录。不是持久保存应用程序状态，而是将数据存储为事件流。数据库事务日志记录和版本控制系统是事件溯源的两个众所周知的示例。事件溯源有几个好处：可以确定和重建任何时间点的状态。它自然会生成持久的审计跟踪，也便于调试。

在微服务架构的上下文中，事件溯源可以通过使用发布/订阅模式解耦应用程序的不同部分，并且它将相同的事件数据输送到独立微服务的不同数据模型中。事件溯源经常与命令和查询责任分割 (CQRS) 模式结合使用，以将读取工作负载与写入工作负载分离，并优化二者的性能、可扩展性和安全性。在传统的数据管理系统中，命令和查询是针对同一个数据存储库运行的。

下图显示如何在 AWS 上实施事件溯源模式。Amazon Kinesis Data Streams 充当中央事件存储的主要组件，可捕获应用程序更改作为事件，并将其保存在 Amazon S3 上。此图描述了三种不同的微服务，包括 Amazon API Gateway、AWS Lambda 和 Amazon DynamoDB。箭头表示事件流：当微服务 1 经历事件状态更改时，它会通过在 Kinesis Data Streams 中写入消息来发布事件。所有微服务都会在 AWS Lambda 中运行自己的 Kinesis Data Streams 应用程序，该应用程序读取消息的副本，根据微服务的相关性进行筛选，并可能转发以供进一步处理。如果您的函数返回一个错误，则 Lambda 将重试批处理，直到处理成功或数据过期。为避免分片停滞，可以将事件源映射配置为以较小的批处理

大小重试，限制重试次数或者丢弃太早的记录。要保留丢弃的事件，可以配置事件源映射，以将有关失败批处理的详细信息发送到 Amazon Simple Queue Service (Amazon SQS) 队列或 Amazon Simple Notification Service (Amazon SNS) 主题。



AWS上的事件溯源模式

Amazon S3 持久存储所有微服务中的所有事件，当进行调试、恢复应用程序状态或审计应用程序更改时，它是唯一的事实来源。有两个主要原因可能导致多次向您的 Kinesis Data Streams 应用程序提供记录：创建者重试和使用者重试。您的应用程序必须预计并适当地应对多次处理单个记录的问题。

配置管理

在具有数十种不同服务的典型微服务架构中，每项服务都需要访问向服务公开数据的若干下游服务和基础设施组件。例如消息队列、数据库和其他微服务。关键挑战之一是如何以一致的方式配置每项服务，以提供有关与下游服务和基础设施的连接的信息。此外，配置还应包含有关服务运行环境的信息，而不需要重新启动应用程序以使用新的配置数据。

十二要素应用程序模式的第三个原理涵盖了这个主题：“十二要素应用程序将配置存储在环境变量（通常缩写为 env vars 或 env）中”。对于 Amazon ECS，可以使用环境容器定义参数（此参数映射到 docker 运行的 `--env` 选项）将环境变量传递给容器。通过使用 `environmentFiles` 容器定义参数列出一个或多个包含环境变量的文件，可以将环境变量批量传递到容器。该文件必须托管在 Amazon S3 中。在 AWS Lambda 中，运行时使环境变量可用于您的代码，并设置其他环境变量，这些变量包含有

关函数和调用请求的信息。对于 Amazon EKS，您可以在相应的一组容器 (pod) 的配置清单的 env 字段中定义环境变量。使用环境变量的另一种方法是使用 ConfigMap。

异步通信和轻量级消息收发

在传统的整体式应用程序中，通信是直接的：应用程序的某个部件使用方法调用或内部事件分发机制与其他部件通信。如果使用解耦的微服务实施相同的应用程序，则应用程序的不同部件之间的通信必须使用网络通信实现。

基于 REST 的通信

HTTP/S 协议是实现微服务之间同步通信的最常用的方式。在大多数情况下，RESTful API 使用 HTTP 作为传输层。REST 架构风格依赖于无状态通信、统一界面和标准方法。

使用 API Gateway，您可以创建 API 来充当应用程序用来从后端服务访问数据、业务逻辑或功能的前门。API 开发人员可以创建能够访问 AWS、其他 Web 服务以及存储在 AWS 云中的数据的 API。使用 API Gateway 服务定义的 API 对象是一组资源和方法。

资源是 API 域内的类型化对象，可能具有关联的数据模型或与其他资源的关系。每个资源都可以配置为响应一种或多种方法，即，GET、POST 或 PUT 等标准 HTTP 动词。REST API 可以部署到不同阶段，进行版本控制，以及克隆到新版本。

API Gateway 负责处理多达数十万个并发 API 调用的接受和处理过程中涉及的所有任务，包括流量管理、授权和访问控制、监控以及 API 版本管理。

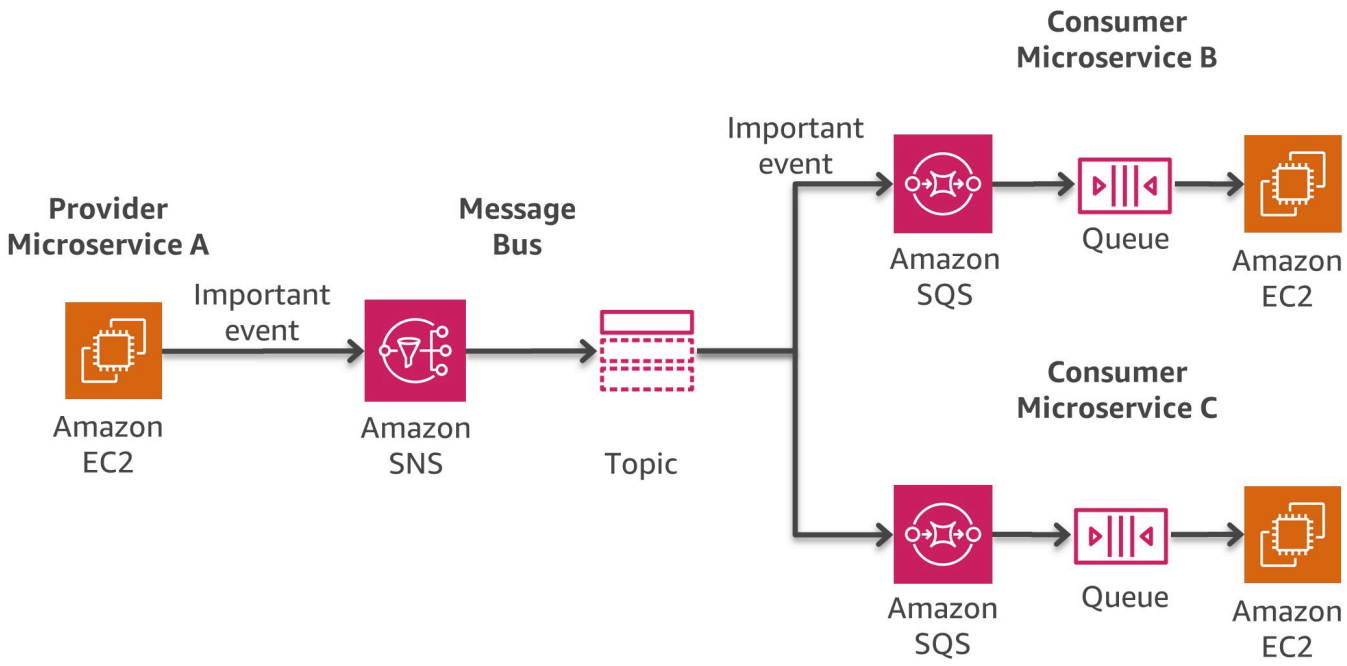
异步消息收发和事件传递

事件传递是用于在微服务之间实现通信的另一种模式。服务通过队列交换消息进行通信。这种通信方式的一个主要优势是，不需要服务发现，而且服务是松散耦合的。

同步系统是紧密耦合的，这意味着同步下游依赖关系中的问题会直接影响上游调用者。上游调用者的重试操作可以快速扩散和放大问题。

根据协议等特定要求，AWS 提供不同的服务来帮助实施此模式。一种可能的实施方法是结合使用 Amazon Simple Queue Service (Amazon SQS) 队列或 Amazon Simple Notification Service (Amazon SNS)。

这两种服务紧密协作：Amazon SNS 使应用程序能够通过推送机制向多个订阅者发送消息。通过结合使用 Amazon SNS 和 Amazon SQS，一条消息可以传递给多个使用者。下图演示了 Amazon SNS 和 Amazon SQS 的集成。



AWS 上的消息总线模式

当您针对 SNS 主题订阅 Amazon SQS 队列时，您可以将消息发布到该主题，同时 Amazon SNS 将消息发送到订阅的 Amazon SQS 队列。该消息包含发布到该主题的标题和消息，以及 JSON 格式的元数据信息。

Amazon EventBridge 是构建事件驱动型架构的另一个选项，该架构具有大规模跨越内部应用程序、第三方 SaaS 应用程序和 AWS 服务的事件源。EventBridge 是一项完全托管式事件总线服务，它接收来自不同来源的事件，根据路由规则识别目标，然后向该目标（包括 AWS Lambda、Amazon SNS 和 Amazon Kinesis Streams 等）提供近乎实时的数据。入站事件也可以在传送之前通过输入转换器进行自定义。

为了显著加快开发事件驱动型应用程序的速度，EventBridge 架构注册表收集和组织架构，包括 AWS 服务生成的所有事件的架构。客户还可以定义自定义架构或使用推断架构选项来自动发现架构。然而，总的来说，所有这些功能的一个潜在权衡是 EventBridge 传输的延迟值相对较高。此外，EventBridge 的原定设置吞吐量和配额可能需要根据使用场景，通过支持请求来要求增加。

另一种实施策略基于 Amazon MQ，可在现有软件使用开放标准 API 和协议（包括 JMS、NMS、AMQP、STOMP、MQTT 和 WebSocket）进行消息收发时使用。Amazon SQS 公开了一个自定义 API，这意味着，如果您想要迁移某个现有应用程序（例如，将其从本地部署环境迁移到 AWS），则需要更改代码。使用 Amazon MQ，在许多情况下都不需要这样做。

Amazon MQ 负责管理和维护 ActiveMQ，这是一个常见的开源消息代理。自动预置底层基础设施，以实现高可用性和消息持久性，从而支持应用程序的可靠性。

编排和状态管理

微服务具有分布式特性，因此很难编排涉及多个微服务的工作流。开发人员可能想要直接向服务中添加编排代码。应避免这样做，因为这会引入更紧密的耦合，使得快速替换单个服务变得更加困难。

您可以使用 AWS Step Functions，通过分别执行离散函数的各个组件构建应用程序。Step Functions 提供状态机，此状态机隐藏了服务编排的复杂性，例如，错误处理、序列化和并行化。这使您可以快速扩展和更改应用程序，同时避免在服务中添加额外的协调代码。

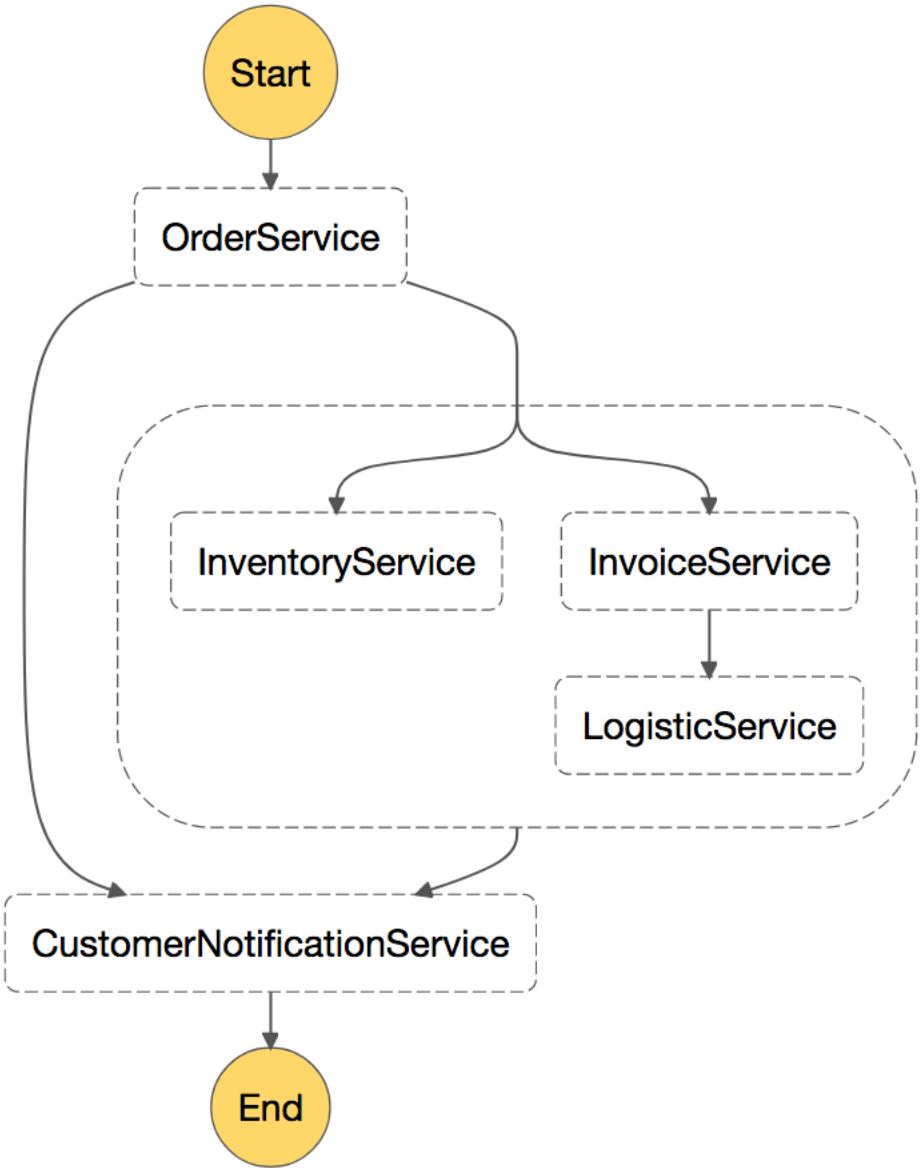
Step Functions 是协调组件和单步调试应用程序函数的可靠方法。Step Functions 提供一个图形控制台，可按照一系列步骤排列应用程序的组件，并以可视化方式呈现这些组件。这样可以更轻松地构建和运行分布式服务。

Step Functions 可以自动开始和跟踪各个步骤，并在出现错误时重试，以便您的应用程序按照预期顺序执行。Step Functions 可记录每个步骤的状态，因此在出现错误时，您能够迅速诊断并调试问题。您甚至无需编写代码就可以更改和添加步骤，从而使您的应用程序更快地发展并创新。

Step Functions 是 AWS 无服务器平台的一部分，支持编排 Lambda 函数、基于计算资源（如 Amazon EC2、Amazon EKS、Amazon ECS）的应用程序，以及 Amazon SageMaker 和 AWS Glue 等其他服务。Step Functions 为您管理操作和底层基础设施，从而帮助确保您的应用程序在任何规模下均可用。

Step Functions 使用 Amazon States Language 构建工作流。工作流可以包含连续或平行步骤以及分支步骤。

下图展示了一个包含顺序和并行步骤的微服务架构工作流的示例。调用此类工作流可以通过 Step Functions API 或 API Gateway 完成。



Step Functions 调用的微服务工作流的示例

分布式监控

微服务架构由许多必须受监控的不同分布式部件组成。您可以使用 Amazon CloudWatch 来收集和跟踪各项指标、集中和监控日志文件、设置告警以及自动应对 AWS 环境中的更改。CloudWatch 可以监控各种 AWS 资源，例如 Amazon EC2 实例、DynamoDB 表、Amazon RDS 数据库实例、应用程序和服务生成的自定义指标以及应用程序生成的所有日志文件。

监控

您可以使用 CloudWatch 全面地了解资源使用率、应用程序性能和运行状况。CloudWatch 提供可靠、可扩展且灵活的监控解决方案，您可以在几分钟内开始使用。您不再需要设置、管理和扩展自己的监控系统和基础设施。在微服务架构中，使用 CloudWatch 监控自定义指标的功能是一项额外的优势，因为开发人员可以决定应该为每个服务收集哪些指标。此外，您还可以根据自定义指标实施动态扩缩。

除了 Amazon CloudWatch 之外，还可以使用 CloudWatch Container Insights 来从容器化的应用程序和微服务中收集、聚合和汇总指标和日志。CloudWatch Container Insights 会自动收集许多资源（如 CPU、内存、磁盘和网络）的指标，并在集群、节点、一组容器 (pod)、任务和服务级别聚合为 CloudWatch 指标。使用 CloudWatch Container Insights，您可以访问 CloudWatch Container Insights 控制面板指标。它还提供诊断信息（如容器重新启动失败），以帮助您查明问题并快速解决问题。还可以对 Container Insights 收集的指标设置 CloudWatch 告警。

Container Insights 适用于 Amazon ECS、Amazon EKS 以及 Amazon EC2 上的 Kubernetes 平台。Amazon ECS 支持包括对 Fargate 的支持。

另一个常见选项（尤其是对于 Amazon EKS）是使用 Prometheus。Prometheus 是一个开源的监控和提醒工具包，经常与 Grafana 结合使用，以可视化收集到的指标。许多 Kubernetes 组件将指标存储在 `//metrics` 下，且 Prometheus 会定期擦除这些指标。

Amazon Managed Service for Prometheus (AMP) 是一项与 Prometheus 兼容的监控服务，使您能够大规模监控容器化应用程序。通过 AMP，您可以使用开源 Prometheus 查询语言 (PromQL) 来监控容器化工作负载的性能，而不必管理相应的底层基础设施（这是管理运营指标的提取、存储和查询所需的）。您可以使用 AWS Distro for OpenTelemetry 或 Prometheus 服务器作为收集代理，从 Amazon EKS 和 Amazon ECS 环境收集 Prometheus 指标。

AMP 通常与 Amazon Managed Service for Grafana (AMG) 结合使用。无论指标存储在何处，通过 AMG 都可以轻松查询、可视化、提示和理解指标。借助 AMG，您可以分析指标、日志和跟踪数据，而无需预置服务器、配置和更新软件，或者在生产中完成保护和扩展 Grafana 所涉及的繁重工作。

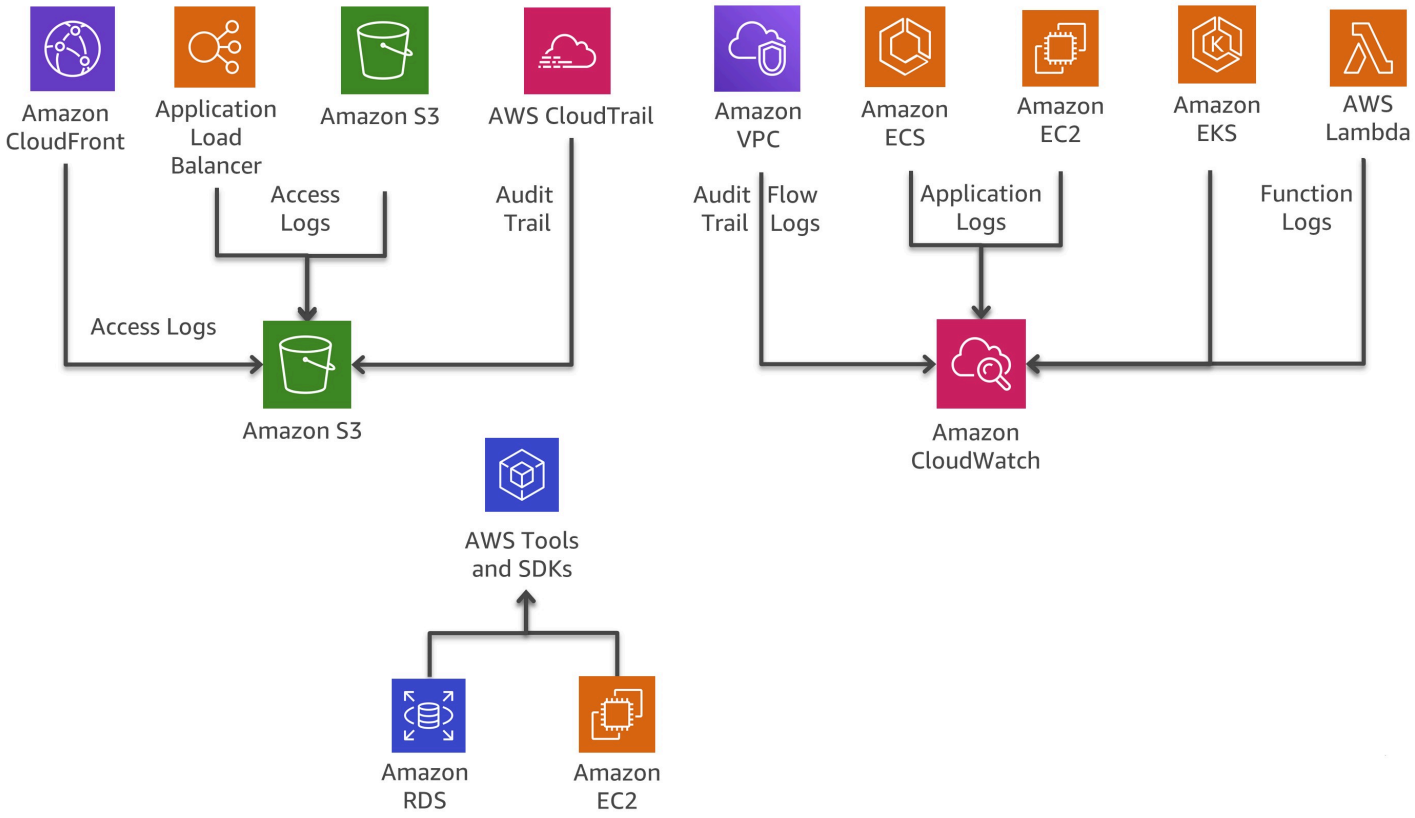
集中日志

一致的日志记录对于故障排除和问题识别至关重要。微服务使团队能够发布比以往更多的版本，并鼓励工程团队在生产中对新功能进行试验。了解客户影响对于逐步改进应用程序至关重要。

原定设置情况下，大多数 AWS 服务都会集中其日志文件。AWS上日志文件的主要目标是 Amazon S3 和 Amazon CloudWatch Logs。对于在 Amazon EC2 实例上运行的应用程序，可以使用守护进程将日志文件发送到 CloudWatch Logs。Lambda 函数将其日志输出以原生方式发送到 CloudWatch Logs，

且 Amazon ECS 随附对可以将容器日志集中到 CloudWatch Logs 的 awslogs 日志驱动程序的支持。对于 Amazon EKS，Fluent Bit 或 Fluentd 可以将集群中各个实例的日志转发到一个集中的日志记录 CloudWatch Logs 中，以便使用 Amazon OpenSearch Service 和 Kibana 将这些日志组合在一起，从而提供更高级别的报告。由于 Fluent Bit 更小且具有性能优势，建议使用 Fluent Bit 而非 FluentD。

下图说明了其中某些服务的日志记录功能。然后，团队可以使用 Amazon OpenSearch Service 和 Kibana 等工具搜索和分析这些日志。Amazon Athena 可用于针对 Amazon S3 中集中的日志文件运行一次性查询。



AWS 服务的日志记录功能

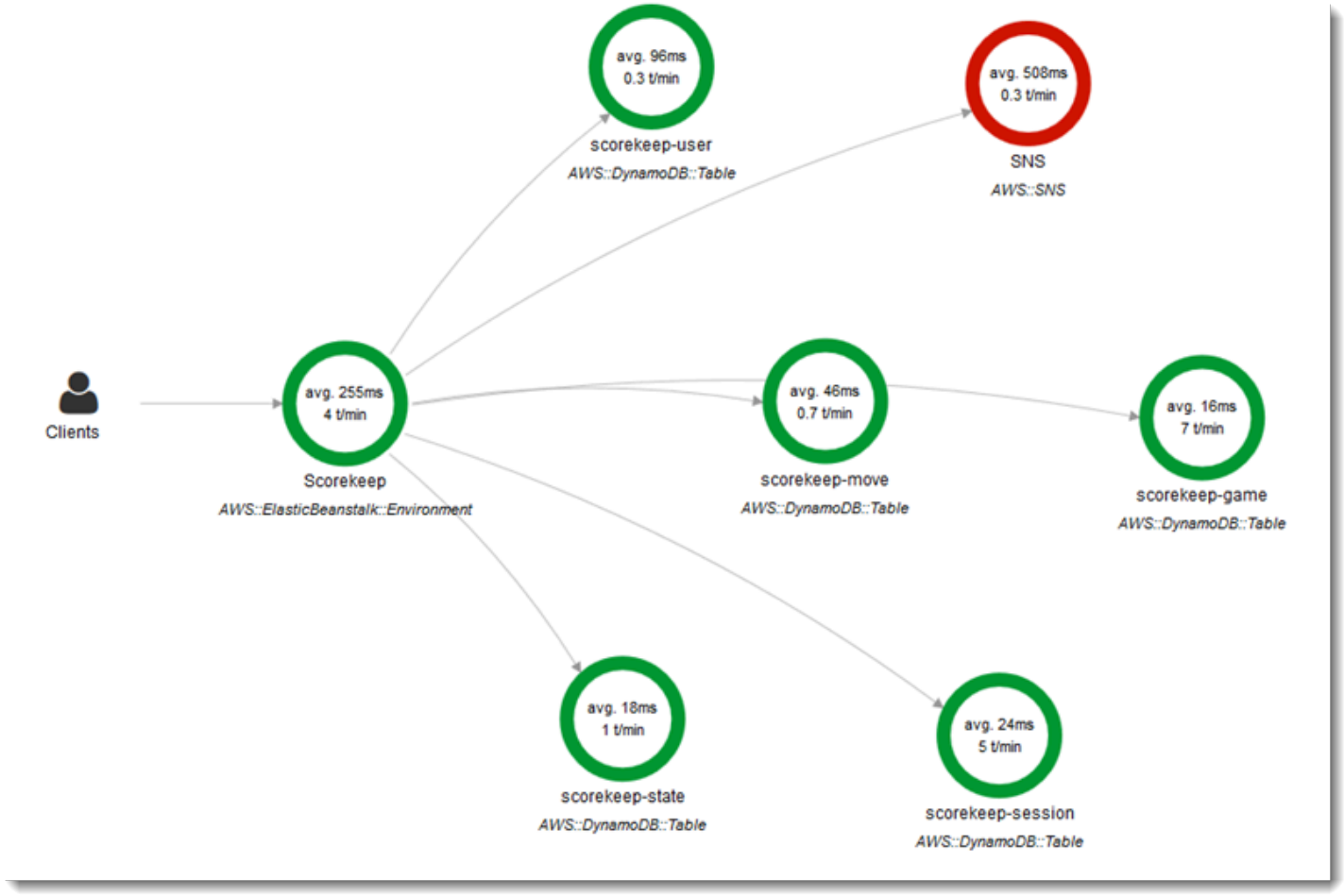
分布式跟踪

在许多情况下，一组微服务会共同处理请求。设想一个由数十个微服务组成的复杂系统，其中调用链中的一个服务中发生了错误。即使每个微服务都正确记录并且日志整合在一个中央系统中，也很难找到所有相关的日志消息。

AWS X-Ray 的中心思想是使用相关 ID，这是附加到与特定事件链相关的所有请求和消息的唯一标识符。当请求到达第一个集成 X-Ray 的服务（例如，Application Load Balancer 或 API Gateway）并且

包含在响应中时，跟踪 ID 将添加到 HTTP 请求中名为 `X-Amzn-Trace-Id` 的特定跟踪标头中。通过 X-Ray 开发工具包，所有微服务都可以读取此标头，但也可以添加或更新此标头。

X-Ray 可与 Amazon EC2、Amazon ECS、Lambda 和 AWS Elastic Beanstalk 结合使用。您可以将 X-Ray 与在这些服务上部署的采用 Java、Node.js 和 .NET 编写的应用程序结合使用。



AWS X-Ray 服务地图

Epsagon 是完全托管式 SaaS，包括对所有 AWS 服务、第三方 API（通过 HTTP 调用）以及其他常见服务（例如 Redis、Kafka 和 Elastic）的跟踪。Epsagon 服务包括监控功能、对最常见服务发出提示，并且通过此项服务可以了解代码所进行的每次调用的有效负载。

AWS Distro for OpenTelemetry 是 AWS 支持的 OpenTelemetry 项目发行版，安全且可直接用于生产。AWS Distro for OpenTelemetry 属于云原生计算基金会，提供开源 API、库和代理来收集应用程序监控的分布式跟踪数据和指标。使用 AWS Distro for OpenTelemetry，只需检测一次应用程序，即可将相关的指标和跟踪数据发送至多个 AWS 和合作伙伴监控解决方案。使用自动检测代理收集跟踪数据，无需更改您的代码。AWS Distro for OpenTelemetry 还可以从您的 AWS 资源和托管式服务中收

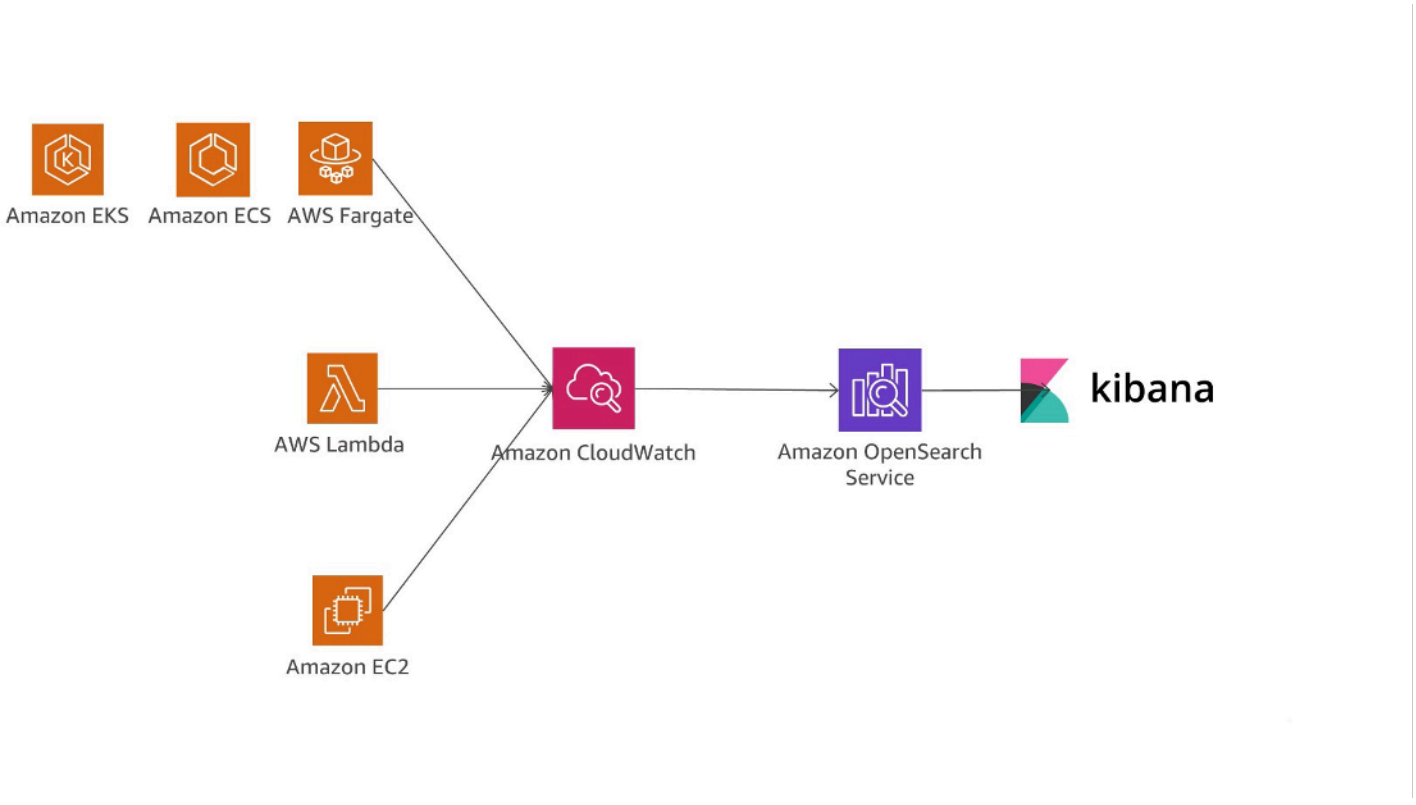
集元数据，以便将应用程序性能数据与底层基础设施数据关联，从而减少解决问题的平均时间。使用 AWS Distro for OpenTelemetry 可以对在 Amazon EC2、Amazon ECS、Amazon EC2 上的 Amazon EKS、Fargate 和 AWS Lambda 以及本地部署的设施上运行的应用程序进行检测。

AWS 上的日志分析选项

搜索、分析和可视化日志数据是了解分布式系统的重要方面。Amazon CloudWatch Logs Insights 使您能够即时探索、分析和可视化日志。该服务允许您排除操作问题。分析日志文件的另一个选项是将 Amazon OpenSearch Service 与 Kibana 结合使用。

Amazon OpenSearch Service 可用于全文搜索、结构化搜索、分析以及这三者的组合。Kibana 是一款开源数据可视化插件，可与 Amazon OpenSearch Service 无缝集成。

下图演示了使用 Amazon OpenSearch Service 和 Kibana 进行的日志分析。CloudWatch Logs 可以配置为通过 CloudWatch Logs 订阅近乎实时地将日志条目流式传输到 Amazon OpenSearch Service。Kibana 将数据可视化，并为 Amazon OpenSearch Service 中的数据存储公开便捷的搜索接口。此解决方案可以与诸如 ElastAlert 等软件结合使用以实施提醒系统，从而在检测到数据中的异常值、峰值或其他感兴趣的模式时，发送 SNS 通知和电子邮件以及创建 JIRA 票证等。



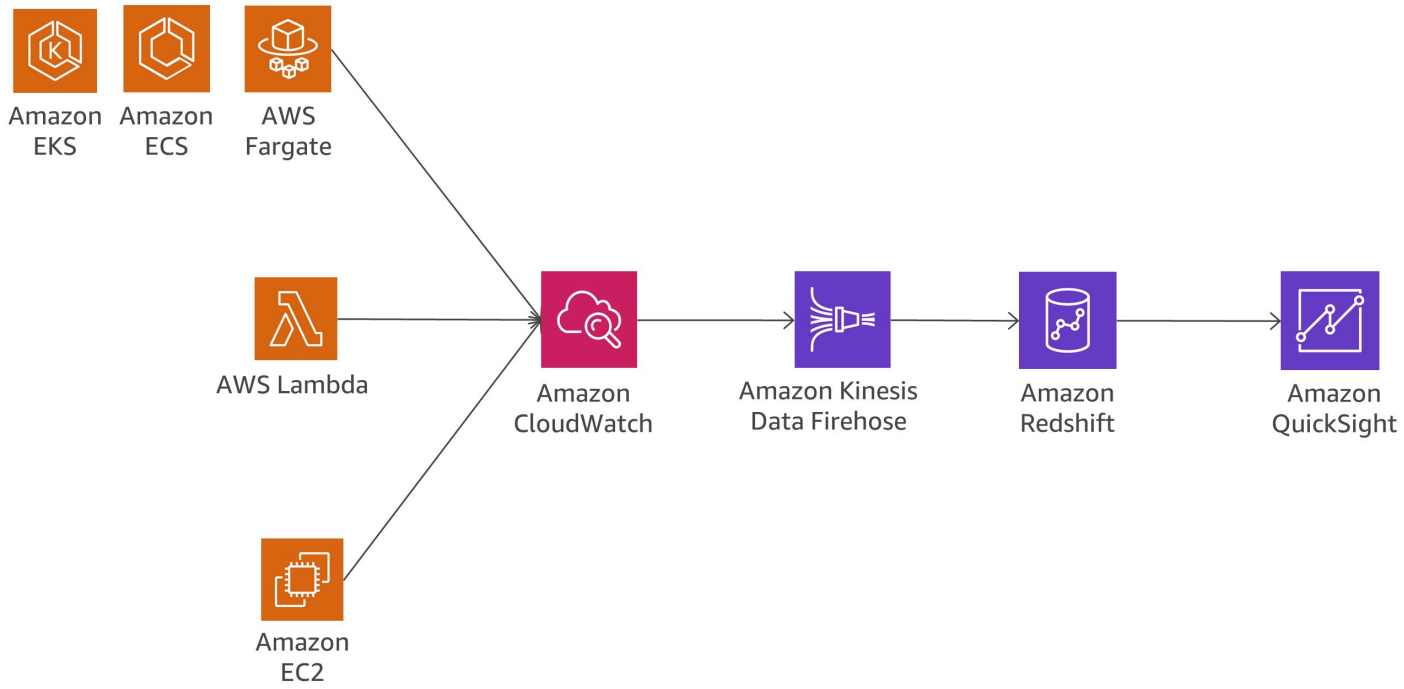
使用 Amazon OpenSearch Service 和 Kibana 进行日志分析

分析日志文件的另一个选项是将 Amazon Redshift 与 Amazon QuickSight 结合使用。

QuickSight 可以轻松连接到 AWS 数据服务，其中包括 Amazon Redshift、Amazon RDS、Amazon Aurora、Amazon EMR、DynamoDB、Amazon S3 和 Amazon Kinesis。

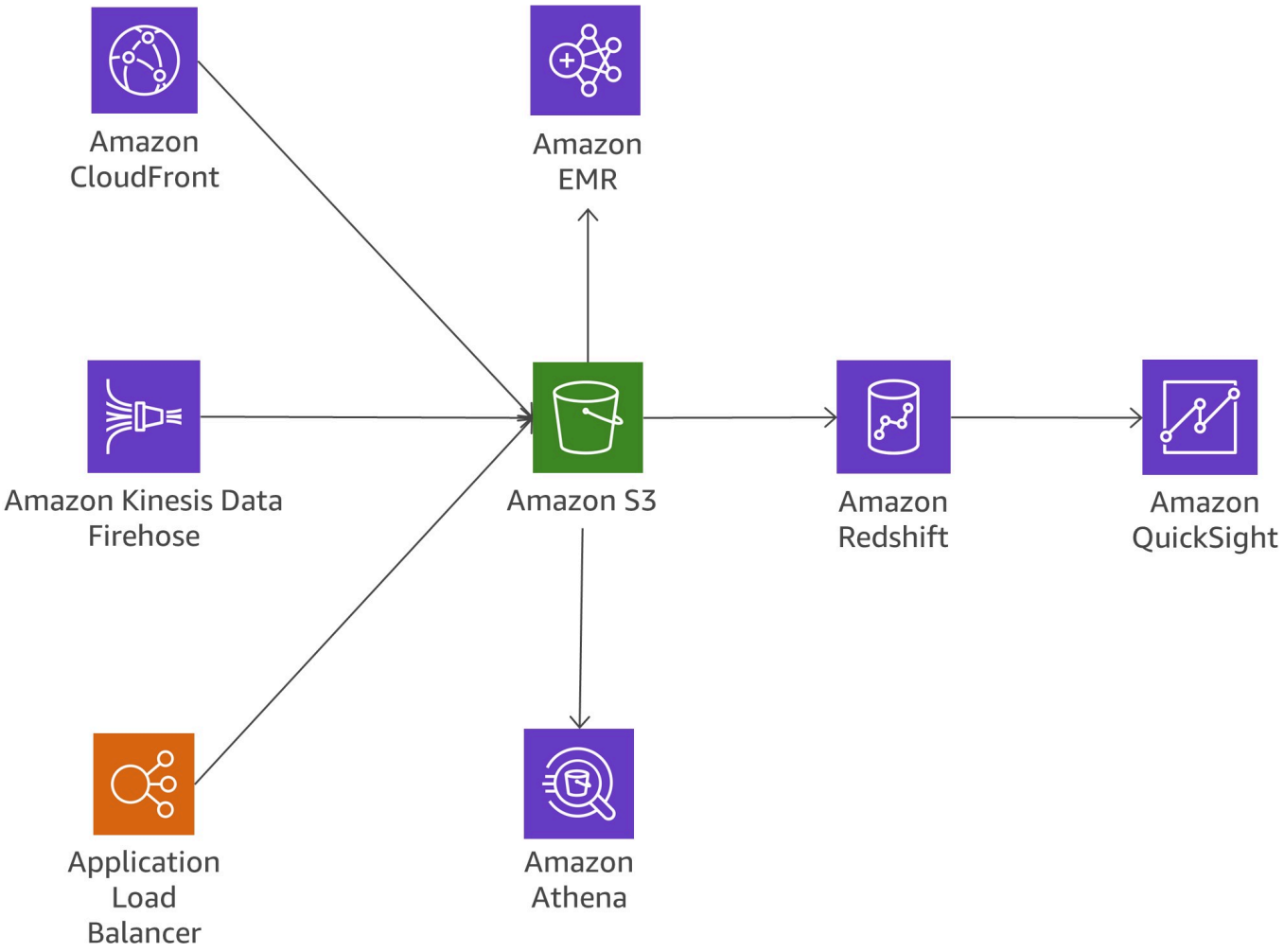
CloudWatch Logs 可以充当日志数据的集中存储，不仅能存储数据，还可以将日志条目流式传输到 Amazon Kinesis Data Firehose。

下图描述了使用 CloudWatch Logs 和 Kinesis Data Firehose 将日志条目从不同来源流式传输到 Amazon Redshift 的场景。Amazon QuickSight 使用 Amazon Redshift 中存储的数据进行分析、报告和可视化。



使用 Amazon Redshift 和 Amazon QuickSight 进行日志分析

下图描述了在 Amazon S3 上进行日志分析的场景。当日志存储在 Amazon S3 存储桶中时，日志数据可以加载到不同的 AWS 数据服务（例如 Amazon Redshift 或 Amazon EMR）中，以分析存储在日志流中的数据并查找异常。



Amazon S3 上的日志分析

干扰

通过将整体式应用程序分解为小型微服务，通信开销增加，因为微服务必须相互通信。在许多实施情况下，之所以使用 HTTP 上的 REST，是因为它是轻量级的通信协议，但消息量大可能会导致问题。在某些情况下，您可能需要考虑整合来回发送许多消息的服务。如果您发现自己整合越来越多的服务只是为了减少干扰，则您应检查问题域和域模型。

协议

在本白皮书前面的章节the section called “异步通信和轻量级消息收发”中，讨论了各种可能的协议。对于微服务，使用像 HTTP 这样的简单协议是很常见的。由服务交换的消息可以通过不同的方式编码，例如，以人类可读取的格式（如 JSON 或 YAML）或以有效的二进制格式（如 Avro 或协议缓冲区）。

缓存

缓存是减少微服务架构的延迟和干扰的好方法。可以使用多个缓存层，具体取决于实际的使用案例和瓶颈。在 AWS 上运行的许多微服务应用程序使用 ElastiCache 通过在本地缓存结果来减少对其他微服务的调用量。API Gateway 提供内置缓存层以减少后端服务器上的负载。此外，缓存还有助于减少数据持久层的负载。所有缓存机制面临的挑战是，在良好的缓存命中率和数据的及时性和一致性之间找到适当的平衡点。

审计

在可能具有数百个分布式服务的微服务架构中要解决的另一个挑战是，确保用户操作在所有服务上的可见性，并且能够在组织级别跨所有服务获得良好的整体视图。为了帮助实施安全策略，请务必审计资源访问和导致系统更改的活动。

更改必须在各个服务级别以及跨在更广泛的系统上运行的服务进行跟踪。通常，更改在微服务架构中频繁发生，这使得审计更改更加重要。本节将介绍 AWS 中可帮助审计微服务架构的关键服务和功能。

审计跟踪记录

AWS CloudTrail 是一种用于跟踪微服务中更改的有用工具，因为它支持记录在 AWS Cloud上进行的所有 API 调用，并几乎实时地发送到 CloudWatch Logs 或者在几分钟内发送到 Amazon S3。

所有用户和自动系统操作都变得可搜索，并可以针对意外行为、公司策略违规或调试进行分析。记录的信息包括时间戳、用户和账户信息、调用的服务、请求的服务操作、调用者的 IP 地址，以及请求参数和响应元素。

CloudTrail 允许为同一账户定义多个跟踪，这使不同的利益攸关方（如安全管理员、软件开发人员或 IT 审计人员）能够创建和管理自己的跟踪记录。如果微服务团队拥有不同的 AWS 账户，则可以将跟踪记录聚合到单个 S3 存储桶中。

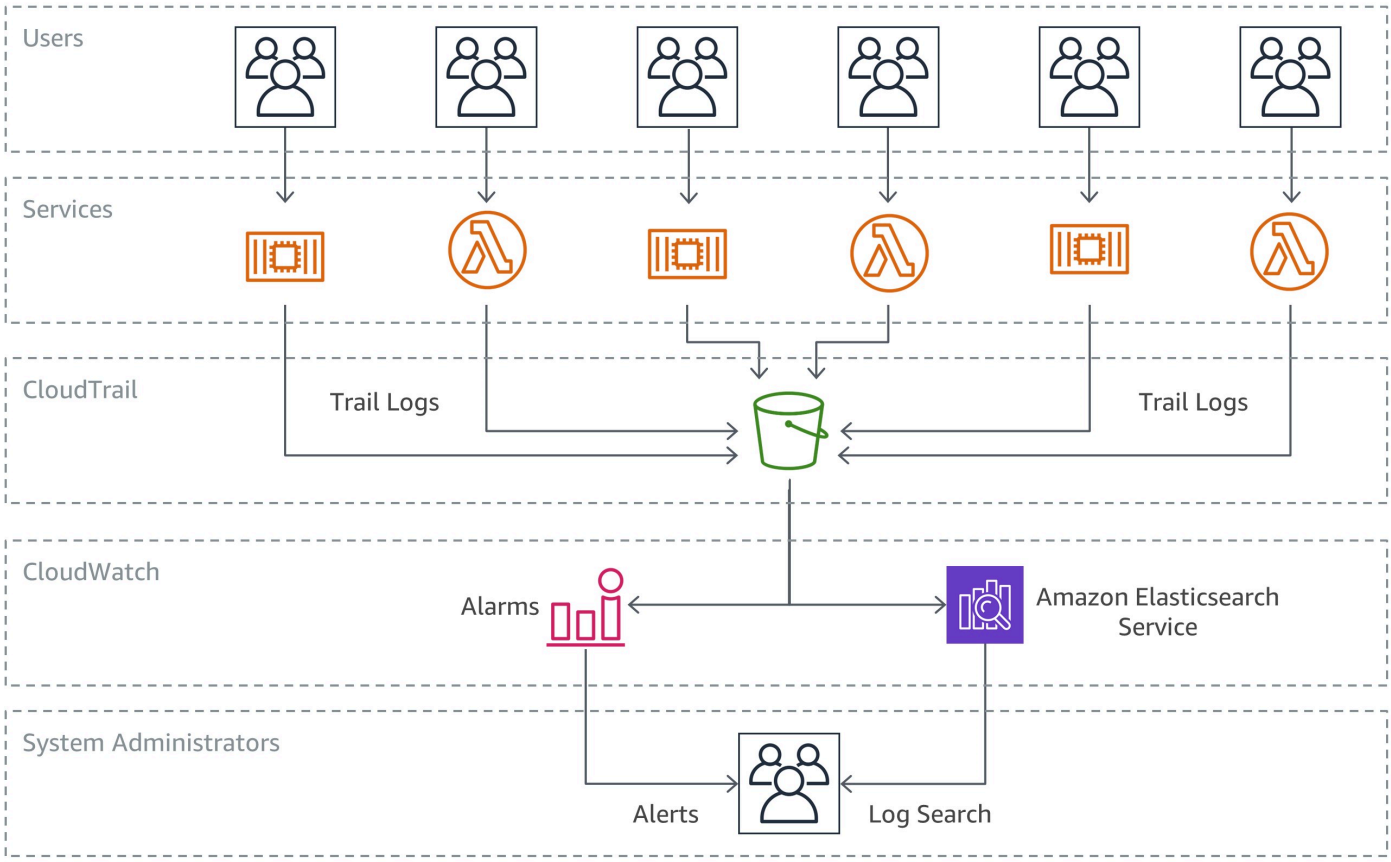
在 CloudWatch 中存储审计跟踪记录的优势是，实时捕获审计跟踪记录数据，并轻松地将信息重新路由到 Amazon OpenSearch Service 以进行搜索和可视化。您可以将 CloudTrail 配置为同时登录 Amazon S3 和 CloudWatch Logs。

事件和实时操作

必须快速响应系统架构中的某些更改，并且必须采取行动来补救这种情况，或者启动特定的监管程序来授权更改。Amazon CloudWatch Events 与 CloudTrail 的集成允许它为所有 AWS 服务中的所有变异 API 调用生成事件。此外，也可以根据固定计划定义自定义事件或生成事件。

当触发事件并且符合定义的规则时，可以立即通知组织中预定义的一组人员，以便他们可以采取适当的措施。如果可以自动执行所需的操作，规则可以自动触发内置工作流或调用 Lambda 函数来解决问题。

下图介绍了一种环境，其中 CloudTrail 和 CloudWatch Events 协同工作，以满足微服务架构中的审计和补救要求。CloudTrail 正在跟踪所有微服务，审计跟踪记录存储在 Amazon S3 存储桶中。CloudWatch Events 会在发生操作更改时感知到这些更改。CloudWatch Events 将响应这些操作更改并在必要时采取纠正措施，方式是发送消息以响应环境、激活函数、进行更改并捕获状态信息。CloudWatch Events 位于 CloudTrail 之上，并在对您的架构进行特定更改时触发提醒。



审计和补救

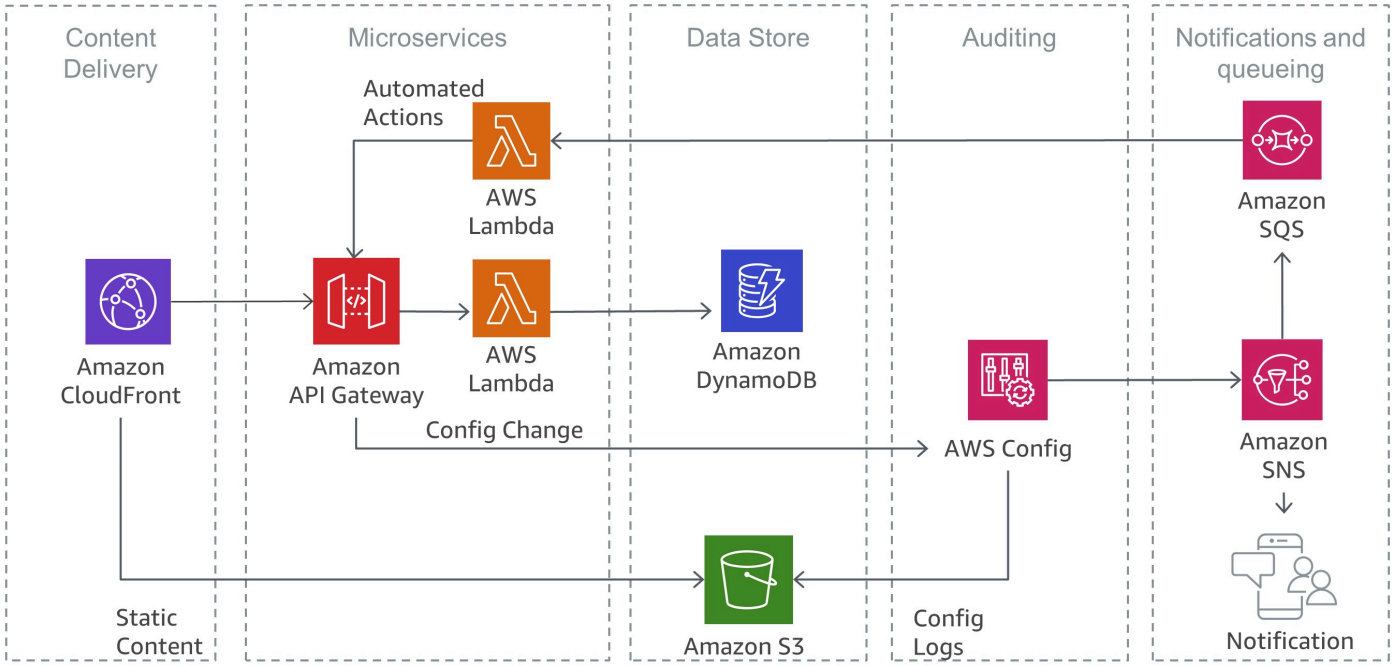
资源清单和更改管理

要在敏捷开发环境中维持对快速变化的基础设施配置的控制，更加自动化、托管式架构审计和控制方法很有用。

尽管 CloudTrail 和 CloudWatch Events 是跟踪和响应跨微服务的基础设施更改的重要构建块，但 AWS Config 规则允许公司使用特定规则定义安全策略，并自动检测、跟踪这些策略的违规行为并向您发送提醒。

下一个示例演示了如何在微服务架构中检测、通知和自动响应不合规的配置更改。开发团队的一名成员对微服务的 API Gateway 进行了更改，以允许终端节点接受入站 HTTP 流量，而不是只允许 HTTPS 请求。

因为这种情况之前已被组织确定为安全合规问题，所以 AWS Config 规则已在监控这种情况。该规则将更改标识为安全违规行为，并执行两项操作：在 Amazon S3 存储桶中创建检测到的更改日志以进行审计，并创建一个 SNS 通知。在本场景中，Amazon SNS 有两种用途：将电子邮件发送到指定的组以通知他们存在安全违规行为，以及将消息添加到 SQS 队列。然后，获取消息，并通过更改 API Gateway 配置来还原合规状态。



使用 AWS Config 检测安全违规行为

总结

微服务架构是一种分布式设计方法，旨在克服传统整体式架构的局限性。微服务有助于扩展应用程序和扩大组织规模，同时缩短周期时间。但是，它们也带来了一些挑战，可能会额外增加架构复杂性和操作负担。

AWS 提供了大量托管式服务组合，可帮助产品团队构建微服务架构，并最大限度地降低架构和操作复杂性。本白皮书可指导您了解相关的 AWS 服务，以及如何使用 AWS 服务以原生方式实施典型的模式，如服务发现或事件溯源。

资源

- AWS 架构中心
- AWS 白皮书
- AWS 架构月刊
- AWS 架构博客
- “这是我的架构”视频
- AWS Answers
- AWS 文档

文档历史记录和贡献者

文档历史记录

要获得有关此白皮书的更新通知，请订阅 RSS 源。

更新-历史记录-更改	更新-历史记录-描述	更新-历史记录-日期
已更新白皮书	集成 Amazon EventBridge、AWS OpenTelemetry、AMP、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	细微的文本更改。	2019 年 8 月 1 日
已更新白皮书	集成 Amazon EKS、AWS Fargate、Amazon MQ、AWS PrivateLink、AWS App Mesh 和 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	发布了“在 AWS 上实施微服务”。	2016 年 12 月 1 日

> **Note**
>
> 要订阅 RSS 更新，您必须为正在使用的浏览器启用 RSS 插件。

贡献者

以下是对本文做出贡献的个人和组织：

- Sascha Möllering，AWS 解决方案架构师
- Christian Müller，AWS 解决方案架构师
- Matthias Jung，AWS 解决方案架构师
- Peter Dalbhanjan，AWS 解决方案架构师
- Peter Chapman，AWS 解决方案架构师
- Christoph Kassen，AWS 解决方案架构师
- Umair Ishaq，AWS 解决方案架构师
- Rajiv Kumar，AWS 解决方案架构师

声明

客户负责对本文档中的信息进行独立评估判断。本文档：(a) 仅供参考；(b) 代表当前提供的 AWS 产品和实践，如有更改，恕不另行通知；并且 (c) AWS 及其附属机构、供应商或许可方不做任何承诺或保证。AWS 产品或服务“按原样”提供，不提供任何形式的保证、陈述或条件，无论是明示还是暗示。AWS 对其客户的责任和义务由 AWS 协议决定，本文档与 AWS 和客户之间签订的任何协议无关，亦不影响任何此类协议。