AWS 无服务器多层架构

使用 Amazon API Gateway 和 AWS Lambda

2015 年 11 月

版权声明

目录

摘要

本白皮书向您介绍 Amazon Web Services (AWS) 的创新可对您的微服务、移动后端、公共网站等常见模式多层架构设计产生何种影响。架构师和开发人员现在可以使用包含 Amazon API Gateway 和 AWS Lambda 的实现模式，以减少创建和运作管理多层应用程序所需的开发和运营周期。

简介

数十年来，多层应用程序（三层、n 层等）一直是一种有着基石地位的架构模式。多层模式提供了良好的操作指南，可确保解耦的、可扩展的应用程序组件能够独立管理和维护（通常由不同的团队执行）。多层应用程序通常使用面向服务型架构 (SOA) 来构建，通过这种方式使用 Web 服务。在这种方式中，网络充当层与层之间的边界。但是，创建新的 Web 服务层作为应用程序的一部分也有许多无差异化的方面。这种模式本身的一个直接结果就是，多层 Web 应用程序中需要编写大量代码。例如，将一个层集成到另一个层的代码、定义用于实现层间沟通的 API 和数据模型的代码，以及确保不会意外暴露层集成点的安全相关代码。

Amazon API Gateway[1] 是用于创建和管理 API 的服务，AWS Lambda[2] 是用于运行任意代码功能的服务，这两者可以搭配使用，以简化可靠的多层应用程序的创建过程。

Amazon API Gateway 与 AWS Lambda 集成之后，用户定义的代码功能可以通过用户定义的 HTTPS 请求直接触发。无论所需的请求量如何，API Gateway 和 Lambda 都可以自动扩展，以支持您的应用程序的确切需求。如果组合使用两者，您可以为自己的应用程序创建这样一个层：只需编写关乎应用程序的代码，而*不必考虑*实现多层架构的各种其他无差异化的方面，例如构建高可用性、编写客户端软件开发工具包、服务器/操作系统 (OS) 管理、扩展、实现客户端身份验证机制等。如果组合使用两者，您可以为自己的应用程序创建这样一个层：只需编写关乎应用程序的代码，而不必考虑实现多层架构的各种其他无差异化的方面，例如构建高可用性、编写客户端软件开发工具包、服务器/操作系统 (OS) 管理、扩展、实现客户端身份验证机制等。

最近，AWS 宣布推出创建 Lambda 函数的功能，所创建的函数可以在您的 Amazon Virtual Private Cloud (Amazon VPC)[3] 中执行。该功能拓展了组合 API Gateway 和 Lambda 的优势，覆盖了需要保护网络隐私的各种使用案例。例如，当您需要将自己的 Web 服务与包含敏感信息的关系数据库集成时。Lambda 与 Amazon VPC 的集成间接扩展了 Amazon API Gateway 的功能，因为后端作为 Amazon VPC 一部分保持私有和安全性，而这样的集成使开发人员能够定义后端之前通过 Internet 访问的自有 HTTPS API 集。在多层架构的各个层，您都能看到这一强大模式带来的优势。本白皮书重点介绍最常见的多层架构示例，即**三层** Web 应用程序。不过，您完全也可以在典型的三层 Web 应用程序以外的很多领域应用这种多层模式。

三层架构概述

对于面向用户的应用程序来说，三层架构是一种常用模式。组成这种架构的三个层分别为**表示层**、**逻辑层**和**数据层**。表示层代表与用户直接交互的组件（如网页、移动应用程序用户界面等）。逻辑层包含将表示层的用户操作转换为驱动应用程序行为的功能所需的代码。数据层包含存放应用程序相关数据的存储介质（数据库、对象存储、缓存、文件系统等）。图 1 是一个简单三层应用程序的示例。

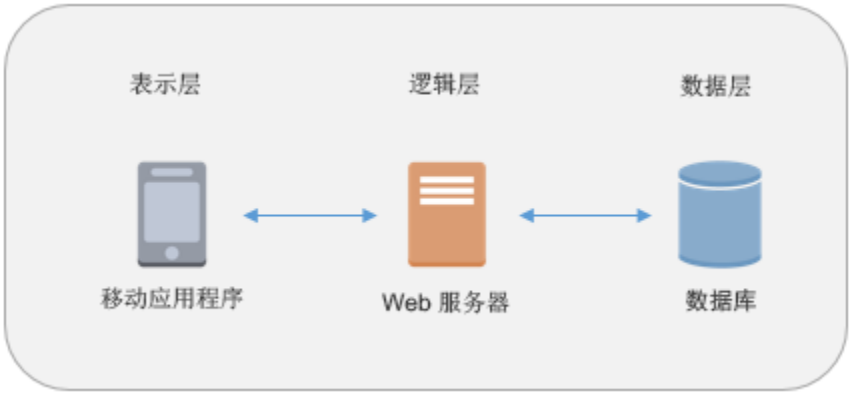

图 1：一个简单三层应用程序的架构模式

如需了解有关*通用* 三层架构模式的更多信息，网上有许多很好的资源可以参考。本白皮书重点介绍这种架构在使用 Amazon API Gateway 和 AWS Lambda 时的特定实现模式。

无服务器的逻辑层

在三层架构中，逻辑层是应用程序的核心。因此，将 Amazon API Gateway 与 AWS Lambda 集成来构成您的逻辑层具有革命性的意义。通过这两项服务的功能，您可以构建高度可用、可扩展且安全的无服务器式生产应用程序。您的应用程序也许需要使用数千服务器，但如果采用这种模式，您不必管理任何服务器。此外，组合使用这些托管服务还有以下好处：

- 不必选择、保护、修补或管理任何操作系统。
- 对服务器的大小、监控和扩展没有任何要求。

- 超额配置不会产生任何成本风险。
- 低额配置不会带来任何性能风险。

此外，每项服务还包含一些能够使多层架构模式受益的特定功能。

Amazon API Gateway

Amazon API Gateway 是一种用于定义、部署和维护 API 的完全托管服务。客户端与使用标准 HTTPS 请求的 API 相集成。因此，它明显非常适合用于面向服务型多层架构。但它有一些特定的功能和特性，足以作为您的逻辑层的强大边界。

与 AWS Lambda 集成

Amazon API Gateway 为您的应用程序提供了一种直接利用 AWS Lambda 创新成果的简单方法（HTTPS 请求）。API Gateway 是一座桥梁，将表示层和您在 AWS Lambda 中编写的函数连接起来。在使用您自己的 API 定义客户端/服务器关系后，客户端的 HTTPS 请求内容被传递到 Lambda 函数执行。这些内容包括请求元数据、请求标头、请求正文。

全球范围内皆可实现稳定的 API 性能

每个 Amazon API Gateway 部署内部都包含 Amazon CloudFront[4]。Amazon CloudFront 是一种内容分发 Web 服务，它借助 Amazon 覆盖全球的边缘站点网络作为（与您的 API 集成的）客户端的连接点。这有助于减少您的 API 的总体响应时间延迟。通过使用分布在世界各地的多个边缘站点，Amazon CloudFront 还能够为您提供抵御分布式拒绝服务 (DDoS) 攻击的能力。有关更多信息，请参阅抵御 DDoS 攻击的 AWS 最佳实践[5]白皮书。

您可以通过 Amazon API Gateway 将响应存储在可选的内存高速缓存中，以提升特定 API 请求的性能。这不仅可为重复的 API 请求提供性能优势，还能减少后端的执行次数，从而降低您的总体成本。

鼓励创新

构建任何新应用程序所需的开发工作都是一种不小的投资。在项目开始前，您需要进行论证。通过减少开发任务所需的投资金额和时间，您可以更自由地实验和创新。

对于许多基于多层 Web 服务的应用程序来说，表示层都很容易在用户层面变得“支离破碎”（各种各样的移动设备、Web 浏览器等）。而且，用户往往不会固定在某一个地域。但解耦的逻辑层不会因为用户而实际碎片化。所有用户都依赖于运行您的逻辑层的同一个基础设施，这凸显了基础设施的重要性。在初次实现逻辑层时，为了更快地交付新的应用程序，您常常会听到请求“偷工减料”的提议，如“最初发布时，我们不需要检测指标”、“早期用户不会很多，我们以后再考虑扩展问题”等。应用程序一旦在生产环境中投入运行，之后再部署此类更改，很可能会导致技术债务和运营风险。Amazon API Gateway 可让*您* 避开这些问题并加快交付速度，这是因为该服务已经预先实施所有更改。

应用程序的整个生命周期有多长，这可能是个未知数，但也可能已经知道生命周期较为短暂。出于这些原因，为新的多层应用程序创建业务案例可能比较困难。如果您的起点已包含 Amazon API Gateway 提供的托管功能，并且在您的 API 开始接收请求后才开始产生基础设施成本，这会变得容易得多。有关更多信息，请参阅 Amazon API Gateway 定价。[6]

快速迭代，保持敏捷

对于新的应用程序，用户群可能难以清晰定义（规模、使用模式等）。在用户群成长期间，逻辑层必须保持敏捷。您的应用程序和业务应能够转变和适应早期用户的变更期望。Amazon API Gateway 能够减少从 API 产生到部署所需的开发周期数。Amazon API Gateway 提供了创建模拟环境[7]的功能，让您能够直接从 API Gateway 生成 API 响应，从而实现客户端应用程序与完整后端逻辑的并行开发。这一优势不仅适用于 API 的首次部署，也适用于企业决定必须快速改变应用程序（及现有 API）以响应用户需求的情况。API Gateway 和 AWS Lambda 提供了版本控制功能，因此，当您以独立 API/功能版本的形式发布新功能时，现有功能和客户端依赖项可以继续不受干扰地运行。

安全性

如果将公共三层 Web 应用程序的逻辑层实现为 Web 服务，这直接将安全性推到了风口浪尖。应用程序需要确保只有经过授权的客户端才有权访问您的逻辑层（在网络上公开）。Amazon API Gateway 通过多种方法解决安全性问题，确保您的后端安全无虞，让您高枕无忧。对于访问控制，不要依赖于为您的客户端应用程序提供静态 API 密钥字符串；密钥字符串可从客户端提取并用到其他地方。您可以充分利用 Amazon API Gateway 提供的多种不同的逻辑层保护方式：

- 针对您的 API 的所有请求都可通过 HTTPS 形式发送，从而在传输过程中实现加密保护。

- 您的 AWS Lambda 函数能够对访问加以限制，从而在 Amazon API Gateway 中的 API 与 AWS Lambda 中的特定函数之间建立信任关系。除了使用您选择公开的 API 以外，没有任何其他方法能够调用该 Lambda 函数。
- Amazon API Gateway 让您能够生成客户端软件开发工具包，以便与您的 API 集成。当 API 需要身份验证时，该软件开发工具包还能管理请求签名。在客户端上用于执行身份验证操作的 API 凭证被直接传递给您的 AWS Lambda 函数 - 如果需要，您可以编写代码执行进一步的身份验证。
- 对于您创建的每个资源/方法组合（作为 API 的一部分），系统都会为其授予独有的 Amazon 资源名称 (ARN)，您可在 AWS Identity and Access Management (IAM)[8] 策略中引用它们。
  - 这意味着，您的 API 与其他 AWS 所有的 API 一样，会受到“一等公民”的对待。IAM 策略可以细化；它们可以引用使用 Amazon API Gateway 创建的 API 的特定资源/方法。
  - API 访问受到您在应用程序代码上下文以外创建的 IAM 策略的限制。这意味着您不必编写任何代码，就能强制实施此类访问级别。由于没有代码，当然也就不存在错误或漏洞了。
  - 使用 AWS 签名版本 4 (SigV4)[9] 授权的授权客户端和针对 API 访问的 IAM 策略允许使用这些凭证根据需要限制或允许对其他 AWS 服务及资源（例如 Amazon S3 存储桶或 Amazon DynamoDB 表）的访问。

AWS Lambda

AWS Lambda 的核心是：只要是使用所支持的语言（基于 Node、JVM 的语言和 Python（2015 年 11 月加入））编写的代码，允许触发任意代码来响应某个事件。事件可以是 AWS 提供的多种编程触发器（称作**事件源** - 单击此处可参阅当前支持的事件源[10]）之一。AWS Lambda 的许多常见使用案例都是围绕事件驱动型数据处理工作流程设计的，例如：处理存储在 Amazon Simple Storage Service (Amazon S3)[11] 中的文件、流式传输 Amazon Kinesis[12] 中的数据记录等。

当与 Amazon API Gateway 结合使用时，AWS Lambda 函数可存在于典型的 Web 服务上下文中，可由 HTTPS 请求直接触发。Amazon API Gateway 是您的逻辑层的“前门”，但您现在需要执行这些 API 背后的逻辑。这正是 AWS Lambda 大显身手之处。

体现您的业务逻辑

AWS Lambda 让您能够编写称作**处理程序**的代码功能，处理程序在被事件触发时执行。例如，您可以编写一个处理程序，在出现针对您的 API 的 HTTPS 请求等事件时触发。Lambda 让您能够在自己选择的粒度级别创建模块化的处理程序（每个 API 一个，或每个 API 方法一个），这些处理程序可以独立地更新、调用和更改。然后，处理程序可以自由访问它的任何其他依赖项（如您通过代码、库、本地二进制文件、甚至是外部 Web 服务上传的其他函数）。Lambda 让您能够在创建期间将所有必需的依赖项打包在函数定义中。创建函数时，您可以指定部署包中的哪个方法将用作请求处理程序。您可以对多个 Lambda 函数定义自由地复用同一个部署包，每个 Lambda 函数可以在同一个部署包中拥有独有的处理程序。在无服务器的多层架构模式中，您在 Amazon API Gateway 中创建的任何一个 API 都将与执行业务逻辑所需的 Lambda 函数（以及其中的处理程序）相集成。

Amazon VPC 集成

AWS Lambda 是您的逻辑层的核心，也是与数据层直接集成的组件。数据层通常包含敏感的业务或用户信息，必须受到严格的保护。对于您可以从 Lambda 函数集成的 AWS 服务，您可以使用 IAM 策略进行访问控制管理。这些服务包括 Amazon S3、Amazon DynamoDB、Amazon Kinesis、Amazon Simple Queue Service (Amazon SQS)、Amazon Simple Notification Service (Amazon SNS)、其他 AWS Lambda 函数等。不过，您可能会有需要管理自身访问控制权限的组件，如关系数据库。借助此类组件，通过将它们部署到私有网络环境 (Amazon Virtual Private Cloud (Amazon VPC)[13]) 中，可以实现更高的安全性。

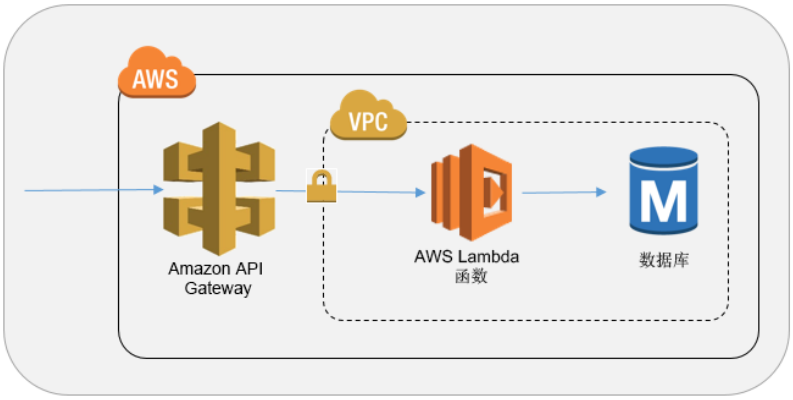

图 2：使用 VPC 的架构模式

使用 VPC 意味着您可以禁止通过 Internet 访问您的业务逻辑依赖的数据库及其他存储介质。此外，VPC 还可确保通过 Internet 与您的数据进行交互的*唯*一方式是：经由您定义的 API 和您编写的 Lambda 代码函数。

安全性

Lambda 函数必须由通过 IAM 策略允许的事件或服务触发才能执行。您可以创建只能通过您定义的 API Gateway 请求调用，否则*根本*无法执行的 Lambda 函数。您的代码将只能在您的有效使用案例中进行处理；由您创建的 API 定义。

每个 Lambda 函数本身需要担任某个 IAM 角色，一种必须通过 IAM 信任关系授予的功能。IAM 角色定义可与您的 Lambda 函数交互的其他 AWS 服务/资源，如 Amazon DynamoDB 表或 Amazon S3 存储桶。您的函数有权访问的服务将在函数以外的位置定义和管理。这很精妙，但极其强大。这样，您编写的代码能够自由存储或检索 AWS 凭证：也就是说，您不必对 API 密钥进行硬编码，也不必编写代码来检索它们或将它们存储在内存中。让您的 Lambda 函数能够调用允许其调用的服务（通过其 IAM 角色定义），这是由服务本身替您管理的。

数据层

使用 AWS Lambda 作为逻辑层时，您的数据层有大量数据存储选项。这些选项可分为两大类：Amazon VPC 托管数据存储和支持 IAM 的数据存储。AWS Lambda 可以与这两者安全地集成。

Amazon VPC 托管数据存储

AWS Lambda 与 Amazon VPC 的集成使得函数能够以私有且安全的方式与各类数据存储技术集成。

- Amazon RDS[14]

  您可以使用 Amazon Relational Database Service (Amazon RDS) 提供的任意引擎。从您在 Lambda 中编写的代码直接连接 Amazon RDS，就像您在 Lambda 函数以外所做的那样，但具有与 AWS Key Management Service (AWS KMS) 简单集成以实现数据库凭证加密的优势。

- Amazon ElastiCache[15]

  将您的 Lambda 函数与托管的内存缓存集成，以提升应用程序的性能。

- Amazon RedShift[16]

  您可以构建能够安全地查询企业数据仓库以编制报告、仪表板或检索临时查询结果的函数。

- 由 Amazon Elastic Compute Cloud (Amazon EC2)[17] 托管的私有 Web 服务

  您可能有在 VPC 中作为 Web 服务以私有方式运行的现有应用程序。从 Lambda 函数通过您的逻辑私有 VPC 网络发起 HTTP 请求。

支持 IAM 的数据存储

由于 AWS Lambda 与 IAM 集成，它可以经由 IAM 实现与可通过使用 AWS API 直接使用的任意 AWS 服务的安全集成。

- Amazon DynamoDB[18]

  Amazon DynamoDB 是 AWS 的可无限扩展的 NoSQL 数据库。如果您需要在任意规模下实现低于 10 毫秒的数据记录（在本文撰写时不超过 400KB）检索性能，则可以考虑 Amazon DynamoDB。借助 Amazon DynamoDB 精细的访问控制功能，您的 Lambda 函数在查询 DynamoDB 中的特定数据时可遵循最小特权原则的最佳实践。

- Amazon S3[19]

  Amazon Simple Storage Service (Amazon S3) 提供 Internet 规模的对象存储。Amazon S3 可实现 99.999999999% 的对象持久性，如果您的应用程序需要价格低廉的高持久性存储，可以考虑它。此外，Amazon S3 在给定一年内可实现高达 99.99% 的对象可用性，因此，如果您的应用程序需要高度可用的存储，也可以考虑它。存储在 Amazon S3 中的对象（文件、图像、日志、任何二进制数据）都可通过 HTTP 直接访问。Lambda 函数能够通过虚拟私有终端节点与 Amazon S3 安全地通信，S3 中可访问的数据也限于与该 Lambda 函数关联的 IAM 策略能够访问的数据。

- Amazon Elasticsearch Service[20]

  Amazon Elasticsearch Service (Amazon ES) 是常用的搜索和分析引擎 Elasticsearch 的托管版本。Amazon ES 提供了托管式的集群、故障检测和节点替换配置；您可以通过使用 IAM 策略来限制对 Amazon ES API 的访问。

表示层

Amazon API Gateway 为表示层带来了无限的可能。可通过 Internet 访问的 HTTPS API 可由具有 HTTPS 通信功能的任何客户端使用。下面的列表包含您可考虑用于应用程序表示层的常见示例：

- 移动应用程序：除了通过 Amazon API Gateway 和 AWS Lambda 与自定义业务逻辑集成外，您还可以使用 Amazon Cognito[21] 作为创建和管理用户身份的机制。
- 静态网站内容（如托管在 Amazon S3 中的文件）：您可以使自己的 Amazon API Gateway API 遵循跨源资源共享 (CORS) 标准。这样，Web 浏览器可以在静态网页中直接调用您的 API。
- 任何其他支持 HTTPS 的客户端设备：许多互连设备都能通过 HTTPS 进行通信。客户端与您使用 Amazon API Gateway 创建的 API 的通信方式没有任何特别之处；它是纯粹的 HTTPS。无需特定的客户端软件或许可证。

示例架构模式

您可以使用 Amazon API Gateway 和 AWS Lambda 作为构成逻辑层的粘合剂，从而实现下面常见的架构模式。对于每个示例，我们将只使用无需用户自行管理基础设施的 AWS 服务。

移动后端

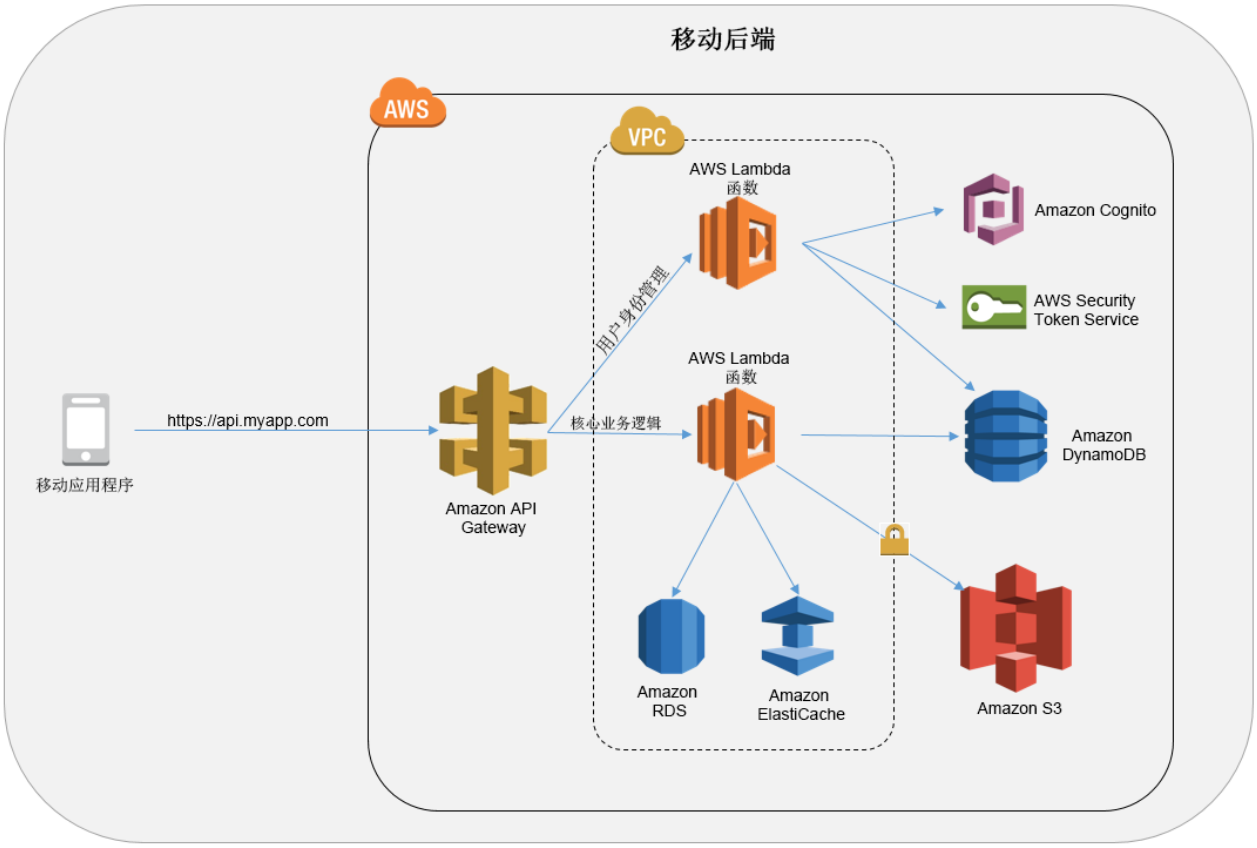

图 3：移动后端的架构模式

- **表示层：**在每个用户的智能手机上运行的移动应用程序。
- **逻辑层：**Amazon API Gateway 和 AWS Lambda。逻辑层通过作为每个 Amazon API Gateway API 的一部分创建的 Amazon CloudFront 分配进行全球分发。一组 Lambda 函数（可特定于用户/设备身份管理和身份验证），由 Amazon Cognito 托管，后者提供与 IAM（以提供临时的用户访问凭证）及常见的第三方身份提供商的集成。其他 Lambda 函数可定义移动后端的核心业务逻辑。
- **数据层：**可以根据需要使用各种数据存储服务；本白皮书前面部分讨论过相关选项。

Amazon S3 托管网站

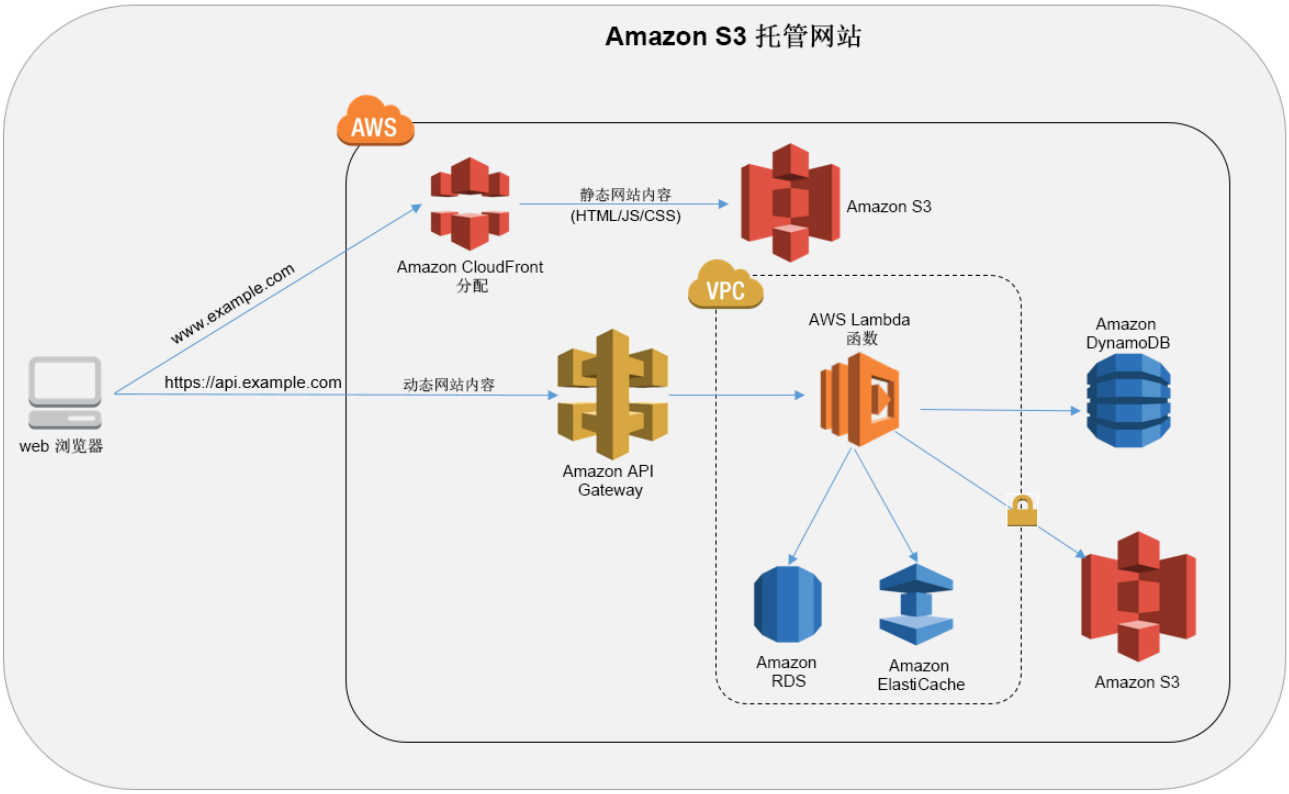

图 4：托管在 Amazon S3 上的静态网站的架构模式

- **表示层：**托管在 Amazon S3 中的静态网站内容，由 Amazon CloudFront 分发。将静态网站内容托管在 Amazon S3 上是将内容托管在基于服务器的基础设施上的一种经济高效替代方法。但是，对于包含丰富功能的网站，静态内容往往必须与动态后端集成。
- **逻辑层**：Amazon API Gateway 和 AWS Lambda。托管在 Amazon S3 中的静态 Web 内容可与 Amazon API Gateway 直接集成，遵循 CORS 标准。
- **数据层：**可以根据需要使用各种数据存储服务。本白皮书前面部分讨论过这些选项。

微服务环境

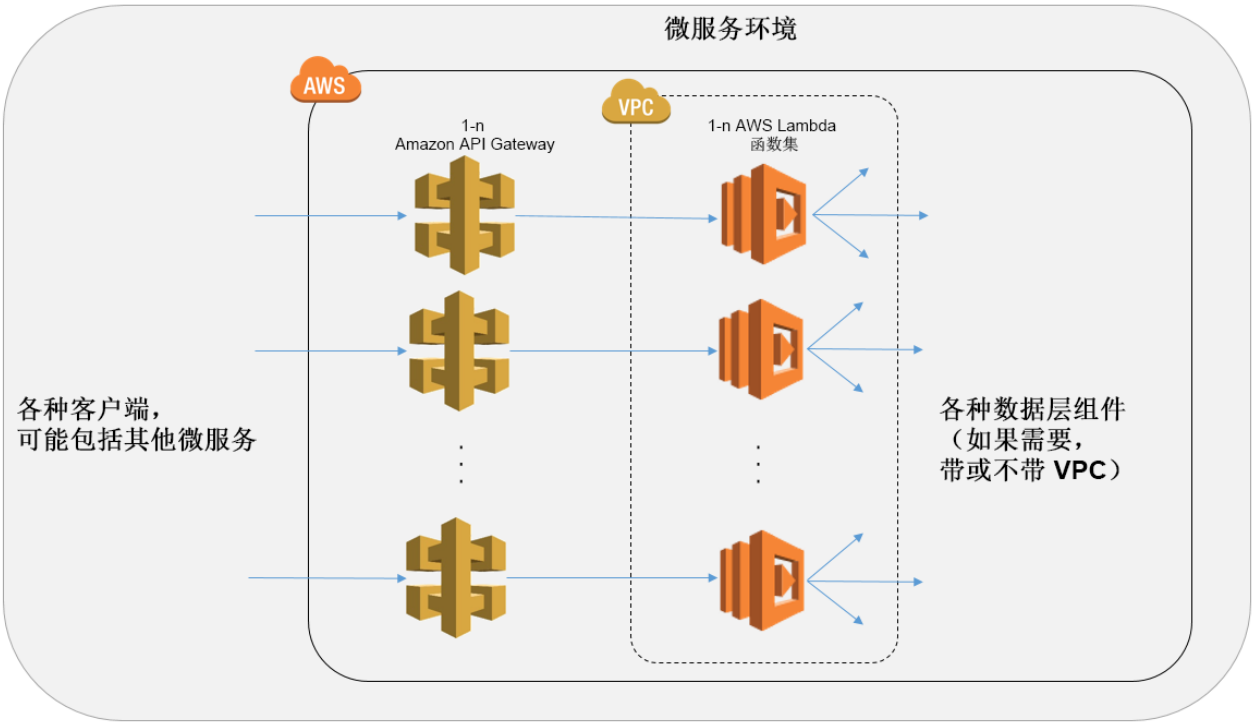

图 5：微服务环境的架构模式

微服务架构模式与我们在本白皮书中讨论的典型三层架构没有关联。微服务架构中存在大量的软件组件解耦，因此，多层架构的优势会放大。要构建微服务，您只需要使用 Amazon API Gateway 创建 API，并通过 AWS Lambda 进行函数的后续执行。您的团队可自由地使用这些服务解耦和分割环境，使其达到所需的粒度级别。

通常，微服务环境可能会带来以下难题：创建各个新的微服务的重复开销、优化服务器密度/利用率的相关问题、同时运行多个微服务的多个版本的复杂性、与许多独立服务集成所导致的客户端代码需求激增等。

但是，如果通过 AWS 无服务器模式创建微服务，这些问题会变得更易解决，在某些情况下，问题甚至直接就消失了。AWS 微服务模式减少了创建各个后续微服务的障碍，Amazon API Gateway 甚至允许克隆现有的 API。这种模式不存在优化服务器利用率的问题。API Gateway 和 Lambda 都提供简单的版本控制功能。最后，Amazon API Gateway 提供了使用多种常见语言编写的程序生成的客户端软件开发工具包，可减少集成开销。

结论

多层架构模式支持创建易于维护、解耦和可扩展的应用程序组件的最佳实践。如果您创建的逻辑层通过 Amazon API Gateway 进行集成并在 AWS Lambda 中进行计算，您已在实现这些目标的路上了，同时减少了实现它们所需耗费的精力。这些服务为您的客户端提供一个 HTTPS API 前端，并在 VPC 中提供一个用于执行业务逻辑的安全环境。这样，您可以充分发挥许多常见方案的优势：您可以使用这些托管服务，而不是自行管理基于服务器的典型基础设施。

贡献者

以下为对此文档有贡献的个人和组织：

Andrew Baird（AWS 解决方案架构师）

Stefano Buliani（AWS 移动部门高级产品经理、技术专家）

Vyom Nagrani（AWS 移动部门高级产品经理）

Ajay Nair（AWS 移动部门高级产品经理）

备注

[1] http://aws.amazon.com/api-gateway/

[2] http://aws.amazon.com/lambda/

[3] https://aws.amazon.com/vpc/

[4] https://aws.amazon.com/cloudfront/

[5] https://d0.awsstatic.com/whitepapers/DDoS_White_Paper_June2015.pdf

[6] https://aws.amazon.com/api-gateway/pricing/

[7] http://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-mock-integration.html

[8] http://aws.amazon.com/iam/

[9] http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html

[10] http://docs.aws.amazon.com/lambda/latest/dg/intro-core-components.html#intro-core-components-event-sources

[11] https://aws.amazon.com/s3/

[12] https://aws.amazon.com/kinesis/

[13] https://aws.amazon.com/vpc/

[14] https://aws.amazon.com/rds/

[15] https://aws.amazon.com/elasticache/

[16] https://aws.amazon.com/redshift/

[17] https://aws.amazon.com/ec2/

[18] https://aws.amazon.com/dynamodb/

[19] https://aws.amazon.com/s3/storage-classes/

[20] https://aws.amazon.com/elasticsearch-service/

[21] https://aws.amazon.com/cognito/