AWS 白皮书

使用 Amazon API Gateway 和 AWS Lambda 的 AWS 无服务器多层架构

使用 Amazon API Gateway 和 AWS Lambda 的 AWS 无服务器多层架构：AWS 白皮书

Table of Contents

使用 Amazon API Gateway 和 AWS Lambda 的 AWS 无服务器多层架构

发布日期：2021 年 10 月 20 日（文档修订）

摘要

本白皮书阐述了如何利用 Amazon Web Services（AWS）提供的创新来改变设计多层架构和实施常用模式（如微服务、移动后端和单页应用程序）的方式。构架师和开发人员可以使用 Amazon API Gateway、AWS Lambda 和其他服务来缩短创建和管理多层应用程序所需的开发和运营周期。

引言

几十年来，多层应用程序（三层、n 层等）一直是重要的架构模式，并且仍然是面向用户的应用程序的常用模式。尽管用于描述多层架构的说法各不相同，但多层应用程序通常由以下组件构成：

- 表示层：与用户直接交互的组件（例如，网页和移动应用程序 UI）。
- 逻辑层：将用户操作转换为应用程序功能（例如，CRUD 数据库操作和数据处理）所需的代码。
- 数据层：保存应用程序相关数据的存储介质（例如，数据库、对象存储、缓存和文件系统）。

多层架构模式提供了一个通用框架，可确保能够独立开发、管理和维护（通常由不同的团队执行）分离且可独立扩展的应用程序组件。

在这种模式中，网络（一个层必须进行网络调用才能与另一层交互）充当层之间的边界，因此，开发多层应用程序通常需要创建许多无差别的应用程序组件。这些组件包括：

- 定义用于层间通信的消息队列的代码
- 定义应用程序编程接口（API）和数据模型的代码
- 确保正确访问应用程序的安全相关代码

所有这些示例都可以被视为“样板”组件，尽管这些组件在多层应用程序中是必需的，但在不同的应用程序之间，它们的实现并没有很大差异。

AWS 提供了许多支持创建无服务器多层应用程序的服务，极大地简化了将此类应用程序部署到生产环境的过程，并消除了与传统服务器管理相关的开销。Amazon API Gateway 是一项用于创建和管理 API 的服务，AWS Lambda 是一项用于运行任意代码函数的服务，这两者结合可以方便地创建强大的多层应用程序。

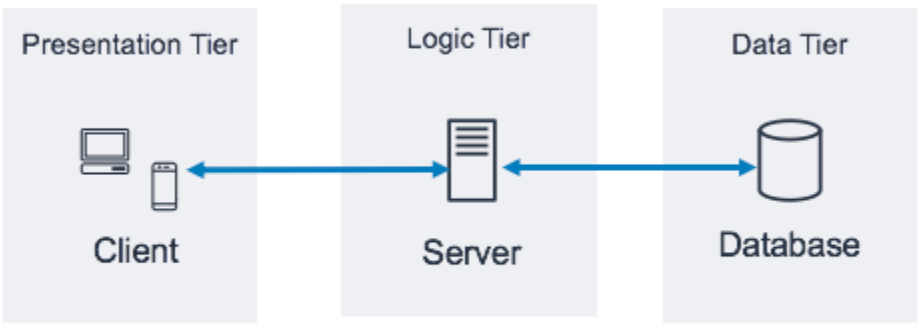
Amazon API Gateway 与 AWS Lambda 集成后，用户定义的代码函数能够直接通过 HTTPS 请求启动。无论请求量如何，API Gateway 和 Lambda 都可以自动实现扩展以完全满足应用程序的需求（请参阅 Amazon API Gateway 配额和重要说明，以了解可扩展性相关信息）。通过将这两种服务相结合，您可以创建一个层，使您只需编写对应用程序至关重要的代码，而不必关注实施多层架构的其他各个无差别方面，例如设计高可用性架构、编写客户端 SDK、服务器和操作系统（OS）管理，以及扩展和实施客户端授权机制等方面。

API Gateway 和 Lambda 支持创建无服务器逻辑层。根据您的应用程序要求，AWS 还提供有相应的选项用于创建无服务器表示层（例如，使用 Amazon CloudFront 和 Amazon Simple Storage Service）和数据层（例如，使用 Amazon Aurora 和 Amazon DynamoDB）。

本白皮书重点介绍的是最常见的多层架构示例，即 三层 Web 应用程序。但是，这种多层模式的应用远不止典型的三层 Web 应用程序。

三层架构概览

三层架构是最常见的多层架构实现，它由一个表示层、一个逻辑层和一个数据层组成。下图显示了一个简单的通用三层应用程序示例。



三层应用程序的架构模式

网上有许多很好的在线资源，可帮助您了解有关通用三层架构模式的更多信息。本白皮书重点介绍使用 Amazon API Gateway 和 AWS Lambda 的三层架构的特定实现模式。

无服务器逻辑层

三层架构的逻辑层代表应用程序的大脑。这是使用 Amazon API Gateway 和 AWS Lambda 的模式与基于服务器的传统实现区别最大的地方。这两项服务的功能使您能够构建高度可用、可扩展且安全的无服务器应用程序。在传统模式中，您的应用程序可能需要数千台服务器；但是，通过使用 Amazon API Gateway 和 AWS Lambda，您不必负责任何规模的服务器管理。此外，通过结合使用这些托管式服务，您还将获得以下优势：

- AWS Lambda：
  - 无需选择、保护、修补或管理操作系统
  - 无需调整服务器大小，也无需监控或扩展服务器
  - 减少因资源过度调配给您带来的成本风险
  - 减少因资源调配不足给您带来的性能风险

- Amazon API Gateway：
  - 简化了部署、监控和保护 API 的机制
  - 通过缓存和内容分发提高了 API 性能

AWS Lambda

AWS Lambda 是一项计算服务，使您能够使用任何受支持的语言（Node.js、Python、Ruby、Java、Go、.NET，有关更多信息，请参阅 Lambda 常见问题）运行任意代码函数，而无需调配、管理或扩展服务器。Lambda 函数在托管的隔离容器中运行，并在响应事件时启动，该事件可以是 AWS 提供的多个编程触发器之一，称为事件源。 有关所有事件源，请参阅 Lambda 常见问题。

Lambda 的许多常见使用案例都围绕事件驱动型数据处理工作流，例如处理存储在 Amazon S3 中的文件或从 Amazon Kinesis 流式传输数据记录。当与 Amazon API Gateway 结合使用时，Lambda 函数将执行典型的 Web 服务功能：它会启动代码以响应客户端 HTTPS 请求；API Gateway 充当逻辑层的前门，而 AWS Lambda 调用应用程序代码。

您的业务逻辑就在这里，无需使用服务器

Lambda 需要您编写代码函数，我们称之为处理程序，当事件启动处理程序时，它们就会运行。要将 Lambda 与 API Gateway 结合使用，您可以将 API Gateway 配置为当 HTTPS 请求调用您的 API 时启

动处理程序函数。在无服务器多层架构中，您在 API Gateway 中创建的每个 API 都将与一个调用所需业务逻辑的 Lambda 函数（以及其中的处理程序）集成。

通过使用 AWS Lambda 函数组成逻辑层，您可以定义公开应用程序功能所需的粒度级别（每个 API 一个 Lambda 函数或每个 API 方法一个 Lambda 函数）。在 Lambda 函数内，处理程序可以访问任何其他依赖项（例如，您随代码上载的其他方法、库、本机二进制文件和外部 Web 服务），甚至其他 Lambda 函数。

创建或更新 Lambda 函数需要将代码以 Lambda 部署程序包 zip 文件的形式上载到 Amazon S3 存储桶，或者将代码连同所有依赖项一起打包为容器映像。函数可以使用不同的部署方法，例如 AWS 管理控制台、运行 AWS Command Line Interface（AWS CLI）或运行基础设施即代码模板或框架，例如 AWS CloudFormation、AWS Serverless Application Model（AWS SAM）或 AWS Cloud Development Kit (AWS CDK)。使用上述任何一种方法创建函数时，都需要指定部署包中的哪个方法将充当请求处理程序。您可以为多个 Lambda 函数定义重复使用同一个部署程序包，而每个 Lambda 函数在同一个部署程序包中可能有一个唯一的处理程序。

Lambda 安全性

要运行 Lambda 函数，必须由 AWS Identity and Access Management（IAM）策略允许的事件或服务调用该函数。使用 IAM 策略，您可以创建一个 Lambda 函数，该函数只有当您定义的 API Gateway 资源调用时才会启动。可以跨各种 AWS 服务使用基于资源的策略来定义此类策略。

每个 Lambda 函数都需要一个 IAM 角色，在部署 Lambda 函数时分配给该函数。此 IAM 角色定义了您的 Lambda 函数可以与之交互的其他 AWS 服务和资源（例如 Amazon DynamoDB Amazon S3）。在 Lambda 函数的上下文中，我们称之为执行角色。

请勿在 Lambda 函数中存储敏感信息。IAM 通过 Lambda 执行角色处理对 AWS 服务的访问；如果您需要从 Lambda 函数内部访问其他凭证（例如数据库凭证和 API 密钥），可以将 AWS Key Management Service（AWS KMS）与环境变量结合使用，或使用像 AWS Secrets Manager 这样的服务，以便在不使用时保护此信息的安全。

规模性能

从 Amazon Elastic Container Registry（Amazon ECR）或者从上载到 Amazon S3 的 zip 文件中以容器映像形式拉入的代码在 AWS 管理的隔离环境中运行。您不必扩展 Lambda 函数 – 每次您的函数收到事件通知时，AWS Lambda 都会在其计算队列中查找可用容量，并使用您定义的运行时、内存、磁盘和超时配置运行代码。通过这种模式，AWS 可以根据需要启动任意数量的函数副本。

基于 Lambda 的逻辑层的规模始终适合您的客户需求。通过托管式扩缩和并发代码启动快速吸收激增的流量，再加上 Lambda 按使用量付费的定价模式，使您能够始终满足客户要求，同时无需为闲置的计算容量付费。

无服务器部署和管理

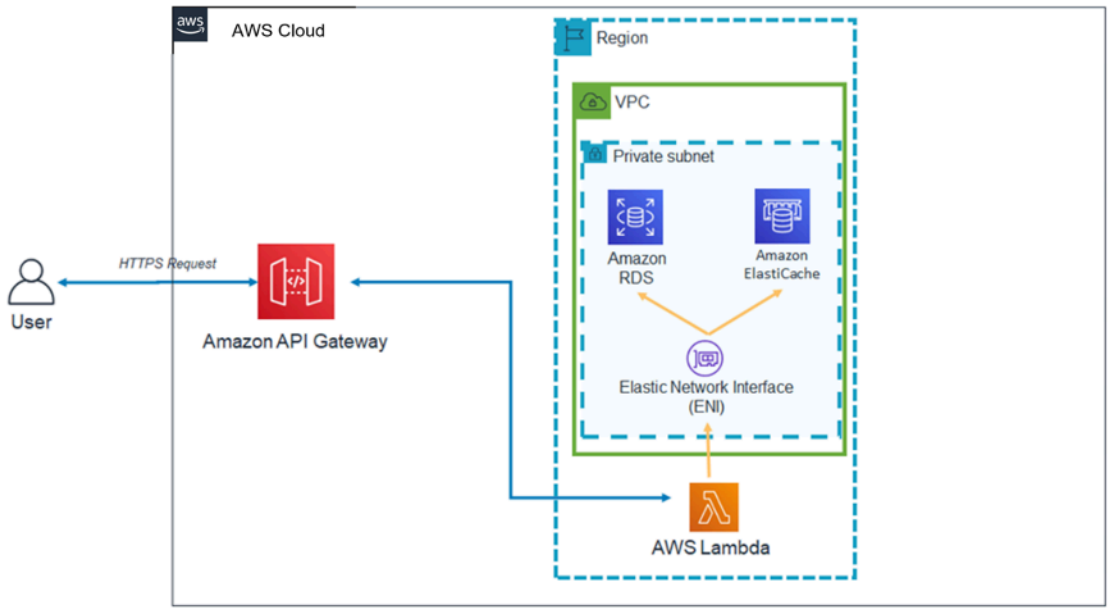
要帮助您部署和管理 Lambda 函数，请使用 AWS SAM AWS Serverless Application Model（AWS SAM），这是一个包含以下内容的开源框架：

- AWS SAM 模板规范 – 用于定义函数并描述其环境、权限、配置和事件，以简化上载和部署的语法。
- AWS SAM CLI – 允许您验证 SAM 模板语法，本地调用函数，调试 Lambda 函数和部署包函数的命令。

您还可以使用 AWS CDK，这是一个软件开发框架，用于使用编程语言定义云基础设施，并通过 CloudFormation 对其进行调配。CDK 定义 AWS 资源采用的是命令式方式，而 AWS SAM 采用的是声明式方式。

通常，当您部署 Lambda 函数时，系统会使用所分配的 IAM 角色定义的权限来调用该函数，并且能够访问面向互联网的端点。作为逻辑层的核心，AWS Lambda 是直接与数据层集成的组件。如果您的数据层包含敏感的业务或用户信息，请务必确保此数据层被适当隔离（在私有子网中）。

如果您希望 Lambda 函数访问您不能公开的资源（如私有数据库实例），可以将 Lambda 函数配置为连接到您的 AWS 账户中 Virtual Private Cloud（VPC）中的私有子网。当您将函数连接到 VPC 时，Λ会在函数的 VPC 配置中为每个子网创建一个弹性网络接口，用于私密访问您的内部资源。



VPC 内部的 Lambda 架构模式

将 Lambda 与 VPC 结合使用意味着，您可以让您的业务逻辑所依赖的数据库和其他存储介质无法通过互联网进行访问。VPC 还能确保，只有通过您定义的 API 和您编写的 Lambda 代码函数才能从互联网与您的数据进行交互。

Amazon API Gateway

Amazon API Gateway 是一项完全托管式服务，它使开发人员能够创建、发布、维护、监控和保护任何规模的 API。

客户端（即表示层）使用标准 HTTPS 请求与通过 API Gateway 公开的 API 集成。通过 API Gateway 公开的 API 是否适用于面向服务的多层架构，取决于是否能够分离应用程序的各项功能并通过 REST 端点公开此功能。Amazon API Gateway 具有特定的功能和品质，可以为您的逻辑层添加强大的功能。

与 AWS Lambda 集成

Amazon API Gateway 支持 REST 和 HTTP 类型的 API。API Gateway API 由资源和方法组成。资源是一种逻辑实体，应用程序可以通过资源路径（例如 `/tickets`）来访问资源。方法对应于提交给 API 资源的 API 请求（例如 `GET /tickets`）。通过 API Gateway，您可以使用 Lambda 函数支持每种方法，也就是说，当您通过 API Gateway 中公开的 HTTPS 端点调用 API 时，API Gateway 会调用此 Lambda 函数。

您可以使用代理集成和非代理集成来连接 API Gateway 和 Lambda 函数。

代理集成

在代理集成中，整个客户端 HTTPS 请求将按原样发送到 Lambda 函数。API Gateway 将整个客户端请求作为 Lambda 处理程序函数的事件参数进行传递，Lambda 函数的输出将直接返回给客户端（包括状态代码、标头等）。

非代理集成

在非代理集成中，您可以配置如何将客户端请求的参数、标头和正文传递给 Lambda 处理程序函数的事件参数。此外，您还可以配置如何将 Lambda 输出转换给用户。

> **Note**
>
> API Gateway 还可以代理连接到 AWS Lambda 外部的其他无服务器资源，例如模拟集成（用于初始应用程序开发），以及直接代理连接到 S3 对象。

跨区域的稳定 API 性能

Amazon API Gateway 的每个部署都包含一个 Amazon CloudFront 分配。CloudFront 是一项内容分发服务，它以 Amazon 的全球边缘站点网络作为连接点，连接到使用您的 API 的客户端。这有助于降低 API 的响应延迟。凭借遍布世界各地的多个边缘站点，Amazon CloudFront 还提供了抵御分布式拒绝服务（DDoS）攻击场景的能力。有关更多信息，请查看实现 DDoS 弹性的 AWS 最佳实践白皮书。

您可以使用 API Gateway 将响应存储在可选的内存中的缓存中，从而提高特定 API 请求的性能。这种方法不仅可以为重复的 API 请求提供性能优势，而且还可以减少调用 Lambda 函数的次数，从而降低总体成本。

利用内置功能鼓励创新并减少开销

构建任何新应用程序的开发成本都是一项投资。使用 API Gateway 可以减少某些开发任务所需的时间，并降低总开发成本，使企业能够更自由地进行实验和创新。

在最初的应用程序开发阶段，为了更快地交付新的应用程序，日志记录和指标收集的实现常常被忽略。在将这些功能部署到在生产环境中运行的应用程序时，这可能会导致技术债务和运营风险。Amazon API Gateway 与 Amazon CloudWatch 无缝集成，后者可以从 API Gateway 收集原始数据并将其处理为可读的接近实时的指标，以监控 API 的执行情况。API Gateway 还支持带有可配置报告的访问日志

记录，以及用于调试的 AWS X-Ray 跟踪。所有这些功能都不需要编写任何代码，并且可以在生产环境中运行的应用程序中进行调整，而不会对核心业务逻辑造成风险。

您可能不知道应用程序的总体生命周期会有多长，也可能明确知道会很短。如果您的开发起点已经包含 API Gateway 提供的托管式功能，并且仅在 API 开始接收请求后才产生基础设施成本，那么构建此类应用程序的立项分析就会变得更加容易。有关更多信息，请参阅 Amazon API Gateway 定价。

快速迭代，保持敏捷

使用 Amazon API Gateway 和 AWS Lambda 构建 API 的逻辑层，您可以通过简化 API 部署和版本管理来快速适应不断变化的用户群需求。

阶段部署

在 API Gateway 中部署 API 时，必须将部署与 API Gateway 阶段相关联 – 每个阶段都是 API 的一个快照，可供客户端应用程序调用。使用此约定，您可以轻松地将应用程序部署到开发、测试、暂存或生产阶段，并在阶段之间移动部署。每次将 API 部署到阶段时，都会创建不同版本的 API，必要时可以将其还原。这些功能使现有功能和客户端依赖项能够在新功能作为单独的 API 版本发布时不受干扰地继续发挥作用。

分离与 Lambda 的集成

API Gateway 中的 API 和 Lambda 函数之间的集成可以使用 API Gateway 阶段变量和 Lambda 函数别名进行分离。这可以简化和加快 API 部署。您可以在 API 中配置可指向 Lambda 函数中特定别名的阶段变量，不用直接在 API 中配置 Lambda 函数名称或别名。在部署期间，将阶段变量值更改为指向 Lambda 函数别名，API 将在特定阶段的 Lambda 别名后面运行 Lambda 函数版本。

金丝雀发布（Canary release）部署

金丝雀发布（Canary release）是一种软件开发策略，在该策略中，您可以在同一阶段上部署一个新版的 API 用于测试，以及一个基础版本作为生产版本用于正常操作。在金丝雀发布（Canary release）部署中，全部 API 流量按照预配置的比率，随机拆分到生产版本和金丝雀发布（Canary release）中。可以为金丝雀发布（Canary release）部署配置 API Gateway 中的 API，以使有限的用户集测试新功能。

自定义域名

您可以为 API 提供直观的业务友好型 URL 名称，而不是 API Gateway 提供的 URL。API Gateway 提供了为 API 配置自定义域的功能。使用自定义域名，您可以设置 API 的主机名，并选择多级基本路径（例如 `myservice`、`myservice/cat/v1` 或 `myservice/dog/v2`），以将备用 URL 映射到 API。

优先考虑 API 安全性

所有应用程序都必须确保只有经过授权的客户端才能访问其 API 资源。在设计多层应用程序时，您可以通过 Amazon API Gateway 提供的几种方式来保护逻辑层：

传输安全性

所有对您的 API 的请求都可以通过 HTTPS 发出，以启用传输中加密。

API Gateway 提供内置的 SSL/TLS 证书 – 如果对面向公众的 API 使用自定义域名选项，您可以使用 AWS Certificate Manager 提供自己的 SSL/TLS 证书。API Gateway 还支持相互 TLS（mTLS）身份验证。相互 TLS 可以提高 API 的安全性，并帮助保护您的数据免受诸如客户端欺骗或中间人攻击等攻击。

API 授权

作为 API 的一部分创建的每个资源/方法组合都会被授予一个唯一的 Amazon Resource Name（ARN），可在 AWS Identity and Access Management（IAM）策略中引用该名称。

在 API Gateway 中向 API 添加授权的一般方法有三种：

- IAM 角色和策略：客户端使用 AWS 签名版本 4（SigV4）授权和 IAM 策略进行 API 访问。相同的凭证可以根据需要限制或允许访问其他 AWS 服务和资源（例如 Amazon S3 存储桶或 Amazon DynamoDB 表）。
- Amazon Cognito 用户池：客户端通过 Amazon Cognito 用户池登录并获取令牌，这些令牌包含在请求的授权标头中。
- Lambda 授权方：定义一个实现自定义授权方案的 Lambda 函数，该方案会使用持有者令牌策略（例如 OAuth 和 SAML）或使用请求参数来识别用户身份。

访问限制

API Gateway 支持生成 API 密钥，以及将这些密钥与可配置的使用计划相关联。您可以使用 CloudWatch 监控 API 密钥的使用情况。

API Gateway 支持对 API 中的每种方法进行节流、速率限制和突发速率限制。

私有 API

使用 API Gateway，您可以创建私有 REST API，这种 API 只能使用接口 VPC 终端节点从 Amazon VPC 中的 Virtual Private Cloud 访问。这是您在 VPC 中创建的端点网络接口。

通过使用资源策略，您可以允许或拒绝从选定的 VPC 和 VPC 终端节点（包括跨 AWS 账户）对您的 API 进行访问。每个终端节点都可用于访问多个私有 API。还可以使用 AWS Direct Connect 建立从本地网络到 Amazon VPC 的连接，并通过该连接访问您的私有 API。

在所有情况下，到您的私有 API 的流量都使用安全连接，并且不会离开 Amazon 网络，因为它与公共互联网是隔离的。

使用 AWS WAF 的防火墙保护

面向互联网的 API 容易受到恶意攻击。AWS WAF 是一个 Web 应用程序防火墙，有助于保护 API 免受此类攻击。它可以保护 API 免受常见的 Web 漏洞攻击，例如 SQL 注入攻击和跨站脚本攻击。您可以结合使用 AWS WAF 和 API Gateway 来帮助保护 API。

数据层

使用 AWS Lambda 作为逻辑层不会限制数据层中可用的数据存储选项。Lambda 函数通过在 Lambda 部署程序包中包含适当的数据库驱动程序来连接到任何数据存储选项，并使用基于 IAM 角色的访问权限或加密凭证（通过 AWS KMS 或 AWS Secrets Manager）。

为应用程序选择数据存储在很大程度上取决于应用程序的要求。AWS 提供了许多无服务器和非无服务器类型的数据存储，您可以使用这些存储来构成应用程序的数据层。

无服务器数据存储选项

Amazon S3 是一项对象存储服务，提供行业领先的可扩展性、数据可用性、安全性和性能。

Amazon Aurora 是一种与 MySQL 和 PostgreSQL 兼容的关系数据库，专为云环境打造，既具有传统企业数据库的性能和可用性，又具有开源数据库的简单性和成本效益。Aurora 同时提供无服务器和传统使用模式。

Amazon DynamoDB 是一种键值和文档数据库，可在任何规模下提供延迟不到十毫秒的性能。它是一个完全托管式无服务器、多区域、多主表的持久数据库，具有面向互联网规模应用程序的内置安全性、备份和还原以及内存中缓存功能。

Amazon Timestream 是一种快速、可扩展且完全托管式的时间序列数据库服务，适用于 IoT 和运营应用程序，通过该服务每天可以轻松存储和分析数万亿个事件，而成本仅为关系数据库的十分之一。在 IoT 设备、IT 系统和智能工业机器兴起的推动下，时间序列数据（衡量事物随时间变化的数据）是增长最快的数据类型之一。

Amazon Quantum Ledger Database（Amazon QLDB）是一种完全托管式分类账数据库，可提供由中央权威机构持有的透明、不可改变且可加密验证的事务日志。Amazon QLDB 可跟踪每次应用程序数据更改，并不断维护完整且可验证的更改历史记录。

Amazon Keyspaces（for Apache Cassandra）是一种可扩展、高度可用的托管式 Apache Cassandra 兼容数据库服务。通过 Amazon Keyspaces，您可以使用与目前相同的 Cassandra 应用程序代码和开发工具，在 AWS 上运行 Cassandra 工作负载。您无需预置、修补或管理服务器，且不需要安装、维护或操作软件。Amazon Keyspaces 是无服务器服务，因此您只需为实际使用的资源付费，并且该服务会根据应用程序流量自动扩展和缩减表。

Amazon Elastic File System（Amazon EFS）提供了一个简单的设置即用式无服务器弹性文件系统，让您无需预置或管理存储即可共享文件数据。它可与 AWS 云服务和本地资源配合使用，并且可按需

扩展至 PB 级，而不中断应用程序。借助 Amazon EFS，您可以在添加和删除文件时自动扩展和缩减文件系统，无需预置和管理容量以适应增长。Amazon EFS 可以通过 Lambda 函数挂载，这使其成为 API 的可行文件存储选项。

非无服务器数据存储选项

Amazon Relational Database Service（Amazon RDS）是一项托管式 Web 服务，该服务可使用任何可用的引擎（Amazon Aurora、PostgreSQL、MySQL、MariaDB、Oracle 和 Microsoft SQL Server），并可在针对内存、性能或 I/O 进行了优化的多个不同数据库实例类型上运行，从而让设置、操作和扩展关系数据库变得更加轻松。

Amazon Redshift 是一项完全托管式 PB 级云中数据仓库服务。

Amazon ElastiCache 是 Redis 或 Memcached 的完全托管式部署。无缝部署、运行和扩展常见兼容开源的内存数据存储。

Amazon Neptune 是一项快速、可靠且完全托管式图数据库服务，可帮助您轻松构建和运行使用高度互连数据集的应用程序。Neptune 支持常见的图模型（属性图和 W3C 资源描述框架（RDF））以及它们各自的查询语言，使您能够轻松构建可以高效导航高度互连数据集的查询。

Amazon DocumentDB（兼容 MongoDB）是一项快速、可扩展、高度可用且完全托管式的文档数据库服务，支持 MongoDB 工作负载。

最后，您还可以使用在 Amazon EC2 上独立运行的数据存储作为多层应用程序的数据层

表示层

表示层负责通过互联网上公开的 API Gateway REST 端点与逻辑层进行交互。任何支持 HTTPS 的客户端或设备都可以与这些端点进行通信，从而使表示层能够灵活地采用多种形式（桌面应用程序、移动应用程序、网页、IoT 设备等）。根据您的要求，表示层可使用以下 AWS 无服务器产品：任何支持 HTTPS 的客户端或设备都可以与这些端点进行通信，从而使表示层能够灵活地采用多种形式（桌面应用程序、移动应用程序、网页、IoT 设备等）。根据您的要求，表示层可使用以下 AWS 无服务器产品：

- Amazon Cognito – 一项无服务器用户身份和数据同步服务，使您能够快速、高效地将用户注册、登录和访问控制添加到您的 Web 和移动应用程序中。Amazon Cognito 可扩展至数百万用户，并支持使用社交身份提供商（如 Facebook、Google 和 Amazon）以及企业身份提供商通过 SAML 2.0 登录。
- 具有 CloudFront 的 Amazon S3 – 使您能够无需预置 Web 服务器，直接从 S3 存储桶为静态网站（例如单页应用程序）提供服务。您可以将 CloudFront 用作托管式内容分发网络（CDN）以提高性能，并使用托管式或自定义证书启用 SSL/TL。

AWS Amplify 是一组既可组合使用也可单独使用的工具和服务，能够帮助前端 Web 和移动开发人员构建由 AWS 提供支持的可扩展全栈式应用程序。Amplify 中的完全托管式服务依托于 Amazon 可靠的 CDN，凭借全球数百个节点以及可加快应用程序发布周期的内置 CI/CD 工作流，可在全球范围内部署和托管静态 Web 应用程序。Amplify 支持常见的 Web 框架（包括 JavaScript、React、Angular、Vue、Next.js）以及移动平台（包括 Android、iOS、React Native、Ionic 和 Flutter）。根据您的网络配置和应用程序要求，您可能需要使您的 API Gateway API 符合跨源资源共享（CORS）标准。符合 CORS 标准允许 Web 浏览器直接从静态网页中调用您的 API。

当您使用 CloudFront 部署网站时，系统会为您提供一个 CloudFront 域名以访问您的应用程序（例如 `d2d47p2vcczkh2.cloudfront.net`）。您可以使用 Amazon Route 53 注册域名并将其定向到您的 CloudFront 分配，或者将已拥有的域名定向到您的 CloudFront 分配。这样，用户就可以使用熟悉的域名访问您的站点。请注意，您还可以使用 Route 53 将自定义域名分配给 API Gateway 分配，这样用户就可以使用熟悉的域名调用 API。

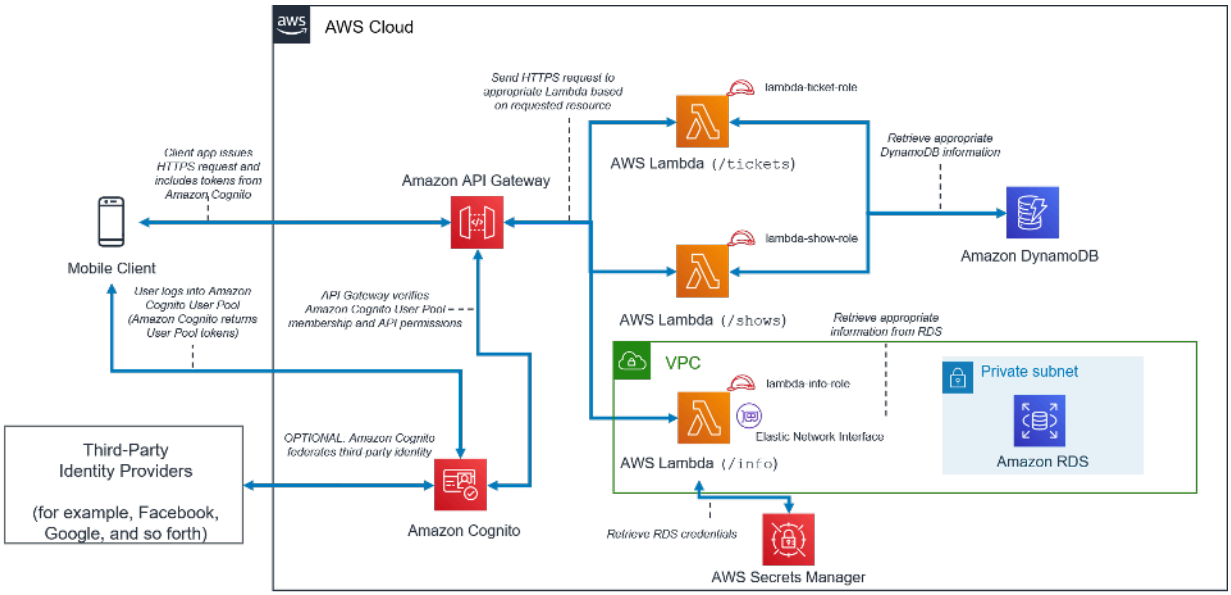
示例架构模式

您可以使用 API Gateway 和 AWS Lambda 作为逻辑层来实现常见的架构模式。本白皮书中包含使用 AWS Lambda 作为逻辑层的最常见架构模式：

- 移动后端 – 移动应用程序与 API Gateway 和 Lambda 通信以访问应用程序数据。这种模式可以扩展到通用 HTTPS 客户端，即不使用无服务器 AWS 资源来托管表示层资源的客户端（例如桌面客户端、在 EC2 上运行的 Web 服务器等）。
- 单页应用程序 – 在 Amazon S3 和 CloudFront 中托管的单页应用程序与 API Gateway 和 AWS Lambda 通信以访问应用程序数据。
- Web 应用程序 – Web 应用程序是通用的、事件驱动型 Web 应用程序后端，它使用 AWS Lambda 及 API Gateway 作为其业务逻辑。它还使用 DynamoDB 作为其数据库，并使用 Amazon Cognito 进行用户管理。所有静态内容都使用 Amplify 托管。

除了这两种模式之外，本白皮书还讨论了 Lambda 和 API Gateway 对一般微服务架构的适用性。微服务架构也是一种常见的模式，尽管它不是标准的三层架构，但它涉及分离应用程序组件，并将其部署为相互通信的无状态、独立的功能单元。

移动后端

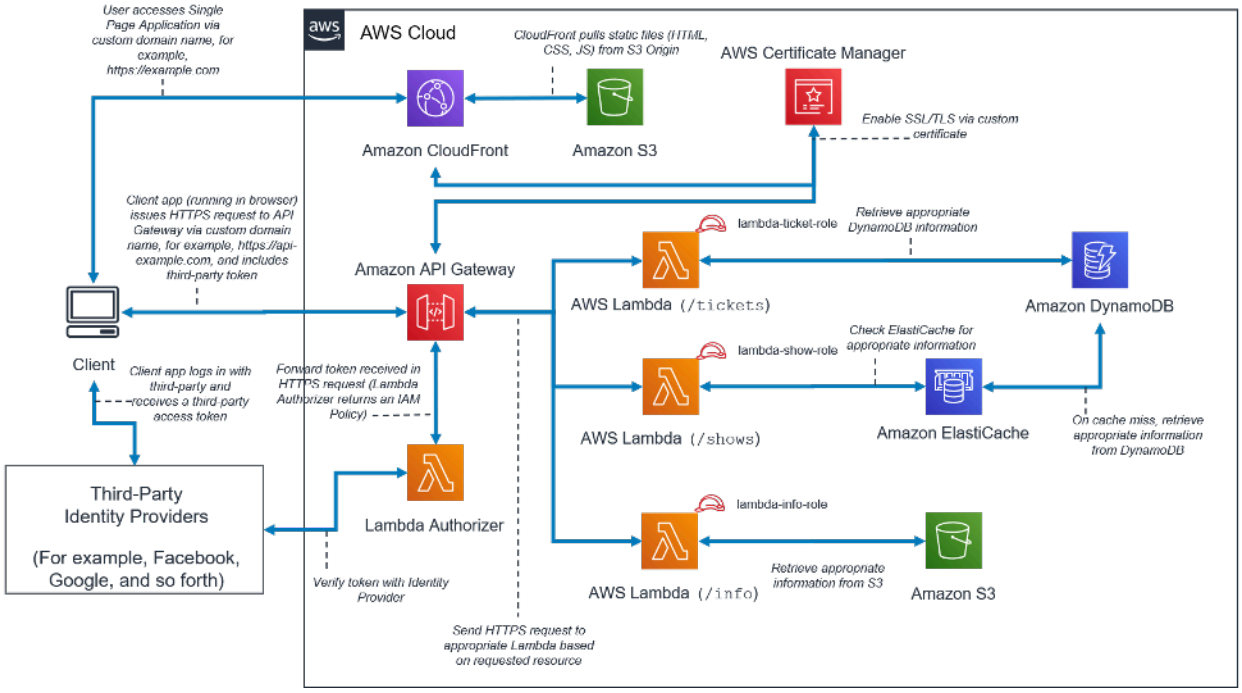


无服务器移动后端的架构模式

表 1 – 移动后端层组件

层	组件
表示	在用户设备上运行的移动应用程序。
逻辑	Amazon API Gateway 和 AWS Lambda。 此架构显示了三项公开服务（`/tickets`、`/shows` 和 `/info`）。API Gateway 端点由 Amazon Cognito 用户池提供保护。在此方法中，用户登录 Amazon Cognito 用户池（必要时使用联合第三方），并接收用于授权 API Gateway 调用的访问权限和 ID 令牌。 每个 Lambda 函数都分配了自己的 Identity and Access Management（IAM）角色，以提供对相应数据源的访问权限。
数据	DynamoDB 用于 `/tickets` 和 `/shows` 服务。 Amazon RDS 用于 `/info` 服务。此 Lambda 函数从 AWS Secrets Manager 中检索 Amazon RDS 凭证，并使用弹性网络接口访问私有子网。

单页应用程序



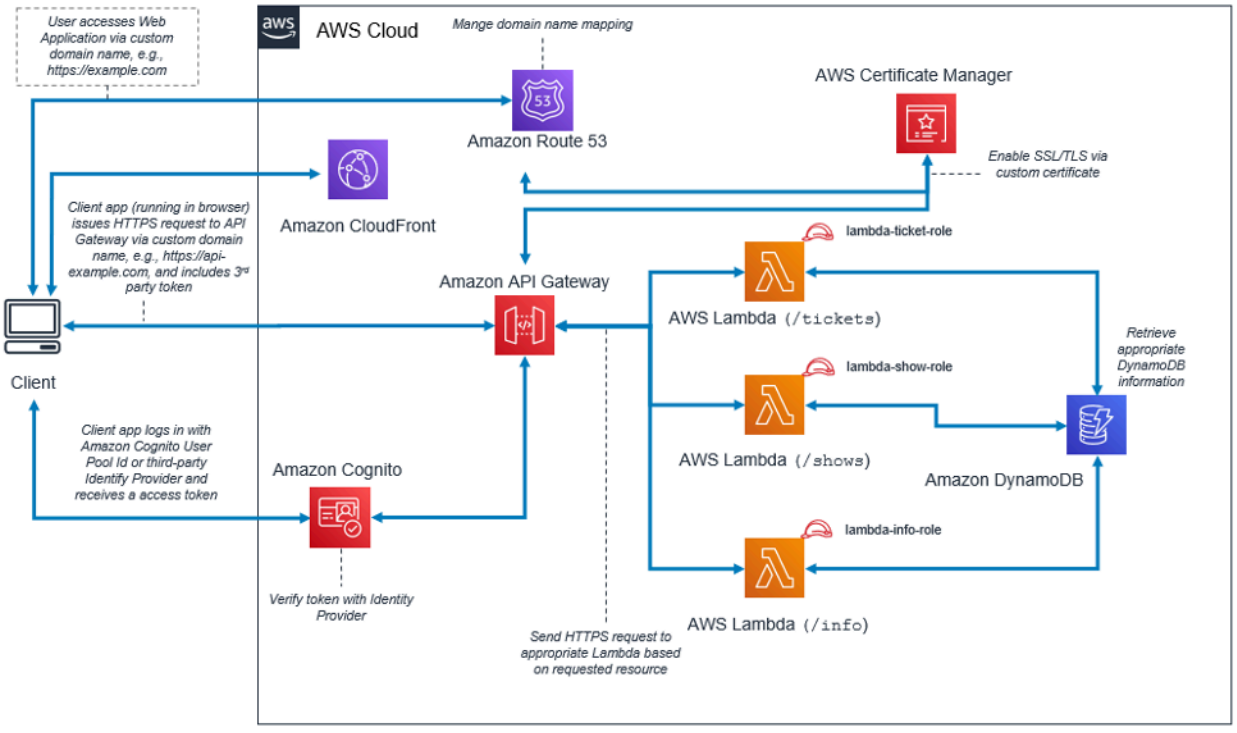
无服务器单页应用程序的架构模式

表 2 – 单页应用程序组件

层	组件
表示	Amazon S3 中托管的静态网站内容，由 CloudFront 分发。 AWS Certificate Manager 允许使用自定义 SSL/TLS 证书。
逻辑	API Gateway 和 AWS Lambda。 此架构显示了三项公开服务（`/tickets`、`/shows` 和 `/info`）。API Gateway 端点由 Lambda 授权方提供安全保护。在此方法中，用户通过第三方身份提供商登录，并获取访问权限和 ID 令牌。这些令牌包含在 API Gateway 调用中，Lambda 授权方会验证这些令牌，并生成包含 API 启动权限的 IAM 策略。

层	组件
	每个 Lambda 函数都分配了自己的 IAM 角色，以提供对相应数据源的访问权限。
数据	Amazon DynamoDB 用于 `/tickets` 和 `/shows` 服务。 `/shows` 服务使用 Amazon ElastiCache 来提高数据库性能。缓存未命中将发送到 DynamoDB。 Amazon S3 用于托管由 `/info service` 使用的静态内容。

Web 应用程序

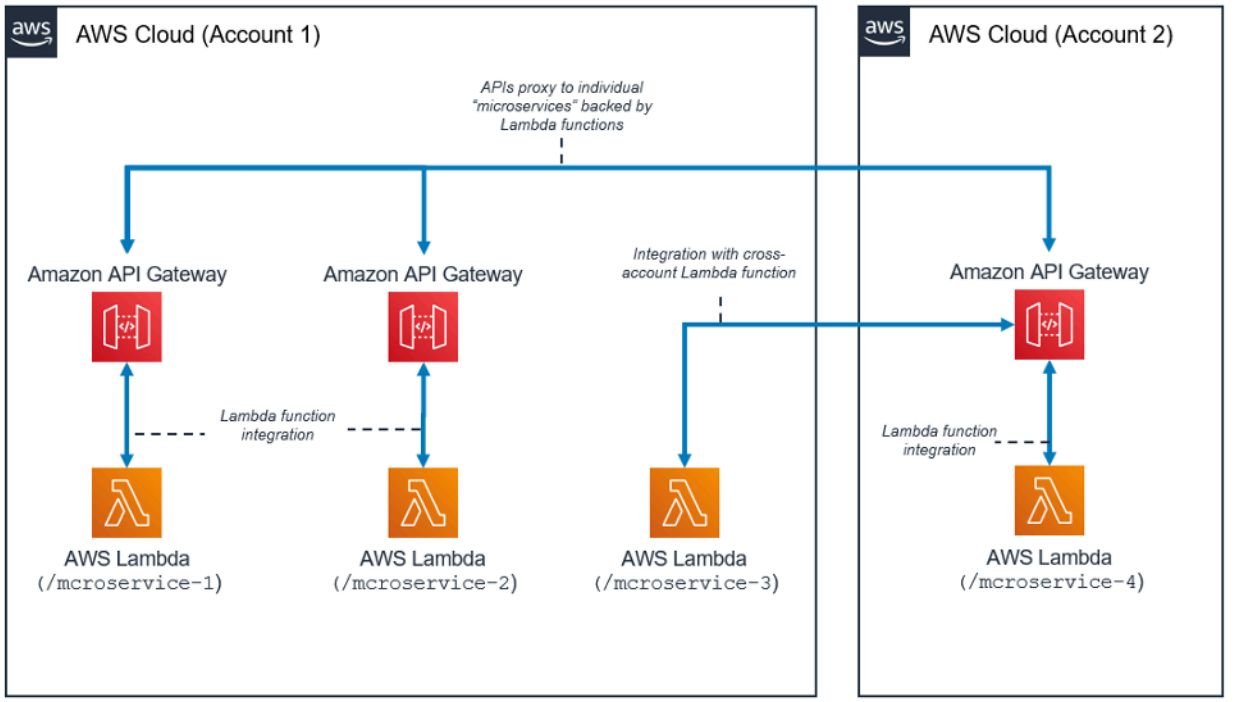


Web 应用程序的架构模式

表 3 – Web 应用程序组件

层	组件
表示	前端应用程序都是静态内容（HTML、CSS、JavaScript 和图像），它们都是由 create-react-app 等 React 实用工具生成的。Amazon CloudFront 可托管所有这些对象。使用 Web 应用程序时，会将所有资源下载到浏览器并从那里开始运行。Web 应用程序连接到调用 API 的后端。
逻辑	逻辑层是使用 API Gateway REST API 前端的 Lambda 函数构建的。 此架构显示了多项公开服务。有多个不同的 Lambda 函数，每个函数负责处理应用程序的不同方面。Lambda 函数位于 API Gateway 之后，可使用 API URL 路径进行访问。 用户身份验证使用 Amazon Cognito 用户池或联合身份用户提供商进行处理。API Gateway 使用与 Amazon Cognito 的开箱即用集成。只有在用户通过身份验证后，客户端才会收到一个 JSON Web 令牌（JWT），然后在进行 API 调用时应使用该令牌。 每个 Lambda 函数都分配了自己的 IAM 角色，以提供对相应数据源的访问权限。
数据	此特定示例中使用 DynamoDB 作为数据存储，但根据使用案例和使用场景，也可以使用其他专门构建的 Amazon 数据库或存储服务。

使用 Lambda 的微服务



使用 Lambda 的微服务的架构模式

微服务架构模式并不局限于典型的三层架构；但是，这种常见的模式可以因使用无服务器资源获得显著的优势。

在此架构中，每个应用程序组件都是分离的，可独立部署和操作。构建微服务所需要的全部要素，就是使用 Amazon API Gateway 创建的 API，以及随后由 AWS Lambda 启动的函数。您的团队可以使用这些服务分离环境并将其分割为所需的粒度级别。

通常，微服务环境可能会带来以下困难：创建每个新微服务的重复开销、优化服务器密度和利用率的问题、同时运行多个版本的多个微服务的复杂性，以及与许多单独服务集成所需的客户端代码激增。

当您使用无服务器资源创建微服务时，这些问题就变得不那么难以解决，在某些情况下，甚至会消失。无服务器微服务模式减少了创建每个后续微服务的障碍（API Gateway 甚至允许克隆现有 API，并在其他账户中使用 Lambda 函数）。这种模式不再需要优化服务器利用率。最后，Amazon API Gateway 还提供了以多种常见语言通过编程方式生成的客户 SDK，以减少集成开销。

总结

设计多层架构模式时，建议采用在创建易于维护、分离和扩展的应用程序组件时所采用的最佳实践。当您创建由 API Gateway 进行集成并在 AWS Lambda 中进行计算的逻辑层时，可以在实现这些目标的同时减少所需的工作量。这些服务共同为您的客户端提供了 HTTPS API 前端，并提供了一个安全的环境来应用业务逻辑，同时消除了管理典型的基于服务器的基础设施所涉及的开销。

贡献者

本文档的贡献者包括：

- AWS 解决方案构架师 Andrew Baird
- AWS ProServe 顾问 Bryant Bost
- AWS Mobile 技术部高级产品经理 Stefano Buliani
- AWS Mobile 高级产品经理 Vyom Nagrani
- AWS Mobile 高级产品经理 Ajay Nair
- 全球解决方案构架师 Rahul Popat
- 高级解决方案构架师 Brajendra Singh

延伸阅读

有关更多信息，请参阅：

- AWS 白皮书和指南

文档修订

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
更新了白皮书	针对新的服务功能和模式进行了更新。	2021 年 10 月 20 日
更新了白皮书	针对新的服务功能和模式进行了更新。	2021 年 6 月 1 日
更新了白皮书	针对新的服务功能进行了更新。	2019 年 9 月 25 日
初次发布	发布白皮书。	2015 年 11 月 1 日

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 对其客户承担的责任和义务受 AWS 协议制约，本文档不是 AWS 与客户直接的协议的一部分，也不构成对该协议的修改。