AWS 백서

Amazon API Gateway와 AWS Lambda를 사용한 AWS 서버리스 멀티 티어 아키텍처

Amazon API Gateway와 AWS Lambda를 사용한 AWS 서버리스 멀티 티어 아키텍처 : AWS 백서

Table of Contents

Amazon API Gateway와 AWS Lambda를 사용한 AWS 서버리스 멀티 티어 아키텍처

게시 날짜: 2021년 10월 20일(문서 개정)

요약

본 백서에서는 Amazon Web Services(AWS)의 혁신을 활용하여 멀티 티어 아키텍처의 설계 방식을 변경하고 마이크로서비스, 모바일 백엔드, 단일 페이지 애플리케이션과 같은 인기 패턴을 구현하는 방법을 설명합니다. 아키텍트와 개발자는 Amazon API Gateway, AWS Lambda 및 기타 서비스를 이용하여 멀티 티어 애플리케이션을 생성하고 관리하는 데 필요한 개발 및 운영 주기를 단축할 수 있습니다.

소개

멀티 티어 애플리케이션(3티어, n티어 등)은 수십 년 동안 초석이 되는 아키텍처 패턴이었으며 사용자 대면 애플리케이션에 널리 사용되는 패턴으로 남아 있습니다. 멀티 티어 아키텍처를 설명하는 데 사용되는 언어는 다양하지만 멀티 티어 애플리케이션은 일반적으로 다음 구성 요소로 구성됩니다.

- 프레젠테이션 티어: 사용자가 직접 상호 작용하는 구성 요소(예: 웹 페이지 및 모바일 앱 UI)
- 로직 티어: 사용자 작업을 애플리케이션 기능(예: CRUD 데이터베이스 작업 및 데이터 처리)으로 변환하는 데 필요한 코드
- 데이터 티어: 애플리케이션과 관련된 데이터를 보관하는 스토리지 미디어(예: 데이터베이스, 객체 스토어, 캐시 및 파일 시스템)

멀티 티어 아키텍처 패턴은 분리되고 독립적으로 확장 가능한 애플리케이션 구성 요소가 개별적으로 개발, 관리 및 유지될 수 있도록 보장하는 일반 프레임워크를 제공합니다(종종 개별 팀에 의해).

네트워크(티어가 다른 티어와 상호 작용하기 위해 네트워크 호출을 해야 함)가 티어 간의 경계 역할을 하는 이러한 패턴의 결과로, 멀티 티어 애플리케이션을 개발하려면 종종 차별화되지 않은 여러 애플리케이션 구성 요소를 생성해야 합니다. 이러한 구성 요소 중 일부는 다음과 같습니다.

- 티어 간 통신을 위한 메시지 대기열을 정의하는 코드
- 애플리케이션 프로그램 인터페이스(API) 및 데이터 모델을 정의하는 코드
- 애플리케이션에 대한 적절한 액세스를 보장하는 보안 관련 코드

이러한 모든 예는 멀티 티어 애플리케이션에 필요하지만 애플리케이션마다 구현이 크게 다르지 않은 '표준 문안' 구성 요소로 간주될 수 있습니다.

AWS는 서버리스 멀티 티어 애플리케이션 생성을 가능하게 하는 다양한 서비스를 제공하여 이러한 애플리케이션을 프로덕션에 배포하는 프로세스를 크게 단순화하고 기존 서버 관리와 관련된 오버헤드를 제거합니다. API 생성 및 관리 서비스인 Amazon API Gateway와 임의 코드 함수 실행 서비스인 AWS Lambda를 함께 사용하여 강력한 멀티 티어 애플리케이션 생성을 간소화할 수 있습니다.

Amazon API Gateway와 AWS Lambda의 통합을 통해 HTTPS 요청을 통해 사용자 정의 코드 함수를 직접 시작할 수 있습니다. 요청 볼륨에 관계없이 API Gateway와 Lambda는 모두 애플리케이션의 요구 사항을 정확히 지원하도록 자동으로 확장됩니다(확장성 정보는 API Gateway Amazon API Gateway 할당량 및 중요 참고 사항 참조). 이 두 서비스를 결합하여 애플리케이션에 중요한 코드만 작성할 수 있고 멀티 티어 아키텍처 구현의 차별화되지 않은 다른 측면(예: 고가용성 아키텍팅, 클라이언트 SDK 작

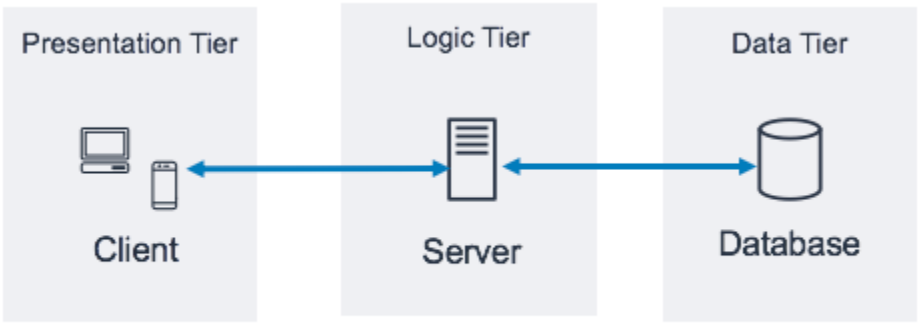
성, 서버 및 운영 체제(OS) 관리, 확장, 클라이언트 권한 부여 메커니즘 구현 등)에는 초점을 맞추지 않는 티어를 생성할 수 있습니다.

API Gateway 및 Lambda를 사용하면 서버리스 로직 티어를 생성할 수 있습니다. 애플리케이션 요구 사항에 따라 AWS는 서버리스 프레젠테이션 티어(예: Amazon CloudFront 및 Amazon Simple Storage Service 사용) 및 데이터 티어(예: Amazon Aurora, Amazon DynamoDB)를 생성하는 옵션도 제공합니다.

본 백서에서는 멀티 티어 아키텍처 중 가장 많이 사용되는 3티어 웹 애플리케이션에 대해 중점적으로 다룹니다. 그러나 이 멀티 티어 패턴은 일반적인 3티어 이상의 웹 애플리케이션에도 적용할 수 있습니다.

3티어 아키텍처 개요

3티어 아키텍처는 멀티 티어 아키텍처에서 가장 널리 사용되는 구현이며 단일 프레젠테이션 티어, 로직 티어 및 데이터 티어로 구성됩니다. 다음 그림은 단순하고 일반적인 3티어 애플리케이션의 예를 보여줍니다.



3티어 애플리케이션을 위한 아키텍처 패턴

일반 3티어 아키텍처 패턴에 대해 자세히 알아볼 수 있는 훌륭한 온라인 리소스가 많이 있습니다. 본 백서에서는 Amazon API Gateway 및 AWS Lambda를 사용하는 이 아키텍처의 특정 구현 패턴을 중점적으로 다룹니다.

서버리스 로직 티어

3티어 아키텍처의 로직 티어는 애플리케이션의 브레인을 나타냅니다. 여기에서 Amazon API Gateway 및 AWS Lambda를 사용하는 것이 기존의 서버 기반 구현과 비교할 때 가장 큰 영향을 미칠 수 있습니다. 이 두 서비스의 기능을 사용하면 가용성, 확장성 및 보안성이 뛰어난 서버리스 애플리케이션을 구축할 수 있습니다. 기존 모델에서는 애플리케이션에 수천 대의 서버가 필요할 수 있습니다. 그러나 Amazon API Gateway 및 AWS Lambda를 사용하면 용량에 관계없이 서버를 관리할 책임이 없습니다. 또한 이러한 관리형 서비스를 함께 사용하면 다음과 같은 이점을 얻을 수 있습니다.

- AWS Lambda:
  - OS를 선택, 보안, 패치 또는 관리할 필요 없음
  - 적절한 크기, 모니터링 또는 확장이 가능한 서버 없음
  - 프로비저닝 초과로 인한 비용 위험 감소
  - 프로비저닝 부족으로 인한 성능 위험 감소

- Amazon API Gateway:
  - API 배포, 모니터링 및 보안을 위한 간소화된 메커니즘
  - 캐싱 및 콘텐츠 전송을 통한 API 성능 개선

AWS Lambda

AWS Lambda는 프로비저닝, 관리 또는 지원 없이 지원되는 모든 언어(Node.js, Python, Ruby, Java, Go, .NET, 자세한 내용은 Lambda FAQ 참조)로 임의 코드 함수를 실행할 수 있는 컴퓨팅 서비스입니다. Lambda 함수는 격리된 관리형 컨테이너에서 실행되며 이벤트 소스라고 하는 AWS가 제공하는 여러 프로그래밍 방식 트리거 중 하나가 될 수 있는 이벤트에 대한 응답으로 시작됩니다. 모든 이벤트 소스는 Lambda FAQ를 참조하세요.

널리 사용되는 Lambda의 사용 사례는 Amazon S3에 저장된 파일 처리 또는 Amazon Kinesis의 스트리밍 데이터 레코드와 같은 이벤트 기반 데이터 처리 워크플로를 중심으로 이루어집니다. Amazon API Gateway와 함께 사용하는 경우 Lambda 함수는 일반적인 웹 서비스의 기능을 수행합니다. 클라이언트 HTTPS 요청에 대한 응답으로 코드를 시작하고 API Gateway는 로직 티어의 프런트 도어 역할을 하며 AWS Lambda는 애플리케이션 코드를 호출합니다.

여기에는 비즈니스 로직이 적용되므로 서버가 필요하지 않습니다.

Lambda를 사용하려면 이벤트에 의해 시작될 때 실행되는 핸들러라는 코드 함수를 작성해야 합니다. API Gateway와 함께 Lambda를 사용하려면 API에 대한 HTTPS 요청이 발생할 때 핸들러 함수를 시작하도록 API Gateway를 구성할 수 있습니다. 서버리스 멀티 티어 아키텍처의 경우 API Gateway에서 생성하는 각 API는 필요한 비즈니스 로직을 호출하는 Lambda 함수(및 내부 핸들러)와 통합됩니다.

AWS Lambda 함수를 사용하여 로직 티어를 구성하면 애플리케이션 기능을 노출하기 위해 원하는 수준의 세분성을 정의할 수 있습니다(API당 Lambda 함수 1개 또는 API 메서드당 Lambda 함수 1개). Lambda 함수 내에서 핸들러는 다른 모든 종속성(예: 코드, 라이브러리, 기본 바이너리 및 외부 웹 서비스와 함께 업로드한 다른 메서드) 또는 다른 Lambda 함수에 연결할 수 있습니다.

Lambda 함수를 생성 또는 업데이트하려면 Amazon S3 버킷에 zip 파일의 Lambda 배포 패키지로 코드를 업로드하거나 모든 종속성과 함께 컨테이너 이미지로 코드를 패키징해야 합니다. 함수는 AWS 관리 콘솔, AWS Command Line Interface(AWS CLI) 실행 또는 AWS CloudFormation, AWS Serverless Application Model(AWS SAM) 또는 AWS Cloud Development Kit (AWS CDK)와 같은 프레임워크 또는 코드형 인프라 템플릿으로 실행하는 것과 같은 다양한 배포 방법을 사용할 수 있습니다. 이러한 방법을 사용하여 함수를 생성할 때 배포 패키지 내에서 요청 핸들러로 작동할 메서드를 지정합니다. 여러 Lambda 함수 정의에 대해 동일한 배포 패키지를 재사용할 수 있습니다. 여기서 각 Lambda 함수는 동일한 배포 패키지 내에서 고유한 핸들러를 가질 수 있습니다.

Lambda 보안

Lambda 함수를 실행하려면 AWS Identity and Access Management(IAM) 정책에서 허용하는 이벤트 또는 서비스에 의해 호출되어야 합니다. IAM 정책을 사용하면 정의한 API Gateway 리소스에서 호출하지 않는 한 전혀 시작할 수 없는 Lambda 함수를 생성할 수 있습니다. 이러한 정책은 다양한 AWS 서비스에서 리소스 기반 정책을 사용하여 정의할 수 있습니다.

각 Lambda 함수는 Lambda 함수가 배포될 때 할당된 IAM 역할을 가정합니다. 이 IAM 역할은 Lambda 함수가 상호 작용할 수 있는 다른 AWS 서비스 및 리소스를 정의합니다(예: Amazon DynamoDB Amazon S3). Lambda 함수의 컨텍스트에서 이를 실행 역할이라고 합니다.

Lambda 함수 내부에 민감한 정보를 저장하지 마세요. IAM은 Lambda 실행 역할을 통해 AWS 서비스에 대한 액세스를 처리합니다. Lambda 함수 내부에서 다른 자격 증명(예: 데이터베이스 자격 증명 및 API 키)에 액세스해야 하는 경우 환경 변수와 함께 AWS Key Management Service(AWS KMS)를 사용하거나 AWS Secrets Manager와 같은 서비스를 사용하여 사용하지 않을 때 이 정보를 안전하게 보관할 수 있습니다.

규모를 확장해도 우수한 성능

Amazon Elastic Container Registry(Amazon ECR) 또는 Amazon S3에 업로드된 zip 파일에서 컨테이너 이미지로 가져온 코드는 AWS에서 관리하는 격리된 환경에서 실행됩니다. Lambda 함수를 확장할 필요가 없습니다. 함수에서 이벤트 알림을 수신할 때마다 AWS Lambda는 컴퓨팅 플릿 내에서 사용 가능한 용량을 찾고 사용자가 정의한 런타임, 메모리, 디스크 및 제한 시간 구성으로 코드를 실행할 수 있습니다. 이 패턴을 사용하면 AWS가 필요한 만큼 함수의 복사본을 시작할 수 있습니다.

Lambda 기반 로직 티어는 항상 고객 요구 사항에 적합한 크기입니다. Lambda 종량제 요금제와 결합된 관리형 크기 조정 및 동시 코드 시작을 통해 트래픽 급증을 빠르게 흡수하는 기능을 사용하면 유휴 컴퓨팅 용량에 대해 비용을 지불하지 않으면서 항상 고객 요청을 충족할 수 있습니다.

서버리스 배포 및 관리

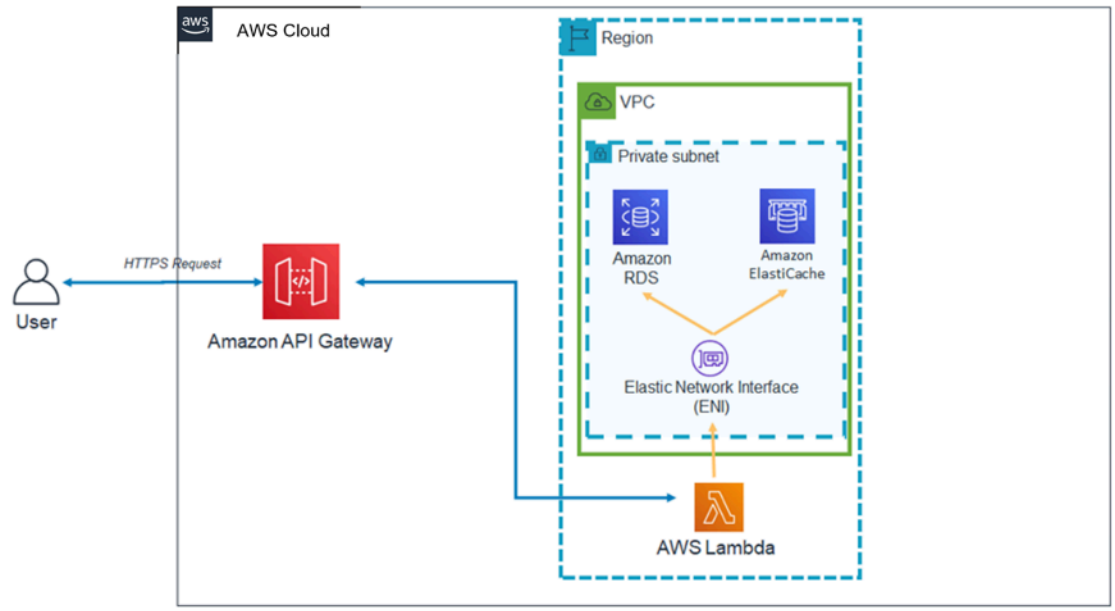
Lambda 함수를 배포하고 관리하려면 다음을 포함하는 오픈 소스 프레임워크인 AWS Serverless Application Model(AWS SAM)을 사용합니다.

- AWS SAM 템플릿 사양 - 간소화된 업로드 및 배포를 위해 함수를 정의하고 해당 환경, 권한, 구성 및 이벤트를 설명하는 데 사용되는 구문입니다.
- AWS SAM CLI - SAM 템플릿 구문을 확인하고, 로컬에서 함수를 호출하고, Lambda 함수를 디버그하고, 배포 패키지 함수를 사용할 수 있게 해주는 명령입니다.

소프트웨어 개발 프레임워크인 AWS CDK를 사용할 경우 프로그래밍 언어를 사용하여 클라우드 인프라를 정의하고 CloudFormation을 통해 프로비저닝할 수 있습니다. CDK는 AWS 리소스를 정의하는 명령적 방법을 제공하는 반면 AWS SAM은 선언적 방법을 제공합니다.

일반적으로 Lambda 함수를 배포할 때 할당된 IAM 역할에 의해 정의된 권한으로 호출되며 인터넷 연결 엔드포인트에 도달할 수 있습니다. 로직 티어의 핵심인 AWS Lambda는 데이터 티어와 직접 통합되는 구성 요소입니다. 데이터 티어에 민감한 비즈니스 또는 사용자 정보가 포함된 경우 이 데이터 티어가 프라이빗 서브넷에서 적절하게 격리되었는지 확인하는 것이 중요합니다.

Lambda 함수가 프라이빗 데이터베이스 인스턴스와 같이 공개적으로 노출할 수 없는 리소스에 액세스하도록 하려면 AWS 계정의 Virtual Private Cloud(VPC)에 있는 프라이빗 서브넷에 연결하도록 Lambda 함수를 구성할 수 있습니다. 함수를 VPC에 연결할 때 Λ는 함수의 VPC 구성에서 각 서브넷에 대한 탄력적 네트워크 인터페이스를 생성하고 탄력적 네트워크 인터페이스는 내부 리소스에 비공개로 액세스하는 데 사용됩니다.



VPC 내부의 Lambda 아키텍처 패턴

VPC와 함께 Lambda를 사용하면 비즈니스 로직이 의존하는 데이터베이스 및 기타 스토리지 미디어를 인터넷을 통해 액세스할 수 없게 만들 수 있습니다. 또한 VPC는 사용자가 정의한 API와 작성한 Lambda 코드 함수를 통해서만 인터넷의 데이터와 상호 작용할 수 있도록 보장합니다.

Amazon API Gateway

Amazon API Gateway는 어떤 규모에서든 개발자가 API를 생성, 게시, 유지 관리, 모니터링 및 보안할 수 있게 해주는 완전관리형 서비스입니다.

클라이언트(즉, 프레젠테이션 티어)는 표준 HTTPS 요청을 사용하여 API Gateway를 통해 노출된 API와 통합됩니다. API Gateway를 통해 서비스 지향 멀티 티어 아키텍처에 노출된 API의 적용 가능성은 애플리케이션 기능의 개별 부분을 분리하고 REST 엔드포인트를 통해 이 기능을 노출하는 기능입니다. Amazon API Gateway에는 로직 티어에 강력한 기능을 추가할 수 있는 특정 기능과 품질이 있습니다.

AWS Lambda와의 통합

Amazon API Gateway는 REST 및 HTTP 유형의 API를 모두 지원합니다. API Gateway API는 리소스와 메서드로 구성됩니다. 리소스란 앱이 리소스 경로를 통해 액세스할 수 있는 논리적 엔터티입니다(예: `/tickets`). 메서드는 API 리소스에 제출된 API 요청에 해당합니다(예: `GET /tickets`). API Gateway를 사용하면 Lambda 함수로 각 메서드를 지원할 수 있습니다. 즉, API Gateway에 노출된 HTTPS 엔드포인트를 통해 API를 호출하면 API Gateway가 Lambda 함수를 호출합니다.

프록시 통합 및 비프록시 통합을 사용하여 API Gateway와 Lambda 함수를 연결할 수 있습니다.

프록시 통합

프록시 통합에서는 전체 클라이언트 HTTPS 요청이 있는 그대로 Lambda 함수로 전송됩니다. API Gateway는 전체 클라이언트 요청을 Lambda 핸들러 함수의 이벤트 파라미터로 전달하고 Lambda 함수의 출력은 클라이언트에 직접 반환됩니다(상태 코드, 헤더 등 포함).

비프록시 통합

비프록시 통합에서는 클라이언트 요청의 파라미터, 헤더 및 본문이 Lambda 핸들러 함수의 이벤트 파라미터로 전달되는 방식을 구성합니다. 또한 Lambda 출력이 사용자에게 다시 변환되는 방식을 구성합니다.

> **Note**
>
> API Gateway는 또한 모의 통합(초기 애플리케이션 개발에 유용)과 같은 AWS Lambda 외부의 추가 서버리스 리소스에 프록시하고 S3 객체에 직접 프록시할 수 있습니다.

리전 간 안정적인 API 성능

Amazon API Gateway의 각 배포에는 내부의 Amazon CloudFront 배포가 포함됩니다. CloudFront는 API를 사용하는 클라이언트의 연결 지점으로 Amazon의 글로벌 엣지 로케이션 네트워크를 사용하는 콘텐츠 전송 서비스입니다. 이는 API의 응답 대기 시간을 줄이는 데 도움이 됩니다. Amazon CloudFront는 전 세계의 여러 엣지 로케이션을 사용하여 분산 서비스 거부(DDoS) 공격 시나리오에 대처할 수 있는 기능도 제공합니다. 자세한 내용은 DDoS 복원성에 대한 AWS 모범 사례 백서를 검토하세요.

API Gateway를 사용하여 선택적으로 인 메모리 캐시에 응답을 저장하면 특정 API 요청의 성능을 향상시킬 수 있습니다. 이 접근 방식은 반복되는 API 요청에 대한 성능상의 이점을 제공할 뿐만 아니라 Lambda 함수가 호출되는 횟수를 줄여 전체 비용을 절감할 수 있습니다.

기본 제공 기능으로 혁신을 장려하고 오버헤드 감소

새로운 애플리케이션을 구축하기 위한 개발 비용은 투자입니다. API Gateway를 사용하면 특정 개발 작업에 필요한 시간을 단축하고 총 개발 비용을 절감할 수 있어 조직이 보다 자유롭게 실험하고 혁신할 수 있습니다.

초기 애플리케이션 개발 단계에서 로깅 및 지표 수집 구현은 새 애플리케이션을 더 빨리 제공하기 위해 종종 무시됩니다. 이는 이러한 기능을 프로덕션 환경에서 실행 중인 애플리케이션에 배포할 때 기술적 부채 및 운영 위험으로 이어질 수 있습니다. Amazon API Gateway는 API Gateway의 원시 데이터를 수집하고 API 실행 모니터링을 위해 거의 실시간으로 읽을 수 있는 지표로 처리하는 Amazon CloudWatch와 원활하게 통합됩니다. 또한 API Gateway는 구성 가능한 보고서를 통한 액세스 로깅과 디버깅을 위한 AWS X-Ray 추적을 지원합니다. 이러한 각 기능은 코드를 작성할 필요가 없으며 핵심 비즈니스 로직에 대한 위험 없이 프로덕션 환경에서 실행되는 애플리케이션에서 조정할 수 있습니다.

애플리케이션의 전체 수명을 알 수 없거나 수명이 짧은 것으로 알려져 있을 수 있습니다. API Gateway가 제공하는 관리형 기능이 이미 시작 지점에 포함되어 있고 API가 요청을 받기 시작한 후에만 인프라 비용이 발생하는 경우 이러한 애플리케이션 구축을 위한 비즈니스 사례를 더 쉽게 만들 수 있습니다. 자세한 내용은 Amazon API Gateway 요금을 참조하세요.

빠른 반복, 민첩성 유지

Amazon API Gateway 및 AWS Lambda를 사용하여 API의 로직 티어를 구축하면 API 배포 및 버전 관리를 간소화하여 사용자 기반의 변화하는 요구 사항에 신속하게 적응할 수 있습니다.

스테이지 배포

API Gateway에서 API를 배포하는 경우 배포를 API Gateway 단계와 연결해야 합니다. 각 단계는 API의 스냅샷이며 클라이언트 앱에서 호출할 수 있습니다. 이 규칙을 사용하면 dev, test, stage 또는 prod 단계에 앱을 쉽게 배포하고 단계 간에 배포를 이동할 수 있습니다. API를 단계에 배포할 때마다 필요한 경우 되돌릴 수 있는 다른 버전의 API를 생성합니다. 이러한 기능을 사용하면 새로운 기능이 별도의 API 버전으로 출시되는 동안 기존 기능과 클라이언트 종속성에 영향을 주지 않고 계속 사용할 수 있습니다.

Lambda와의 분리 통합

API Gateway의 API와 Lambda 함수 간의 통합은 API Gateway 단계 변수와 Lambda 함수 별칭을 사용하여 분리할 수 있습니다. 이렇게 하면 API 배포가 간소화되고 속도가 빨라집니다. API에서 Lambda 함수 이름 또는 별칭을 직접 구성하는 대신 Lambda 함수의 특정 별칭을 가리킬 수 있는 API의 단계 변수를 구성할 수 있습니다. 배포 중에 Lambda 함수 별칭을 가리키도록 단계 변수 값을 변경하면 API가 특정 단계의 Lambda 별칭 뒤에서 Lambda 함수 버전을 실행합니다.

카나리 릴리스 배포

카나리 릴리스란 API의 새 버전을 테스트 목적으로 배포하고 같은 단계에서의 정상 작동을 위해 기본 버전은 프로덕션 릴리스로 배포하는 소프트웨어 개발 전략입니다. 카나리 릴리스 배포에서 전체 API

트래픽은 사전 구성된 비율로 프로덕션 릴리스와 카나리 릴리스로 임의로 분할됩니다. API Gateway의 API를 카나리 릴리스 배포용으로 구성하여 제한된 사용자 집합으로 새로운 기능을 테스트할 수 있습니다.

사용자 지정 도메인 이름

API Gateway에서 제공하는 URL 대신 직관적이고 비즈니스 친화적인 URL 이름을 API에 제공할 수 있습니다. API Gateway는 API에 대한 사용자 지정 도메인을 구성하는 기능을 제공합니다. 사용자 지정 도메인 이름을 사용하면 API의 호스트 이름을 설정하고 다중 수준 기본 경로(예: `myservice`, `myservice/cat/v1` 또는 `myservice/dog/v2`)를 선택하여 대체 URL을 API에 매핑할 수 있습니다.

API 보안 우선 순위

모든 애플리케이션은 승인된 클라이언트만 API 리소스에 액세스할 수 있도록 해야 합니다. 멀티 티어 애플리케이션을 설계할 때 Amazon API Gateway가 로직 티어 보안에 기여하는 여러 가지 방법을 활용할 수 있습니다.

전송 보안

API에 대한 모든 요청은 HTTPS를 통해 전송 중 암호화를 활성화할 수 있습니다.

API Gateway는 내장형 SSL/TLS 인증서를 제공합니다. 퍼블릭 API에 대해 사용자 지정 도메인 이름 옵션을 사용하는 경우 AWS Certificate Manager를 사용하여 자체 SSL/TLS 인증서를 제공할 수 있습니다. API Gateway는 상호 TLS(mTLS) 인증도 지원합니다. 상호 TLS는 API의 보안을 향상하고, 클라이언트 스푸핑, 중간자 공격 등의 공격으로부터 데이터를 보호하는 데 도움이 됩니다.

API 권한 부여

API의 일부로 생성하는 각 리소스/메서드 조합에는 AWS Identity and Access Management(IAM) 정책에서 참조할 수 있는 고유한 Amazon 리소스 이름(ARN)이 부여됩니다.

API Gateway에서 API에 권한 부여를 추가하는 세 가지 일반적인 방법이 있습니다.

- IAM 역할 및 정책: 클라이언트는 API 액세스를 위해 AWS 서명 버전 4(SigV4) 권한 부여 및 IAM 정책을 사용합니다. 동일한 자격 증명으로 필요에 따라 다른 AWS 서비스 및 리소스(예: Amazon S3 버킷 또는 Amazon DynamoDB 테이블)에 대한 액세스를 제한하거나 허용할 수 있습니다.
- Amazon Cognito 사용자 풀: 클라이언트는 Amazon Cognito 사용자 풀을 통해 로그인하고 요청의 권한 부여 헤더에 포함된 토큰을 얻습니다.
- Lambda 권한 부여자: 소유자 토큰 전략(예: OAuth 및 SAML)을 사용하거나 요청 파라미터를 사용하여 사용자를 식별하는 사용자 지정 권한 부여 체계를 구현하는 Lambda 함수를 정의합니다.

액세스 제한

API Gateway는 API 키 생성 및 구성 가능한 사용 계획과 이러한 키의 연결을 지원합니다. CloudWatch를 사용하여 API 키 사용량을 모니터링할 수 있습니다.

API Gateway는 API의 각 메서드에 대한 제한, 속도 제한 및 버스트 속도 제한을 지원합니다.

프라이빗 API

API Gateway를 사용하면 인터페이스 VPC 엔드포인트를 통해 Amazon VPC의 Virtual Private Cloud에서만 액세스할 수 있는 프라이빗 REST API를 생성할 수 있습니다. VPC에서 생성하는 엔드포인트 네트워크 인터페이스입니다.

여러 AWS 계정은 물론 선택한 VPC 및 VPC 엔드포인트에서 API에 액세스하는 것을 리소스 정책으로 허용하거나 거부할 수 있습니다. 각 엔드포인트를 사용하여 여러 개의 프라이빗 API에 액세스할 수 있습니다. 또한 온프레미스 네트워크에서 AWS Direct Connect를 사용하여 Amazon VPC에 연결한 다음 그 연결을 통해 프라이빗 API에 액세스할 수도 있습니다.

어떤 경우에든 프라이빗 API로 가는 트래픽은 안전한 연결을 사용하고, Amazon 네트워크를 벗어나지 않으며, 퍼블릭 인터넷과 격리됩니다.

AWS WAF를 사용한 방화벽 보호

인터넷 연결 API는 악의적인 공격에 취약합니다. AWS WAF는 이러한 공격으로부터 API를 보호하는 데 도움이 되는 웹 애플리케이션 방화벽입니다. SQL 명령어 삽입 및 크로스 사이트 스크립팅 공격과 같은 일반적인 웹 도용으로부터 API를 보호합니다. API Gateway와 함께 AWS WAF를 함께 사용하여 API를 보호할 수 있습니다.

데이터 티어

AWS Lambda를 로직 티어로 사용하면 데이터 티어에서 사용할 수 있는 데이터 스토리지 옵션이 제한되지 않습니다. Lambda 함수는 Lambda 배포 패키지에 적절한 데이터베이스 드라이버를 포함하여 모든 데이터 스토리지 옵션에 연결하고 IAM 역할 기반 액세스 또는 암호화된 자격 증명(AWS KMS 또는 AWS Secrets Manager를 통해)을 사용합니다.

애플리케이션에 대한 데이터 스토어를 선택하는 것은 애플리케이션 요구 사항에 따라 크게 달라집니다. AWS는 애플리케이션의 데이터 티어를 구성하는 데 사용할 수 있는 다양한 서버리스 및 비서버리스 데이터 스토어를 제공합니다.

서버리스 데이터 스토리지 옵션

Amazon S3은 업계 최고의 확장성, 데이터 가용성, 보안 및 성능을 제공하는 객체 스토리지 서비스입니다.

Amazon Aurora는 MySQL 및 PostgreSQL과 호환되는 클라우드용 관계형 데이터베이스로서 기존 엔터프라이즈 데이터베이스의 성능 및 가용성과 오픈 소스 데이터베이스의 단순성 및 비용 효율성을 결합한 것입니다. Aurora는 서버리스 및 기존 사용 모델을 모두 제공합니다.

Amazon DynamoDB는 모든 규모에서 10밀리초 미만의 성능을 제공하는 키–값 및 문서 데이터베이스입니다. 인터넷 규모의 애플리케이션을 위한 보안, 백업 및 복원, 인 메모리 캐싱이 내장된 탁월한 내구력의 완전관리형, 서버리스, 다중 리전, 다중 마스터 데이터베이스입니다.

Amazon Timestream은 IoT 및 운영 애플리케이션을 위한 빠르고 확장 가능한 완전관리형 시계열 데이터베이스 서비스로서, 관계형 데이터베이스의 1/10의 비용으로 하루에 수조 건의 이벤트를 간단히 저장하고 분석할 수 있도록 지원합니다. IoT 디바이스, IT 시스템 및 산업용 스마트 머신이 늘어나면서, 시간에 따른 변화를 측정하는 시계열 데이터는 가장 빠르게 증가하는 데이터 유형 중 하나가 되었습니다.

Amazon Quantum Ledger Database(Amazon QLDB)는 완전관리형 원장 데이터베이스로, 신뢰할 수 있는 중앙 기관에서 소유하는 투명하고 변경 불가능하며 암호화 방식으로 검증 가능한 트랜잭션 로그를 제공합니다. Amazon QLDB는 모든 애플리케이션 데이터 변경 내용을 추적하며 시간이 지나도 완전하고 검증 가능한 변경 내역을 유지 관리합니다.

Amazon Keyspaces(Apache Cassandra용)는 확장성과 가용성이 뛰어나며 관리형 Apache Cassandra와 호환되는 데이터베이스 서비스입니다. Amazon Keyspaces를 사용하면 현재 사용 중인 것과 동일한 Cassandra 애플리케이션 코드 및 개발자 도구를 사용하여 AWS에서 Cassandra 워크로드를 실행할 수

있습니다. 서버를 프로비저닝, 패치 또는 관리할 필요가 없으며 소프트웨어를 설치, 유지 관리 또는 운영할 필요도 없습니다. Amazon Keyspaces는 서버리스이므로 사용하는 리소스에 대해서만 비용을 지불하며 서비스가 애플리케이션 트래픽에 따라 자동으로 테이블 크기를 조정할 수 있습니다.

Amazon Elastic File System(Amazon EFS)은 스토리지를 프로비저닝하거나 관리하지 않고도 파일 데이터를 공유할 수 있게 해주는 간단하고 한 번만 설정하면 되는 탄력적 서버리스 파일 시스템을 제공합니다. AWS 클라우드 서비스 및 온프레미스 리소스와 함께 사용할 수 있으며, 애플리케이션을 중단하지 않고 온디맨드 방식으로 페타바이트 규모의 확장이 가능하도록 구축되었습니다. Amazon EFS를 사용하면 파일을 추가 및 제거할 때 파일 시스템을 자동으로 확장 및 축소할 수 있으므로, 확장을 위해 용량을 프로비저닝하고 관리할 필요가 없습니다. Amazon EFS는 Lambda 함수로 탑재할 수 있으므로 API에 대한 실행 가능한 파일 스토리지 옵션으로 사용할 수 있습니다.

비서버리스 데이터 스토리지 옵션

Amazon Relational Database Service(Amazon RDS)는 관계형 데이터베이스를 보다 쉽게 설정, 운영 및 확장할 수 있게 해주는 관리형 웹 서비스로서, 사용 가능한 엔진(Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, Microsoft SQL Server)을 사용하고 메모리, 성능 또는 I/O에 최적화된 여러 데이터베이스 인스턴스 유형에서 실행됩니다.

Amazon Redshift는 클라우드에서 제공되는 페타바이트 규모의 완전관리형 데이터 웨어하우스 서비스입니다.

Amazon ElastiCache는 Redis 또는 Memcached의 완전관리형 배포입니다. 널리 사용되는 오픈 소스 호환 가능한 인 메모리 데이터 스토어를 원활하게 배포, 실행 및 확장합니다.

Amazon Neptune은 빠르고 안정적인 완전관리형 그래프 데이터베이스 서비스로, 상호연결성이 높은 데이터 집합을 활용하는 애플리케이션을 손쉽게 구축 및 운영할 수 있습니다. Neptune은 널리 사용되는 그래프 모델(속성 그래프 및 W3C Resource Description Framework(RDF))과 해당 쿼리 언어를 지원하므로 상호연결성이 높은 데이터 집합을 효율적으로 탐색하는 쿼리를 쉽게 구축할 수 있습니다.

Amazon DocumentDB(MongoDB 호환)는 MongoDB 워크로드를 지원하는 빠르고, 확장 가능하며, 가용성이 높은 완전관리형 문서 데이터베이스 서비스입니다.

마지막으로 Amazon EC2에서 독립적으로 실행되는 데이터 스토어를 멀티 티어 애플리케이션의 데이터 티어로 사용할 수도 있습니다.

프레젠테이션 티어

프레젠테이션 티어는 인터넷을 통해 노출되는 API Gateway REST 엔드포인트를 통해 로직 티어와 상호 작용합니다. 모든 HTTPS 지원 클라이언트 또는 디바이스는 이러한 엔드포인트와 통신할 수 있으므로 프레젠테이션 티어에 다양한 형태(데스크톱 애플리케이션, 모바일 앱, 웹 페이지, IoT 디바이스 등)를 사용할 수 있는 유연성을 제공합니다. 요구 사항에 따라 프레젠테이션 티어에서 다음 AWS 서버리스 제품을 사용할 수 있습니다. 모든 HTTPS 지원 클라이언트 또는 디바이스는 이러한 엔드포인트와 통신할 수 있으므로 프레젠테이션 티어에 다양한 형태(데스크톱 애플리케이션, 모바일 앱, 웹 페이지, IoT 디바이스 등)를 사용할 수 있는 유연성을 제공합니다. 요구 사항에 따라 프레젠테이션 티어는 다음 AWS 서버리스 제품을 사용할 수 있습니다.

- Amazon Cognito - 웹 및 모바일 앱에 사용자 가입, 로그인 및 액세스 제어를 빠르고 효율적으로 추가할 수 있는 서버리스 사용자 자격 증명 및 데이터 동기화 서비스입니다. Amazon Cognito는 수백만 명의 사용자로 확장되며 Facebook, Google, Amazon과 같은 소셜 자격 증명 공급자와 SAML 2.0을 통한 엔터프라이즈 자격 증명 공급자를 통한 로그인을 지원합니다.
- CloudFront가 포함된 Amazon S3 - 웹 서버를 프로비저닝할 필요 없이 S3 버킷에서 단일 페이지 애플리케이션과 같은 정적 웹 사이트를 직접 제공할 수 있습니다. CloudFront를 관리형 콘텐츠 전송 네트워크(CDN)로 사용하여 성능을 개선하고 관리형 또는 사용자 지정 인증서를 사용하여 SSL/TL을 활성화할 수 있습니다.

AWS Amplify는 모바일 및 프런트 엔드 웹 개발자가 AWS에서 제공하는 확장 가능한 풀 스택 애플리케이션을 구축하도록 지원하는 함께 혹은 단독으로 사용 가능한 도구 및 서비스 집합입니다. Amplify는 전세계 수백 개의 상호 접속 위치(POP)가 포함된 Amazon의 안정적인 CDN과 애플리케이션 릴리스 주기를 가속화하는 기본 제공 CI/CD 워크플로를 통해 정적 웹 애플리케이션의 글로벌 배포 및 호스팅을 지원하는 완전관리형 서비스입니다. Amplify는 JavaScript, React, Angular, Vue, Next.js 등의 널리 사용되는 웹 프레임워크와 Android, iOS, React Native, Ionic, Flutter 등의 모바일 플랫폼을 지원합니다. 네트워킹 구성 및 애플리케이션 요구 사항에 따라 API Gateway API가 CORS(Cross-Origin Resource Sharing)를 준수하도록 활성화해야 할 수 있습니다. CORS 규정 준수를 통해 웹 브라우저는 정적 웹 페이지 내에서 API를 직접 호출할 수 있습니다.

CloudFront를 사용하여 웹 사이트를 배포하면 애플리케이션에 연결할 수 있는 CloudFront 도메인 이름이 제공됩니다(예: `d2d47p2vcczkh2.cloudfront.net`). Amazon Route 53을 사용하여 도메인 이름을 등록하고 CloudFront 배포로 보내거나 이미 소유한 도메인 이름을 CloudFront 배포로 보낼 수 있습니다. 이를 통해 사용자는 친숙한 도메인 이름을 사용하여 사이트에 액세스할 수 있습니다. Route 53을 사용하여 사용자 지정 도메인 이름을 API Gateway 배포에 할당할 수도 있습니다. 이렇게 하면 사용자가 친숙한 도메인 이름을 사용하여 API를 호출할 수 있습니다.

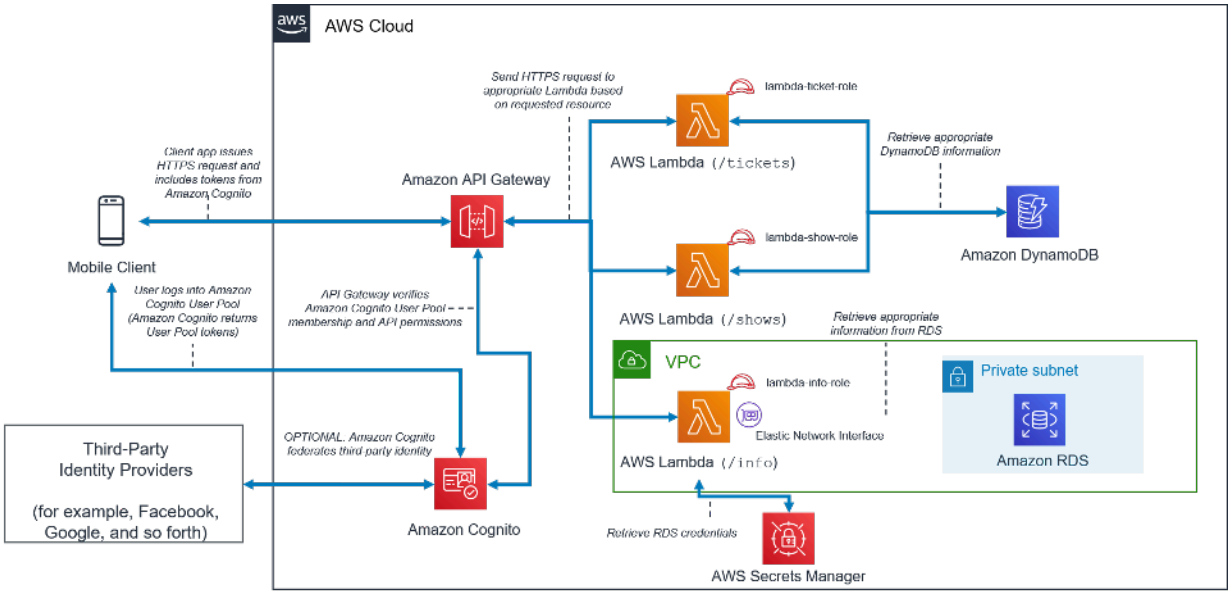
샘플 아키텍처 패턴

API Gateway 및 AWS Lambda를 로직 티어로 사용하여 널리 사용되는 아키텍처 패턴을 구현할 수 있습니다. 본 백서에는 AWS Lambda 기반 로직 티어를 활용하는 가장 널리 사용되는 아키텍처 패턴이 포함되어 있습니다.

- 모바일 백엔드 - 모바일 애플리케이션은 API Gateway 및 Lambda와 통신하여 애플리케이션 데이터에 액세스합니다. 이 패턴은 서버리스 AWS 리소스를 사용하여 프레젠테이션 티어 리소스(예: 데스크톱 클라이언트, EC2에서 실행되는 웹 서버 등)를 호스팅하지 않는 일반 HTTPS 클라이언트로 확장할 수 있습니다.
- 단일 페이지 애플리케이션 - Amazon S3 및 CloudFront에서 호스팅되는 단일 페이지 애플리케이션은 API Gateway 및 AWS Lambda와 통신하여 애플리케이션 데이터에 액세스합니다.
- 웹 애플리케이션 – 웹 애플리케이션은 비즈니스 로직에 API Gateway와 함께 AWS Lambda를 사용하는 범용 이벤트 기반 웹 애플리케이션 백엔드입니다. 또한 DynamoDB를 데이터베이스로 사용하고 Amazon Cognito를 사용자 관리에 사용합니다. 모든 정적 콘텐츠는 Amplify를 사용하여 호스팅합니다.

본 백서에서는 이 두 가지 패턴 외에도 일반 마이크로 서비스 아키텍처에 대한 Lambda 및 API Gateway의 적용 가능성에 대해 설명합니다. 마이크로서비스 아키텍처는 표준 3티어 아키텍처는 아니지만 애플리케이션 구성 요소를 분리하고 서로 통신하는 무상태 개별 기능 단위로 배포하는 일반적인 패턴입니다.

모바일 백엔드



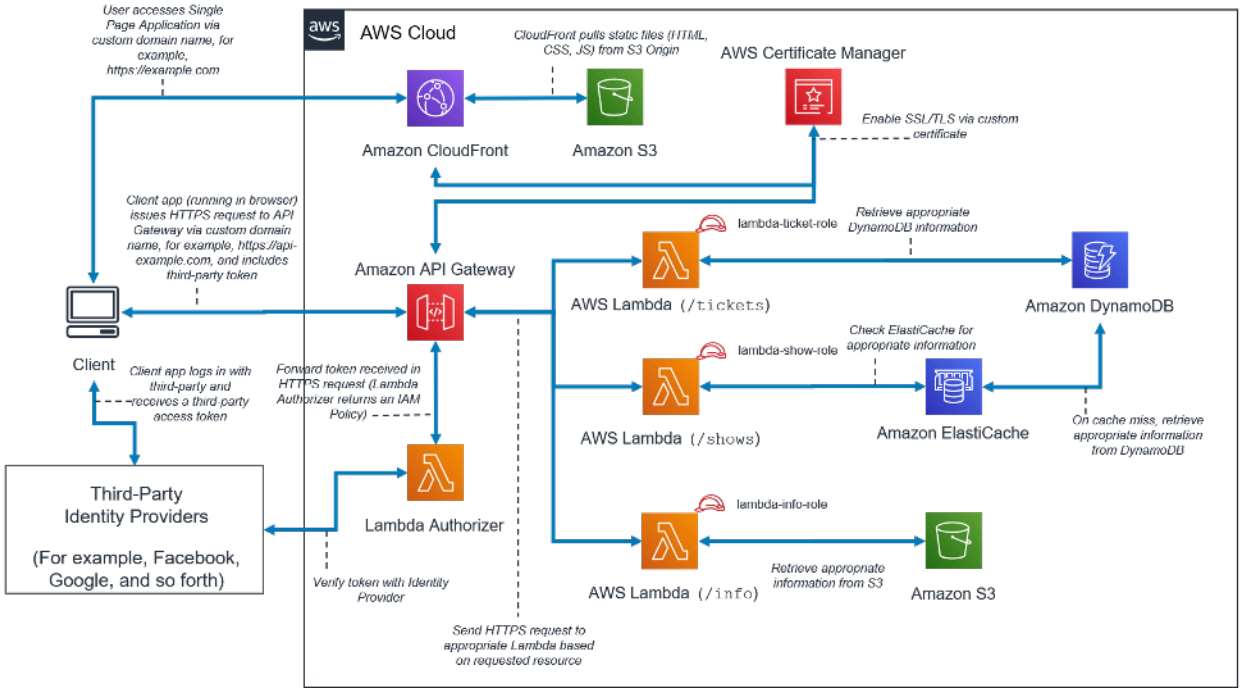
서버리스 모바일 백엔드를 위한 아키텍처 패턴

표 1 - 모바일 백엔드 티어 구성 요소

티어	구성 요소
프레젠테이션	사용자 디바이스에서 실행되는 모바일 애플리케이션입니다.
로직	AWS Lambda가 포함된 Amazon API Gateway입니다. 이 아키텍처는 세 가지 노출된 서비스(`/tickets`, `/shows`, 및 `/info`)를 보여줍니다. API Gateway 엔드포인트는 Amazon Cognito 사용자 풀에 의해 보호됩니다. 이 방법에서 사용자는 Amazon Cognito 사용자 풀(필요한 경우 페더레이션 서드 파티 사용)에 로그인하고 API Gateway 호출을 승인하는 데 사용되는 액세스 및 ID 토큰을 받습니다.

티어	구성 요소
	각 Lambda 함수에는 적절한 데이터 원본에 대한 액세스를 제공하기 위해 고유한 Identity and Access Management(IAM) 역할이 할당됩니다.
데이터	DynamoDB는 `/tickets` 및 `/shows` 서비스에 사용됩니다. Amazon RDS는 `/info` 서비스에 사용됩니다. 이 Lambda 함수는 AWS Secrets Manager에서 Amazon RDS 자격 증명을 검색하고 탄력적 네트워크 인터페이스를 사용하여 프라이빗 서브넷에 액세스합니다.

단일 페이지 애플리케이션

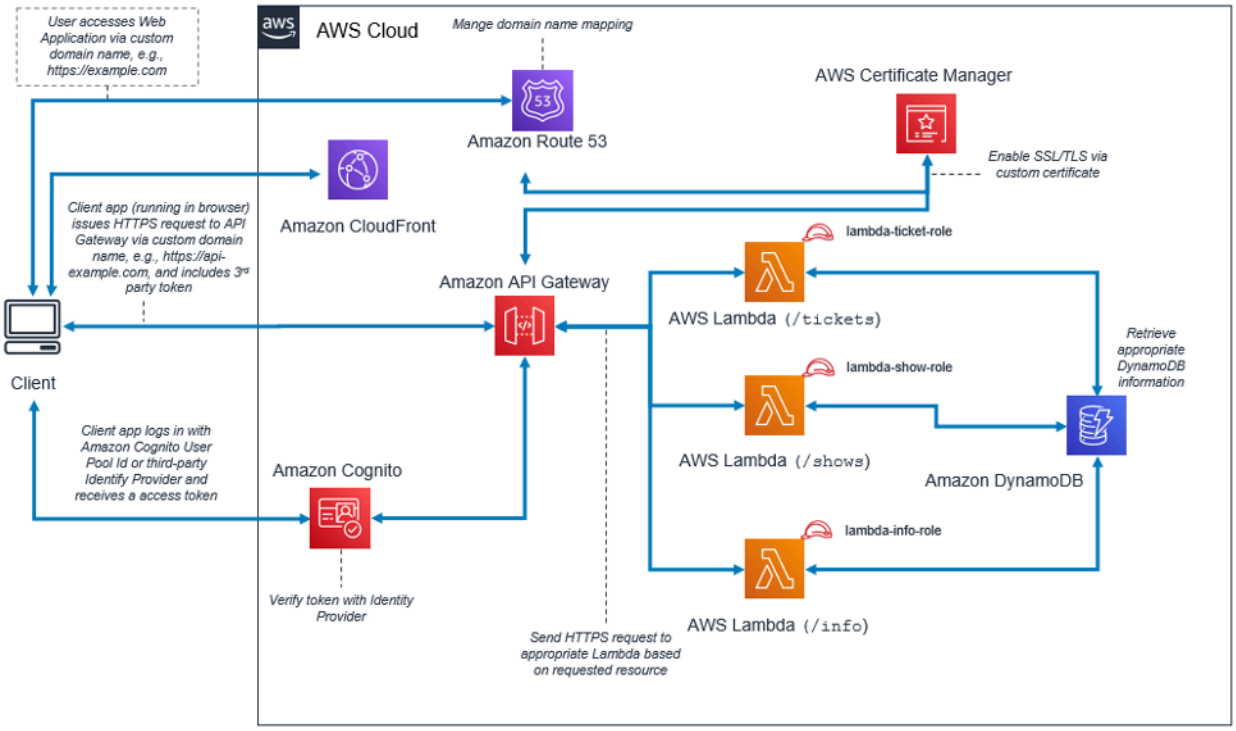


서버리스 단일 페이지 애플리케이션을 위한 아키텍처 패턴

표 2 - 단일 페이지 애플리케이션 구성 요소

티어	구성 요소
프레젠테이션	CloudFront에서 배포하고 Amazon S3에서 호스팅되는 정적 웹 사이트 콘텐츠입니다. AWS Certificate Manager에서는 사용자 지정 SSL/TLS 인증서를 사용할 수 있습니다.
로직	AWS Lambda가 포함된 API Gateway입니다. 이 아키텍처는 세 가지 노출된 서비스(`/tickets`, `/shows`, 및 `/info`)를 보여줍니다. API Gateway 엔드포인트는 Lambda 권한 부여자에 의해 보호됩니다. 이 방법에서 사용자는 서드 파티 자격 증명 공급자를 통해 로그인하고 액세스 및 ID 토큰을 얻습니다. 이러한 토큰은 API Gateway 호출에 포함되며 Lambda 권한 부여자는 이러한 토큰을 검증하고 API 시작 권한이 포함된 IAM 정책을 생성합니다. 각 Lambda 함수에는 적절한 데이터 원본에 대한 액세스를 제공하기 위해 고유한 IAM 역할이 할당됩니다.
데이터	Amazon DynamoDB는 `/tickets` 및 `/shows` 서비스에 사용됩니다. Amazon ElastiCache는 `/shows` 서비스에서 데이터베이스 성능을 개선하는 데 사용됩니다. 캐시 누락 수는 DynamoDB로 전송됩니다. Amazon S3은 `/info service`에서 사용하는 정적 콘텐츠를 호스팅하는 데 사용됩니다.

웹 애플리케이션



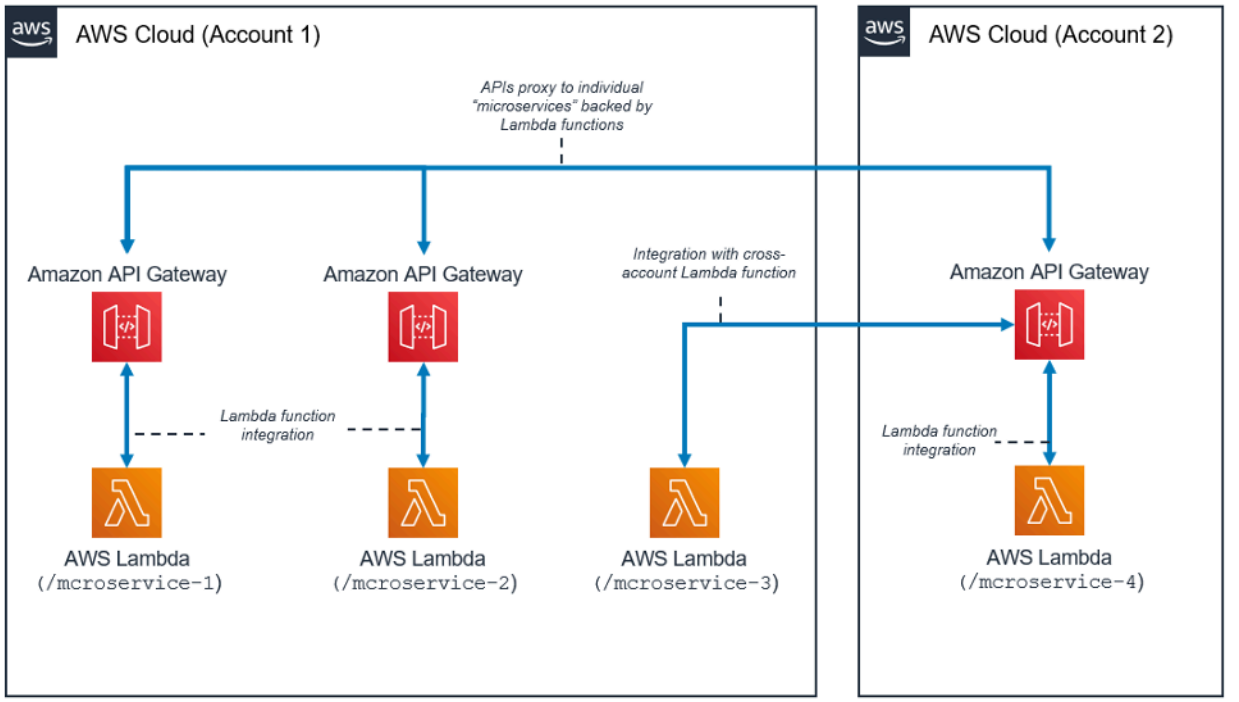
웹 애플리케이션을 위한 아키텍처 패턴

표 3 - 웹 애플리케이션 구성 요소

티어	구성 요소
프레젠테이션	프런트 엔드 애플리케이션은 create-react-app과 같은 React 유틸리티에 의해 생성되는 모든 정적 콘텐츠(HTML, CSS, JavaScript 및 이미지)입니다. Amazon CloudFront는 이러한 모든 객체를 호스팅합니다. 웹 애플리케이션을 사용하면 모든 리소스를 브라우저로 다운로드하고 브라우저에서 실행을 시작합니다. 웹 애플리케이션은 API를 호출하는 백엔드에 연결합니다.
로직	로직 계층은 API Gateway REST API가 앞에 있는 Lambda 함수를 사용하여 구축됩니다.

티어	구성 요소
	이 아키텍처는 노출된 여러 서비스를 보여줍니다. 애플리케이션의 서로 다른 측면을 각각 처리하는 여러 가지 Lambda 함수가 있습니다. Lambda 함수는 API Gateway 뒤에 있으며 API URL 경로를 사용하여 액세스할 수 있습니다. 사용자 인증은 Amazon Cognito 사용자 풀 또는 페더레이션 사용자 공급자를 사용하여 처리됩니다. API Gateway는 Amazon Cognito와의 즉시 사용 가능한 통합을 사용합니다. 클라이언트는 사용자가 인증된 후에만 JSON 웹 토큰(JWT)을 수신합니다. 이 토큰은 API 호출 시 사용해야 합니다. 각 Lambda 함수에는 적절한 데이터 원본에 대한 액세스를 제공하기 위해 고유한 IAM 역할이 할당됩니다.
데이터	이 특정 예에서는 DynamoDB가 데이터 스토리지에 사용되지만 사용 사례 및 사용 시나리오에 따라 다른 목적별 Amazon 데이터베이스 또는 스토리지 서비스를 사용할 수 있습니다.

마이크로서비스 및 Lambda



마이크로서비스 및 Lambda를 위한 아키텍처 패턴

마이크로서비스 아키텍처 패턴은 일반적인 3티어 아키텍처에 종속되지 않습니다. 그러나 널리 사용되는 이 패턴은 서버리스 리소스를 사용하여 상당한 이점을 실현할 수 있습니다.

이 아키텍처에서는 각 애플리케이션 구성 요소가 분리되어 독립적으로 배포 및 운영됩니다. Amazon API Gateway로 생성된 API와 이후에 AWS Lambda에서 시작되는 함수만 있으면 마이크로서비스를 구축할 수 있습니다. 팀은 이러한 서비스를 사용하여 원하는 세부 수준으로 환경을 분리하고 조각화할 수 있습니다.

일반적으로 마이크로서비스 환경에서는 각각의 새로운 마이크로서비스를 생성하기 위한 반복적인 오버헤드, 서버 밀도 및 활용 최적화 문제, 여러 버전의 여러 마이크로서비스를 동시에 실행하는 복잡성, 많은 개별 서비스와 통합하기 위한 클라이언트 측 코드 요구 사항의 확산 등의 어려움을 초래할 수 있습니다.

서버리스 리소스를 사용하여 마이크로서비스를 만들면 이러한 문제를 해결하기가 더 쉬워지고 어떤 경우에는 사라지기도 합니다. 서버리스 마이크로서비스 패턴은 각 후속 마이크로서비스 생성에 대한 장벽을 낮춥니다(API Gateway는 기존 API 복제 및 다른 계정의 Lambda 함수 사용도 허용). 서버 활용 최적화는 더 이상 이 패턴과 관련이 없습니다. 마지막으로 Amazon API Gateway는 프로그래밍 방식으로 생성된 클라이언트 SDK를 여러 인기 언어로 제공하여 통합 오버헤드를 줄입니다.

결론

멀티 티어 아키텍처 패턴은 유지 관리, 분리 및 확장이 간편한 애플리케이션 구성 요소를 만드는 모범 사례를 권장합니다. API Gateway에서 통합이 발생하고 AWS Lambda 내에서 계산이 발생하는 로직 티어를 생성하면 목표를 달성하기 위한 노력을 줄이면서 이러한 목표를 실현할 수 있습니다. 이러한 서비스는 클라이언트를 위한 HTTPS API 프런트 엔드와 비즈니스 로직을 적용할 수 있는 보안 환경을 제공하는 동시에 일반적인 서버 기반 인프라 관리와 관련된 오버헤드를 제거합니다.

기여자

본 문서를 작성하는 데 도움을 주신 분들입니다.

- Andrew Baird, AWS, 솔루션스 아키텍트
- Bryant Bost, AWS, ProServe 컨설턴트
- Stefano Buliani, AWS Mobile, 기술 부문 선임 제품 관리자
- Vyom Nagrani, AWS Mobile, 선임 제품 관리자
- Ajay Nair, AWS Mobile, 선임 제품 관리자
- Rahul Popat, 글로벌 솔루션스 아키텍트
- Brajendra Singh, 선임 솔루션스 아키텍트

참고 문헌

자세한 내용은 다음을 참조하세요.

- AWS 백서 및 가이드

문서 개정

본 백서의 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하세요.

update-history-change	update-history-description	update-history-date
백서 업데이트됨	새로운 서비스 기능 및 패턴에 대해 업데이트했습니다.	2021년 10월 20일
백서 업데이트됨	새로운 서비스 기능 및 패턴에 대해 업데이트했습니다.	2021년 6월 1일
백서 업데이트됨	새로운 서비스 기능에 대해 업데이트했습니다.	2019년 9월 25일
최초 게시	백서를 게시했습니다.	2015년 11월 1일

고지 사항

고객은 본 문서에 포함된 정보를 독자적으로 평가할 책임이 있습니다. 본 문서는 (a) 정보 제공만을 위한 것이며, (b) 사전 고지 없이 변경될 수 있는 현재의 AWS 제품 제공 서비스 및 사례를 보여 주며, (c) AWS 및 자회사, 공급업체 또는 라이선스 제공자로부터 어떠한 약정 또는 보증도 하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 진술 또는 조건 없이 '있는 그대로' 제공됩니다. 고객에 대한 AWS의 책임과 법적 책임은 AWS 계약서에 준하며 본 문서는 AWS와 고객 간의 계약에 포함되지 않고 계약을 변경하지도 않습니다.