AWS Serverless Multi-Tier Architectures

Amazon API Gateway 및 AWS Lambda 사용

2015년 11월

고지 사항

이 문서는 정보 제공 목적으로만 제공됩니다. 본 문서의 발행일 당시 AWS의 현재 제품 및 실행방법을 설명하며, 예고 없이 변경될 수 있습니다. 고객은 본 문서에 포함된 정보나 AWS 제품 또는 서비스의 사용을 독립적으로 평가할 책임이 있으며, 각 정보 및 제품은 명시적이든 묵시적이든 어떠한 종류의 보증 없이 "있는 그대로" 제공됩니다. 본 문서는 AWS, 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 보증, 표현, 계약 약속, 조건 또는 보증을 구성하지 않습니다. 고객에 대한 AWS의 책임 및 의무는 AWS 계약에 준거합니다. 본 문서는 AWS와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

목차

요약

본 백서는 Amazon Web Services(AWS)의 혁신이 어떻게 마이크로서비스, 모바일 백엔드, 퍼블릭 웹 사이트와 같은 인기 있는 패턴을 위한 멀티 티어 아키텍처를 설계하는 방법을 바꿀 수 있는지 보여줍니다. 아키텍트와 개발자는 이제 Amazon API Gateway와 AWS Lambda가 포함된 구현 패턴을 사용함으로써 멀티 티어 애플리케이션을 생성하고 운영 관리하는 데 필요한 개발 및 운영 주기를 단축할 수 있습니다.

서론

멀티 티어 애플리케이션(3티어 및 n티어 등)은 수십 년간 기초 아키텍처 패턴으로 사용되어 왔습니다. 멀티 티어 패턴은 주로 서로 다른 팀에서 따로 관리하고 유지되는 애플리케이션 구성 요소를 분리하여 확장하기 위해 수행할 수 있는 유용한 지침을 제공하고, 멀티 티어 애플리케이션은 주로 웹 서비스 사용에 서비스 중심 아키텍처(SOA) 접근 방식을 이용하여 구축됩니다. 이 접근 방식에서 네트워크가 티어 간의 경계 역할을 합니다. 그러나 애플리케이션의 일부로 새 웹 서비스 티어가 생성되는 등 구분되지 않는 면도 많습니다. 멀티 티어 웹 애플리케이션 내에서 작성된 상당한 코드가 패턴의 직접적인 결과 자체가 됩니다. 이러한 예로는, 티어 간에 통합하는 코드, 티어가 서로 파악하기 위해 사용하는 데이터 모델과 API를 정의하는 코드 및 티어의 통합 지점이 원치 않는 방식으로 노출되지 않도록 보장하는 보안 관련 코드가 있습니다.

API를 생성 및 관리하는 서비스인 Amazon API Gateway[1]와 임의의 코드 함수를 실행하는 서비스인 AWS Lambda[2]를 함께 사용하면 강력한 멀티 티어 애플리케이션을 손쉽게 생성할 수 있습니다.

AWS Lambda와 Amazon API Gateway의 통합으로, 사용자 정의 코드 함수를 사용자 정의 HTTPS 요청을 통해 직접 트리거할 수 있습니다. 필요한 요청 볼륨에 상관없이, API Gateway와 Lambda가 둘 다 애플리케이션의 요구를 정확히 지원할 수 있도록 자동으로 조정합니다. 결합되면 애플리케이션에 중요한 코드를 작성하지만 고가용성을 위한 설계, 클라이언트 SDK 작성, 서버/운영 체제(OS) 관리, 확장 및 클라이언트 권한 부여 메커니즘 구현과 같이 멀티 티어 아키텍처 구현의 여러 가지 다른 획일적인 측면에 중점을 두지 *않는* 애플리케이션의 티어를 생성할 수 있습니다.

최근에 AWS가 Amazon Virtual Private Cloud(Amazon VPC)[3] 내에서 실행되는 Lambda 함수를 생성할 수 있는 기능을 발표했습니다. 이 기능을 통해 API Gateway와 Lambda가 결합되어 네트워크 프라이버시가 요구되는 다양한 사용 사례를 적용할 수 있습니다. 예를 들어, 중요한 정보를 포함하는 관계형 데이터베이스와 웹 서비스를 통합해야 하는 경우가 있습니다. Lambda와 Amazon VPC가 통합됨에 따라 개발자에게 Amazon VPC의 일부로 비공개로 보안을 유지하는 백엔드의 앞에 인터넷으로 액세스 가능한 HTTPS API 세트를 고유하게 정의할 수 있는 기능이 제공되므로 Amazon API Gateway의 기능이 간접적으로 확장되었습니다. 멀티 티어 아키텍처의 각 티어 전체에서 이 강력한 패턴의 이점을 확인할 수 있습니다. 이 백서에서는 가장 일반적인 멀티 티어 아키텍처의 예로

3티어 웹 애플리케이션을 중점적으로 다룹니다. 그러나 이 멀티 티어 패턴을 일반적인 3티어 웹 애플리케이션 이상으로 적용할 수 있습니다.

3티어 아키텍처 개요

3티어 아키텍처는 사용자 지향 애플리케이션에 일반적으로 사용되는 패턴입니다. 이 아키텍처를 구성하는 티어로는 **프레젠테이션 티어, 로직 티어** 및 **데이터 티어**가 있습니다. 프레젠테이션 티어는 사용자가 웹 페이지, 모바일 앱 UI 등과 직접 상호 작용하는 구성 요소를 나타냅니다. 로직 티어에는 프레젠테이션 티어의 사용자 작업을 애플리케이션의 동작을 유도하는 기능으로 변환하는 데 필요한 코드가 포함되고, 데이터 티어는 애플리케이션과 관련된 데이터를 보관하는 스토리지 미디어(데이터베이스, 객체 스토어, 캐시, 파일 시스템 등)로 구성됩니다. 그림 1에 간단한 3티어 애플리케이션의 예가 나와 있습니다.

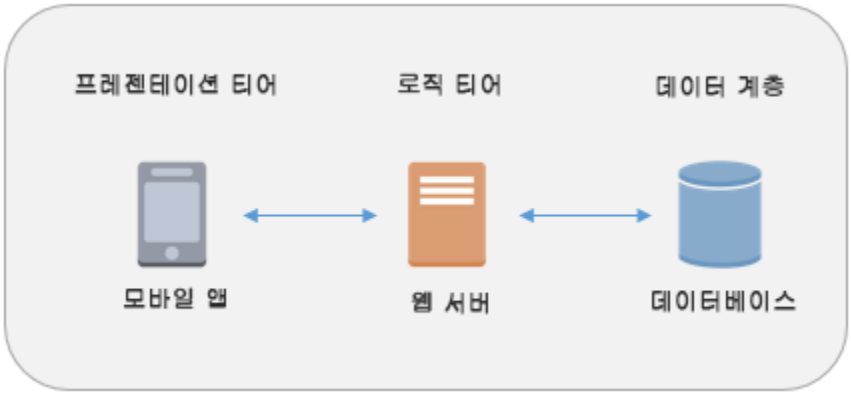

그림 1: 간단한 3티어 애플리케이션의 아키텍처 패턴

일반 3티어 아키텍처 패턴에 대해 자세히 알아 볼 수 있는 많은 유용한 리소스를 온라인에서 확인할 수 있습니다. 이 백서는 Amazon API Gateway 및 AWS Lambda를 이용하는 이 아키텍처에 대한 특정 구현 패턴에 초점을 맞추고 있습니다.

서버가 없는 로직 티어

3티어 아키텍처 중 로직 티어는 애플리케이션의 두뇌에 해당합니다. 따라서 로직 티어를 구성하기 위해 Amazon API Gateway와 AWS Lambda를 통합하는 것은 획기적인 방법입니다. 두 서비스의 기능을 사용하면 고가용성의 확장 가능하고안전하며 서버가 없는 프로덕션 애플리케이션을 구축할 수 있습니다. 사용 중인 애플리케이션이 수천 개의 서버를 사용해도, 이 패턴을 활용하면 단 하나만 관리하지 않아도 됩니다. 또한 이러한 관리형 서비스를 함께 사용하면 다음과 같은 이점을 누릴 수 있습니다.

- 선택, 보호, 패치 적용 또는 관리할 운영 체제 없음
- 크기 조정, 모니터링 또는 확장할 서버 없음
- 프로비저닝 과다로 인한 비용 손실 위험 없음
- 프로비저닝 부족으로 인한 성능저하 위험 없음

또한 각각의 서비스에 멀티 티어 아키텍처 패턴에 유용한 세부 기능이 있습니다.

Amazon API Gateway

Amazon API Gateway는 API를 정의, 배포 및 유지 보수할 수 있도록 완벽하게 관리하는 서비스이며, 클라이언트가 표준 HTTPS 요청을 이용해 API와 통합합니다. 서비스 지향 멀티 티어 아키텍처에 명백하게 적용되지만, 로직 티어에 대한 강력한 에지를 생성하는 특정 특성과 본질을 가지고 있습니다.

AWS Lambda와 통합

Amazon API Gateway를 사용하면 애플리케이션에서 AWS Lambda의 혁신을 직접 간편한 방식(HTTPS 요청)으로 이용할 수 있습니다. API Gateway가 AWS Lambda에서 작성하는 함수와 프레젠테이션 티어를 연결하는 브리지를 형성합니다. API를 사용하여 클라이언트/서버 관계를 정의하면 클라이언트의 HTTPS 요청 콘텐츠가 실행될 수 있도록 Lambda 함수에 전달됩니다. 이러한 콘텐츠에 요청 메타데이터, 요청 헤더 및 요청 본문이 포함됩니다.

전 세계에서 안정적인 API 성능

Amazon API Gateway의 배포에 각각 Amazon CloudFront[4] 배포가 함께 포함됩니다. Amazon CloudFront는 사용하는 API와 통합되는 클라이언트의 연결 지점으로 Amazon 글로벌 네트워크의 엣지 로케이션을 이용하는 콘텐츠 전송 웹 서비스이며, 이를 통해 API의 총 응답 지연 시간을 억제할 수 있습니다. 또한 전 세계의 여러 엣지 로케이션을 이용하여 Amazon CloudFront가 DDoS(분산 서비스 거부) 공격 시나리오에 대처할 수 있는 기능을 제공합니다. 자세한 내용은 DDoS 대응을 위한 AWS 모범사례[5] 백서를 읽으십시오.

선택적 인 메모리 캐시에 응답을 저장하도록 Amazon API Gateway를 사용하여 특정 API 요청의 성능을 개선할 수 있습니다. 이를 통해 반복 API 요청에 대한 성능이 향상될 뿐만 아니라 백엔드 실행을 줄여 전체 비용을 절감할 수 있습니다.

혁신 조장

새 애플리케이션을 구축하는 데 필요한 개발 작업은 투자입니다. 프로젝트에 착수하려면 정당한 이유를 설명해야 합니다. 개발 작업에 필요한 투자 금액과 시간을 줄임으로써, 더 자유롭게 실험하고 여유 있게 혁신을 이룰 수 있습니다.

많은 멀티 티어 웹 서비스 기반 애플리케이션에 대해 프레젠테이션 티어가 사용자(개별 모바일 장치 및 웹 브라우저 등) 간에 쉽게 조각화됩니다. 또한 이러한 사용자는 지리적 위치에 구애받지 않는 경우가 많습니다. 그러나 분리된 로직 티어는 사용자에 의해 물리적으로 조각화되지 않습니다. 모든 사용자가 로직 티어가 실행되는 동일한 인프라에 종속되므로, 인프라의 중요성이 더 커집니다. 초기에 로직 티어를 구현할 때 절약("최초 시작 시 메트릭을 계측할 필요 없음", “초기 사용량이 줄어 나중에 조정할 방법 고려" 등)하는 방법이 주로 새 애플리케이션을 더 빠르게 전달하는 메커니즘으로 제시됩니다. 이로 인해 이미 프로덕션에서 실행 중인 애플리케이션에 이러한 변경 사항을 배포해야 할 경우 기술적인 부담과 운영상의 위험이 발생할 수 있습니다. Amazon API Gateway를 사용하면 서비스가 이미 구현되어 있기 때문에 *사용자가* 이러한 위험을 줄이고 더욱 빠르게 전달할 수 있습니다.

애플리케이션의 전체 수명을 알 수 없거나 수명이 짧은 상태일 수 있습니다. 이러한 이유 때문에 새 멀티 티어 애플리케이션의 비즈니스 사례를 만들기가 어려울 수 있습니다. 시작점에 Amazon API Gateway가 제공하는 관리형 기능이 이미 포함되어 있고, API가 요청을 수신하기 시작한 후 인프라 비용이 발생되기 시작하는 위치에서 이를 더욱 쉽게 수행할 수 있습니다. 자세한 내용은 Amazon API Gateway 요금을 참조하십시오.[6]

신속한 반복으로 민첩성 유지

새 애플리케이션을 사용하면 사용자 기반이 잘못 정의될 수 있습니다(크기, 사용 패턴 등). 사용자 기반이 형성되는 동안 로직 티어가 민첩하게 유지되어야 합니다. 애플리케이션과 비즈니스를 전환하여 변화하는 얼리 어답터의 기대에 부응할 수 있어야 합니다. Amazon API Gateway를 통해 시작부터 배포까지 API를 작성하는 데 필요한 개발 주기의 횟수를 줄일 수 있습니다. Amazon API Gateway는 클라이언트 애플리케이션이 전체 백엔드 로직이 개발 중인 상태에서 동시에 개발할 수 있는 API 응답을 API Gateway에서 직접 생성할 수 있는 가장 통합[7] 생성 기능을 제공합니다. 이러한 이점은 API의 최초 배포 시에만 적용되는 것이 아니라 비즈니스에서 애플리케이션 및 기존 API가 사용자에게 신속하게 대응하여 피벗해야 하는 것으로 결정한 후에도 적용됩니다. API Gateway 및 AWS Lambda를 사용하면 기존 기능과 클라이언트 종속성을 계속 그대로 유지하면서 새 기능이 별도의 API/기능 버전으로 릴리스될 수 있도록 버전 관리를 수행할 수 있습니다.

보안

퍼블릭 3티어 웹 애플리케이션의 로직 티어를 웹 서비스로 구현하면 보안에 관한 화제가 즉시 대두됩니다. 애플리케이션이 네트워크를 통해 노출되는 로직 티어에 권한이 있는 클라이언트만 액세스하도록 보증해야 합니다. Amazon API Gateway는 백엔드가 안전하다는 확신을 주는 방식으로 보안 관련 화제를 해결합니다. 액세스 제어를 위해 클라이언트 애플리케이션에 정적 API 키 문자열을 제공하는 데 의존하지 마십시오. 클라이언트로부터 추출되어 다른 곳에서 사용될 수 있습니다. Amazon API Gateway가 로직 티어의 보안을 위해 수행하는 다음과 같은 여러 가지 방식을 활용할 수 있습니다.

- 전송 중에 암호화되도록 API에 대한 모든 요청을 HTTPS를 통해 수행합니다.
- AWS Lambda 함수가 Amazon API Gateway 내의 특정 API와 AWS Lambda의 특정 함수 간에만 신뢰 관계가 적용되도록 액세스를 제한할 수 있습니다. 내보내도록 선택한 API를 사용하지 않으면 해당 Lambda 함수를 호출할 수 있는 다른 방법이 없습니다.
- Amazon API Gateway를 통해 API와 통합할 클라이언트 SDK를 생성할 수 있습니다. 또한 API가 인증을 요구할 때 해당 SDK가 요청에 대한 서명을 관리합니다. 클라이언트 측에서 인증하는 데 사용된 API 자격 증명이 AWS Lambda 함수에 직접 전달되며, 이때 필요한 경우 보유하고 작성하는 코드 내에서 추가 인증이 발생할 수 있습니다.

- API의 일환으로 생성하는 각 리소스/메서드 조합에 AWS Identity and Access Management(IAM)[8] 정책에서 참조될 수 있는 고유한 특정 Amazon 리소스 이름(ARN)이 부여됩니다.
  - API가 다른 AWS 소유 API와 함께 가장 먼저 처리됩니다. IAM 정책이 세분화될 수 있으며, Amazon API Gateway를 통해 생성된 API의 특정 리소스/메서드를 참조할 수 있습니다.
  - API 액세스에 애플리케이션 코드의 컨텍스트 외부에서 생성하는 IAM 정책이 적용됩니다. 즉, 이러한 액세스 수준을 인식하거나 적용하기 위해 코드를 작성할 필요가 없습니다. 코드에 버그가 포함되거나 버그가 없는 경우 악용되면 안 됩니다.
  - API 액세스를 위해 AWS 서명 버전 4(SigV4)[9] 권한 부여 및 IAM 정책을 사용하여 클라이언트를 인증하면 필요한 경우 이러한 동일한 자격 증명이 다른 AWS 서비스 및 리소스에 대한 액세스를 제한하거나 허용할 수 있습니다(예: Amazon S3 버킷 또는 Amazon DynamoDB 테이블).

AWS Lambda

핵심은 AWS Lambda가 이벤트에 대한 응답으로 지원되는 언어로 작성된 임의의 코드(2015년 11월을 기준으로 노드, JVM 기반 및 Python)를 트리거하도록 허용하는 것입니다. 해당 이벤트는 AWS가 사용 가능할 수 있도록 지정하는 **이벤트 소스**(여기에서 현재 지원되는 이벤트 소스 참조[10])라는 여러 프로그래밍 방식 트리거 중 하나일 수 있습니다. AWS Lambda에 많이 사용되는 사용 사례가 Amazon Simple Storage Service(Amazon S3)[11]에 저장되는 처리 파일 또는 Amazon Kinesis[12]의 스트리밍 데이터 레코드와 같이 이벤트 중심 데이터 처리 워크플로우를 중심으로 고려됩니다.

Amazon API Gateway와 결합하여 사용하면 AWS Lambda 함수가 일반 웹 서비스의 컨텍스트 내에 존재하고 HTTPS 요청에 의해 직접 트리거될 수 있습니다. Amazon API Gateway가 로직 티어의 프런트 도어 역할을 하지만, 현재 해당 로직을 이러한 API 뒤에서 실행해야 하며, 이 위치에서 AWS Lambda가 시행됩니다.

비즈니스 로직 시작

AWS Lambda를 사용하면 이벤트에 의해 트리거될 때 실행되는 **핸들러**라는 코드 함수를 작성할 수 있습니다. 예를 들어, API에 HTTPS 요청과 같은 이벤트가 발생하면 트리거되는 핸들러를 작성할 수 있습니다. Lambda를 이용해 각각 업데이트, 호출 및 변경할 수 있는 선택한 세부 수준(API당 1개 또는 API 메서드당 1개)에 모듈형 핸들러를 생성할 수 있습니다. 이렇게 하면 핸들러가 적용되는 다른 종속성에 자유롭게 접근할 수 있습니다(예: 코드, 라이브러리, 기본 바이너리 또는 외부 웹 서비스를 이용해 업로드한 다른 함수). Lambda를 사용하면 생성하는 동안 필요한 종속성을 함수 정의로 모두 패키지할 수 있습니다. 함수를 만들 때 배포 패키지 안에 결과 핸들러 역할을 수행할 메서드를 지정합니다. 여러 Lambda 함수 정의에 동일한 배포 패키지를 자유롭게 재사용할 수 있지만, 각 Lambda 함수의 고유 핸들러가 동일한 배포 패키지 내에 지정될 수 있습니다. 서버가 없는 멀티 티어 아키텍처 패턴에서는 Amazon API Gateway에서 생성하는 API가 각각 필요한 비즈니스 로직을 실행하는 Lambda 함수 및 내부 핸들러와 통합됩니다.

Amazon VPC 통합

로직 티어의 핵심인 AWS Lambda가 데이터 티어와 직접 통합되는 구성 요소가 됩니다. 데이터 티어에는 중요한 비즈니스 또는 사용자 정보가 자주 포함되므로 데이터 티어를 엄격하게 보호해야 합니다. Lambda 함수에서 통합시킬 수 있는 AWS 서비스에 대해 IAM 정책을 사용하여 액세스 제어를 관리할 수 있습니다. 이러한 서비스에는 Amazon S3, Amazon DynamoDB, Amazon Kinesis, Amazon Simple Queue Service(Amazon SQS), Amazon Simple Notification Service(Amazon SNS) 및 기타 AWS Lambda 함수 등이 포함됩니다. 그러나 관계형 데이터베이스와 같이 자체 액세스 제어를 관리하는 구성 요소가 있을 수 있습니다. 이와 같은 구성 요소가 있으면 Amazon Virtual Private Cloud(Amazon VPC)[13]와 같은 프라이빗 네트워킹 환경에서 해당 구성 요소를 배포하여 보안을 향상할 수 있습니다.

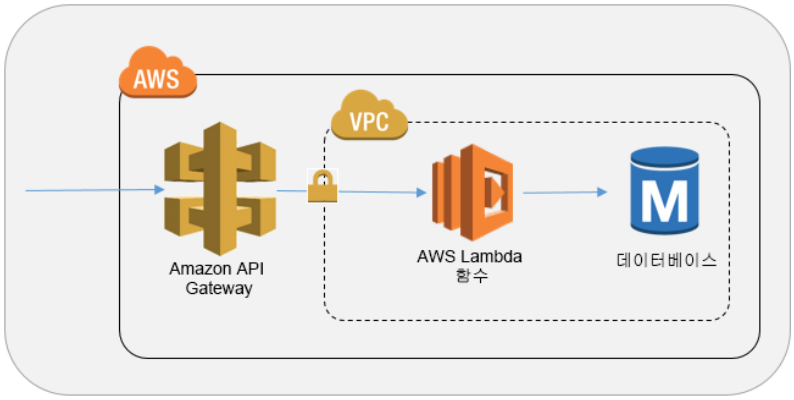

그림 2: VPC를 이용한 아키텍처 패턴

VPC를 사용하면 비즈니스 로직이 종속되는 데이터베이스 및 기타 스토리지 미디어에 인터넷을 통해 액세스할 수 없는 상태가 될 수 있습니다. 또한 VPC를 통해 사용자가 정의한 API 및 작성한 Lambda 코드 함수를 *통해서만* 인터넷에서 데이터와 상호 작용할 수 있도록 지정할 수 있습니다.

보안

Lambda 함수를 실행하려면 IAM 정책을 통해 이를 수행할 수 있도록 허용된 서비스 또는 이벤트로만 트리거해야 합니다. 사용자가 정의하는 API Gateway 요청에 의해 호출되지 않는 한, *아예* 실행되지 않는 Lambda 함수를 생성할 수 있습니다. 코드가 사용자가 생성한 API에 따라 정의된 유효한 사용 사례의 일부로만 처리합니다.

각 Lambda 함수가 IAM 신뢰 관계를 통해 권한을 부여해야 하는 기능인 IAM 역할을 자체적으로 가정합니다. 해당 IAM 역할에서 Lambda 함수가 상호 작용할 수 있는 다른 AWS 서비스/리소스를 정의합니다(예: Amazon DynamoDB 테이블 또는 Amazon S3 버킷). 함수가 액세스할 수 있는 서비스는 함수의 외부에서 직접 정의되고 제어되며, 감지하기는 힘들지만 강력합니다. 이를 통해 사용자가 작성하는 코드가 AWS 자격 증명을 저장하거나 수신하지 않아도 됩니다. 즉, API 키를 하드 코딩할 필요가 없으며 수신하고 메모리에 저장하기 위해 코드를 작성하지 않아도 됩니다. Lambda 함수가 서비스를 호출하도록 설정하면 IAM 역할에 의해 정의된 대로 서비스 자체에서 관리됩니다.

데이터 티어

AWS Lambda를 로직 티어로 사용하면 데이터 티어에 대한 다양한 데이터 스토리지 옵션을 사용할 수 있습니다. 이러한 옵션은 넓게 보면 Amazon VPC 호스팅 데이터 스토어 및 IAM 지원 데이터 스토어와 같은 두 가지 범주로 나뉩니다. AWS Lambda에 안전하게 통합하는 기능이 있습니다.

Amazon VPC 호스팅 데이터 스토어

Amazon VPC와 AWS Lambda의 통합을 통해 함수가 비공개 보안 방식으로 다양한 데이터 스토리지 기술과 통합할 수 있습니다.

- Amazon RDS[14]

  Amazon Relational Database Service(Amazon RDS)에서 사용할 수 있는 엔진을 사용합니다. Lambda의 외부에서와 같이 Lambda에서 작성한 코드에서 직접 Amazon RDS에 연결하지만, 데이터베이스 자격 증명 암호화를 위해 AWS Key Management Service(AWS KMS)와 간단히 통합할 수 있다는 이점이 있습니다.

- Amazon ElastiCache[15]

  Lambda 함수를 관리되는 인 메모리 캐시와 통합하여 애플리케이션의 성능을 향상합니다.

- Amazon RedShift[16]

  보고서 작성, 대시보드 또는 특별 쿼리 결과 검색을 위해 엔터프라이즈 데이터 웨어하우스를 안전하게 쿼리하는 기능을 구축할 수 있습니다.

- Amazon Elastic Compute Cloud(Amazon EC2)[17]에서 호스팅하는 프라이빗 웹 서비스

  VPC 내에서 비공개 상태로 웹 서비스로 실행되는 기존 애플리케이션이 있을 수 있습니다. Lambda 함수에서 프라이빗 VPC 네트워크를 통해 HTTP 요청을 로컬에 작성합니다.

IAM 지원 데이터 스토어

AWS Lambda가 IAM과 통합되므로 AWS API를 이용해 직접 활용할 수 있는 AWS 서비스와의 통합 보안 유지를 위해 IAM을 사용할 수 있습니다.

- Amazon DynamoDB[18]

  Amazon DynamoDB는 AWS의 무제한 확장 가능한 NoSQL 데이터베이스입니다. 확장에 상관없이 한 자릿수 밀리초 단위 성능으로 데이터 레코드를 검색하려는 경우 Amazon DynamoDB를 고려하십시오(이 문서의 작성일 현재 400KB 미만). Amazon DynamoDB 세부 액세스 제어를 사용하면 DynamoDB에서 특정 데이터를 쿼리할 때 Lambda 함수가 최소 권한의 모범 사례를 따를 수 있습니다.

- Amazon S3[19]

  Amazon Simple Storage Service(Amazon S3)는 인터넷급 객체 스토리지를 제공합니다. Amazon S3는 99.999999999% 객체의 내구성에 맞게 설계되었으므로, 애플리케이션에 저렴하고 내구성이 뛰어난 스토리지가 필요할 경우 사용을 고려하십시오. 또한 Amazon S3는 일정 기간 동안 객체의 가용성이 최대 99.99%에 맞게 설계되었으므로, 애플리케이션에 고가용성 스토리지가 필요할 경우 사용할 것을 고려하십시오. Amazon S3에 저장된 객체(파일, 이미지, 로그 및 바이너리 데이터)에 HTTP를 통해 직접 액세스할 수 있습니다. Lambda 함수가 가상 프라이빗 종단점을 통해 Amazon S3과 안전하게 통신할 수 있으며, S3 내부 데이터를 Lambda 함수와 연결된 IAM 정책으로만 제한할 수 있습니다.

- Amazon Elasticsearch Service[20]

  Amazon Elasticsearch Service(Amazon ES)는 인기 있는 검색 및 분석 엔진인 Elasticsearch의 관리형 버전입니다. Amazon ES에서 클러스터, 장애 탐지 및 노드 대체의 관리 프로비저닝을 제공하므로, IAM 정책을 사용하여 Amazon ES API에 대한 액세스를 제한할 수 있습니다.

프레젠테이션 티어

Amazon API Gateway를 통해 프레젠테이션 티어를 광범위하게 사용할 수 있습니다. HTTPS 통신이 가능한 클라이언트가 인터넷 액세스 가능 HTTPS API에 액세스할 수 있습니다. 다음 목록에 애플리케이션의 프레젠테이션 티어에 대한 사용을 고려할 수 있는 일반적인 예가 나와 있습니다.

- 모바일 앱: Amazon API Gateway 및 AWS Lambda를 통한 고객 비즈니스 로직과의 통합 외에도 Amazon Cognito[21]를 메커니즘으로 사용하여 사용자 ID를 생성하고 관리할 수 있습니다.
- 정적 웹 사이트 콘텐츠(예: Amazon S3에 호스팅된 파일): Amazon API Gateway API를 cross-origin 리소스 공유(CORS)와 호환되도록 설정할 수 있습니다. 이렇게 하면 웹 브라우저가 정적 웹 페이지에서 API를 직접 호출할 수 있습니다.
- 기타 HTTPS 지원 클라이언트 디바이스: 여러 연결된 디바이스가 HTTPS를 통해 통신할 수 있습니다. Amazon API Gateway를 이용해 생성한 API와 클라이언트가 API와 통신하는 방식에 적용되는 고유하거나 독점적인 사항이 없으며, HTTPS만 있으면 됩니다. 특정 클라이언트 소프트웨어 또는 라이선스도 필요하지 않습니다.

샘플 아키텍처 패턴

Amazon API Gateway 및 AWS Lambda를 로직 티어를 형성하는 글루로 사용하여 다음과 같이 인기 있는 아키텍처 패턴을 구현할 수 있습니다. 각각의 예에는 사용자가 자신의 인프라를 관리할 필요가 없는 AWS 서비스만 사용됩니다.

모바일 백엔드

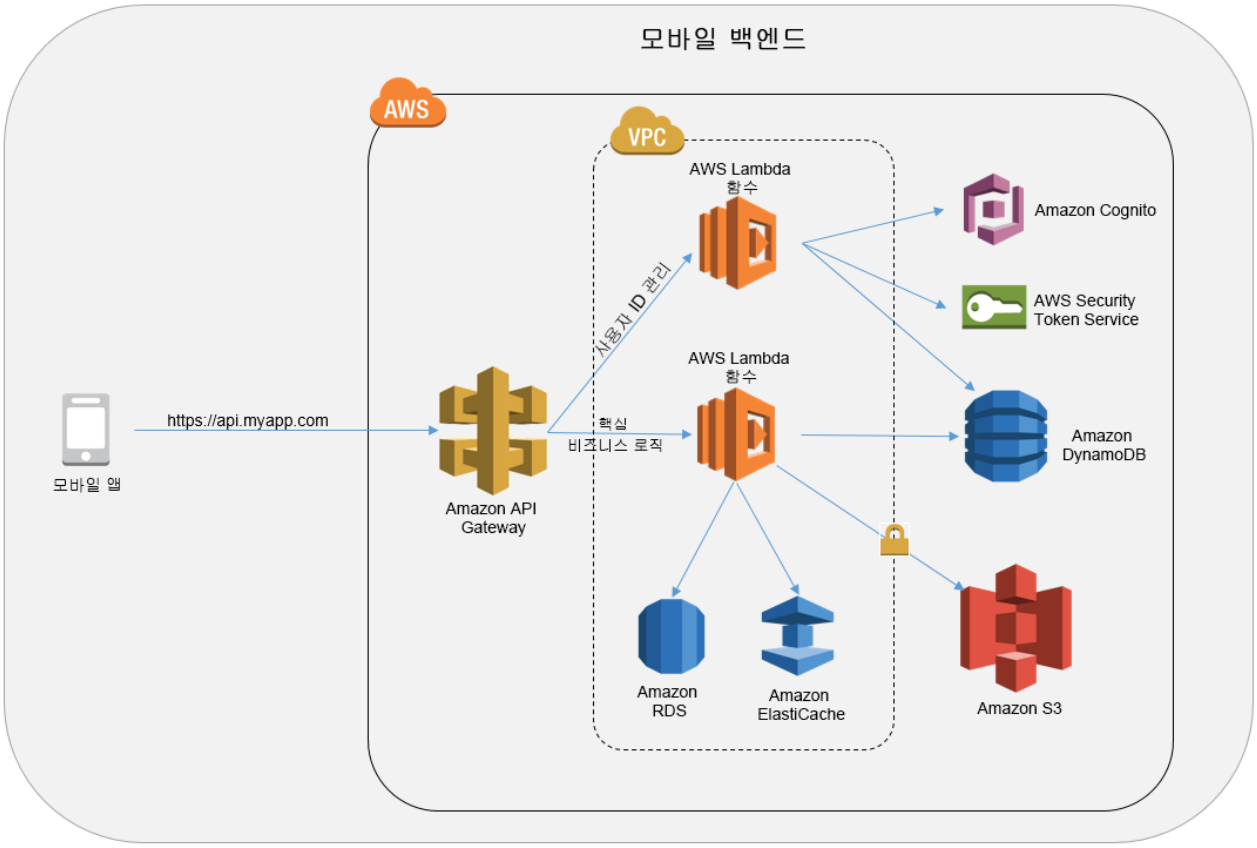

그림 3: 모바일백 엔드의 아키텍처 패턴

- **프레젠테이션 티어:** 각 사용자의 스마트폰에서 실행되는 모바일 애플리케이션입니다.
- **로직 티어:** Amazon API Gateway 및 AWS Lambda 입니다. 로직 티어는 각 Amazon API Gateway API 의 일환으로 생성된 Amazon CloudFront 배포에 의해 전역적으로 분산됩니다. Lambda 함수 집합은 사용자/디바이스 ID 관리 및 인증에 고유하고, 임시 사용자 액세스 자격 증명에 대한 IAM 및 인기 있는 타사 ID 공급자와의 통합을 제공하는 Amazon Cognito 에서 관리됩니다. 다른 Lambda 함수가 모바일 백엔드에 대한 핵심 비즈니스 로직을 정의할 수 있습니다.
- **데이터 티어:** 필요에 따라 다양한 데이터 스토리지 서비스를 이용할 수 있으며, 옵션은 본 백서의 앞 부분에 설명되어 있습니다.

Amazon S3 호스팅 웹 사이트

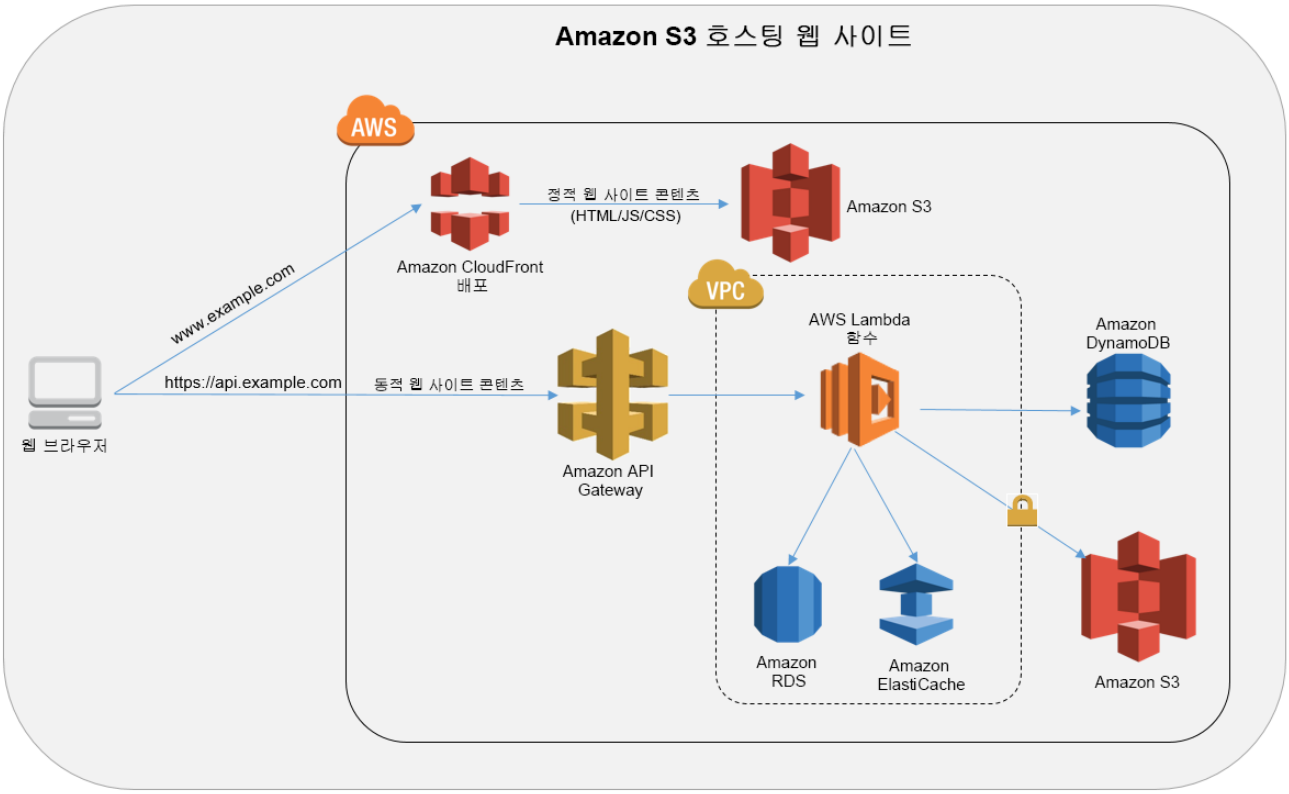

그림 4: Amazon S3에 호스팅된 정적 웹 사이트의 아키텍처 패턴

- **프레젠테이션 티어:** Amazon S3 에 호스팅되고 Amazon CloudFront 에 의해 분산된 정적 웹 사이트 콘텐츠입니다. Amazon S3 에 정적 웹 사이트 콘텐츠를 호스팅하는 것은 서버 기반 인프라에 콘텐츠를 호스팅하는 것보다 비용 효율적인 대안입니다. 그러나 다양한 기능을 포함하는 웹 사이트의 경우 정적 콘텐츠를 동적 백엔드와 통합해야 하는 경우가 많습니다.
- **로직 티어:** Amazon API Gateway 및 AWS Lambda입니다. Amazon S3에 호스팅된 정적 웹 콘텐츠가 CORS 호환 가능한 Amazon API Gateway와 직접 통합할 수 있습니다.
- **데이터 티어:** 필요에 따라 다양한 데이터 스토리지 서비스를 이용할 수 있습니다. 이러한 옵션은 본 백서의 앞 부분에 설명되어 있습니다.

마이크로서비스 환경

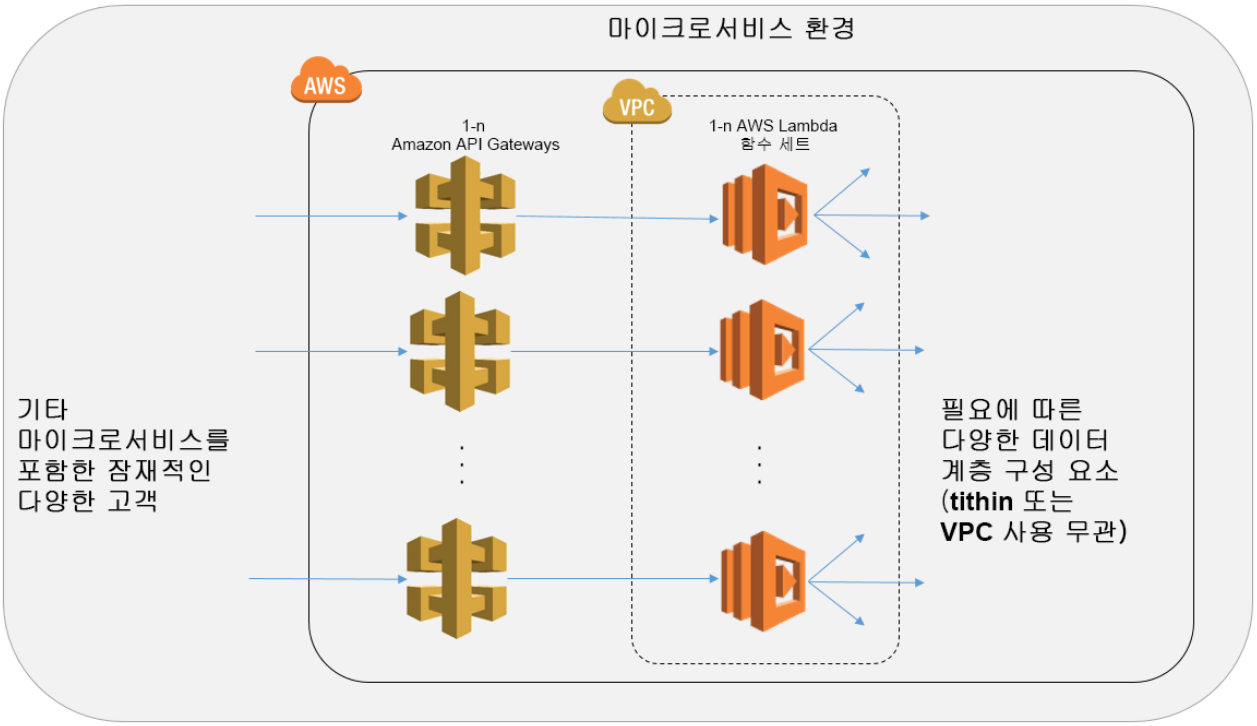

그림 5: 마이크로서비스 환경의 아키텍처 패턴

마이크로서비스 아키텍처 패턴은 본 백서에서 다룬 일반 3티어 아키텍처에 구속되지 않습니다. 마이크로서비스 아키텍처에서 소프트웨어 구성 요소의 대규모 분리가 발생하여 멀티 티어 아키텍처의 이점이 증폭되었습니다. Amazon API Gateway로 생성된 APIDHK AWS Lambda에서 이후에 실행된 함수가 모두 마이크로서비스를 빌드하는 데 필요합니다. 팀에서 이러한 서비스를 자유롭게 이용하여 원하는 세부 수준으로 환경을 분리하고 조각화할 수 있습니다.

일반적으로 마이크로서비스 환경에는 새로운 마이크로서비스를 각각 생성하기 위해 반복되는 오버헤드, 서버 밀도/사용률 최적화 관련 문제, 여러 버전의 멀티 마이크로서비스를 동시에 실행하는 복잡성 및 많은 개별 서비스와 통합하기 위한 클라이언트 측 코드 요구 사항의 급증 등의 어려움이 발생할 수 있습니다.

그러나 AWS의 서버가 없는 패턴을 사용하여 마이크로서비스를 만들면 이러한 문제가 더 단순해지며 일부 사례에서는 문제가 해결되기도 합니다. AWS 마이크로서비스 패턴이 각 후속 마이크로서비스에 대한 장벽을 낮출 수 있으며, Amazon API Gateway를 통해 기존 API를 복제할 수도 있습니다. 이 패턴에서는 더 이상 서버 사용률의 최적화를 수행할 필요가 없습니다. API Gateway와 Lambda를 둘 다 사용하여 버전 관리 기능을 간단히 수행할 수 있습니다. 마지막으로 Amazon API Gateway가 통합 오버헤드를 줄일 수 있도록 많이 사용되는 몇 가지 언어로 프로그래밍 방식으로 생성된 클라이언트 SDK를 제공합니다.

결론

멀티 티어 아키텍처 패턴에서는 쉽게 유지 보수, 분리 및 확장 가능한 애플리케이션 구성 요소를 생성하는 모범 사례를 따르는 것이 좋습니다. Amazon API Gateway를 통해 통합이 발생하고 AWS Lambda 내에서 계산이 수행되는 로직 티어를 만들면 목표를 달성하기 위한 수고를 줄이면서 이를 실현할 수 있습니다. 또한 이러한 서비스가 VPC 내의 클라이언트 및 보안 환경에서 비즈니스 로직을 실행할 수 있도록 HTTPS API 프런트 엔드를 제공합니다. 이를 통해 일반 서비스 기반 인프라를 직접 관리하지 않고 관리형 서비스를 사용할 수 있는 여러 보편적인 시나리오를 활용할 수 있습니다.

기고자

다음은 이 문서의 작성에 도움을 준 개인 및 조직입니다.

Andrew Baird, AWS 솔루션 아키텍트

Stefano Buliani, 선임 제품 관리자, Tech, AWS Mobile

Vyom Nagrani, 선임 제품 관리자, AWS Mobile

Ajay Nair, 선임 제품 관리자, AWS Mobile

참고

[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/