AWS 백서

에서 마이크로서비스 구현 AWS

에서 마이크로서비스 구현 AWS: AWS 백서

Table of Contents

에서 마이크로서비스 구현 AWS

게시일: 2023년 7월 31일(문서 이력)

마이크로서비스는 배포를 가속화하고, 혁신을 장려하고, 유지 관리를 개선하고, 확장성을 높이는 소프트웨어 개발에 대한 간소화된 접근 방식을 제공합니다. 이 방법은 자율 팀이 관리하는 잘 정의된 APIs를 통해 통신하는 작고 느슨하게 결합된 서비스에 의존합니다. 마이크로서비스를 채택하면 확장성, 복원력, 유연성 및 개발 주기 단축과 같은 이점을 얻을 수 있습니다.

이 백서에서는 API 기반, 이벤트 기반, 데이터 스트리밍의 세 가지 인기 마이크로서비스 패턴을 살펴봅니다. 각 접근 방식에 대한 개요를 제공하고, 마이크로서비스의 주요 기능을 간략하게 설명하고, 개발 시 발생하는 문제를 해결하고, Amazon Web Services(AWS)가 애플리케이션 팀이 이러한 장애물을 해결하는 데 어떻게 도움이 될 수 있는지 설명합니다.

데이터 스토어, 비동기 통신 및 서비스 검색과 같은 주제의 복잡한 특성을 고려할 때 아키텍처 결정을 내릴 때 제공된 지침과 함께 애플리케이션의 특정 요구 사항 및 사용 사례를 고려하는 것이 좋습니다.

소개

마이크로서비스 아키텍처는 다음과 같은 다양한 필드의 성공적 개념과 검증된 개념을 결합합니다.

- 애자일 소프트웨어 개발
- 서비스 지향 아키텍처
- API 우선 설계
- 지속적 통합/지속적 전송(CI/CD)

마이크로서비스는 12단계 앱의 설계 패턴을 통합하는 경우가 많습니다.

마이크로서비스는 많은 이점을 제공하지만 사용 사례의 고유한 요구 사항과 관련 비용을 평가하는 것이 중요합니다. 경우에 따라 모놀리식 아키텍처 또는 대체 접근 방식이 더 적절할 수 있습니다. 마이크로서비스 또는 모놀리스는 규모, 복잡성 및 특정 사용 사례와 같은 요소를 고려하여 case-by-case 결정해야 합니다.

먼저 확장성과 내결함성이 뛰어난 마이크로서비스 아키텍처(사용자 인터페이스, 마이크로서비스 구현, 데이터 스토어)를 살펴보고 컨테이너 기술을 AWS 사용하여 이를 구축하는 방법을 보여줍니다. 그런 다음 일반적인 서버리스 마이크로서비스 아키텍처를 구현하여 운영 복잡성을 줄이는 AWS 서비스를 제안합니다.

Serverless는 다음 원칙을 특징으로 합니다.

- 프로비저닝 또는 관리할 인프라 없음
- 소비 단위로 자동 조정
- "값 지불" 결제 모델
- 기본 제공 가용성 및 내결함성
- 이벤트 기반 아키텍처(EDA)

마지막으로 전체 시스템을 검사하고 분산 모니터링, 로깅, 추적, 감사, 데이터 일관성, 비동기 통신과 같은 마이크로서비스 아키텍처의 교차 서비스 측면에 대해 설명합니다.

이 문서는 하이브리드 시나리오 및 마이그레이션 전략을 제외하고 AWS 클라우드에서 실행되는 워크로드에 중점을 둡니다. 마이그레이션 전략에 대한 자세한 내용은 컨테이너 마이그레이션 방법 백서를 참조하세요.

귀사는 Well-Architected입니까?

AWS Well-Architected 프레임워크는 클라우드에서 시스템을 구축할 때 내리는 결정의 장단점을 이해하는 데 도움이 됩니다. 이 프레임워크를 사용하여 클라우드에서 안정적이고 안전하며 효율적이고 비용 효율적인 시스템을 설계하고 운영하기 위한 아키텍처 모범 사례를 살펴볼 수 있습니다. AWS Management Console에서 무료로 제공되는 AWS Well-Architected Tool를 사용하면 각 요소에 대한 일련의 질문에 답하여, 이러한 모범 사례와 비교하여 워크로드를 검토할 수 있습니다.

서버리스 애플리케이션 렌즈에서는 서버리스 애플리케이션을 설계하기 위한 모범 사례에 중점을 둡니다 AWS.

참조 아키텍처 배포, 다이어그램, 백서 등 클라우드 아키텍처에 대한 더 많은 전문가 지침과 모범 사례를 보려면 AWS 아키텍처 센터를 참조하세요.

마이크로서비스로 현대화

마이크로서비스는 애플리케이션을 구성하는 본질적으로 작고 독립적인 단위입니다. 기존 모놀리식 구조에서 마이크로서비스로 전환하는 것은 다양한 전략을 따를 수 있습니다.

이 전환은 조직의 운영 방식에도 영향을 미칩니다.

- 팀이 빠른 주기로 작업하는 애자일 개발을 장려합니다.

- 팀은 일반적으로 작으며, 피자 팀 두 개로 설명되기도 합니다. 피자 두 개가 전체 팀에 공급할 수 있을 만큼 작습니다.
- 팀은 생성부터 배포 및 유지 관리에 이르기까지 서비스에 대한 모든 책임을 집니다.

의 간단한 마이크로서비스 아키텍처 AWS

일반적인 모놀리식 애플리케이션은 프레젠테이션 계층, 애플리케이션 계층, 데이터 계층 등 다양한 계층으로 구성됩니다. 반면 마이크로서비스 아키텍처는 기술 계층이 아닌 특정 도메인에 따라 기능을 응집력 있는 수직으로 구분합니다. 그림 1은 일반적인 마이크로서비스 애플리케이션의 참조 아키텍처를 보여줍니다 AWS.

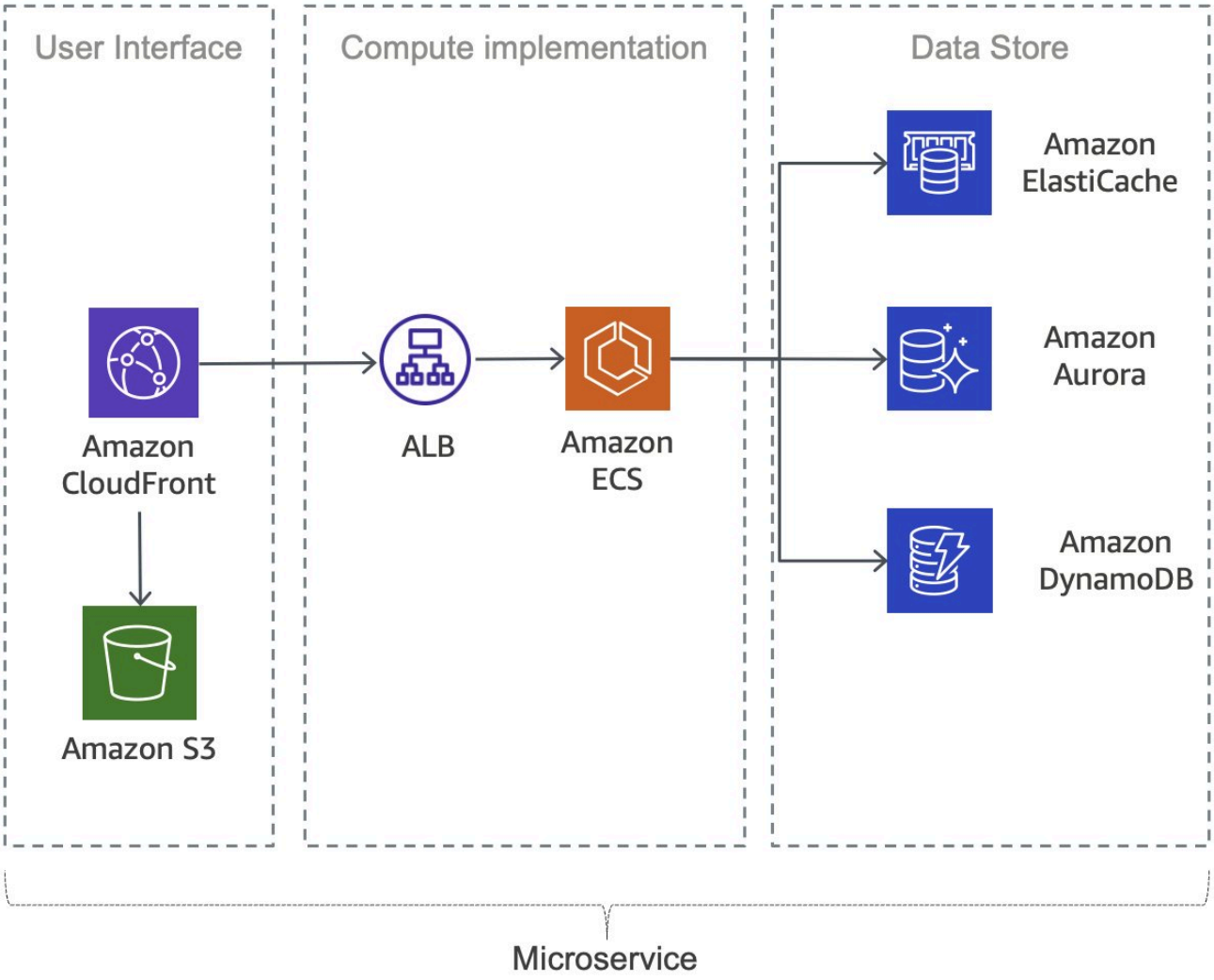

그림 1:의 일반적인 마이크로서비스 애플리케이션 AWS

사용자 인터페이스

최신 웹 애플리케이션은 JavaScript 프레임워크를 사용하여 백엔드 APIs. 이러한 APIs는 일반적으로 REST(Representational State Transfer) 또는 RESTful APIs 또는 GraphQL APIs. 정적 웹 콘텐츠는

Amazon Simple Storage Service(Amazon S3) 및 Amazon CloudFront를 사용하여 제공할 수 있습니다.

마이크로서비스

APIs는 애플리케이션 로직의 진입점이므로 마이크로서비스의 정문으로 간주됩니다. 일반적으로 RESTful 웹 서비스 API 또는 GraphQL APIs 사용됩니다. 이러한 APIs 클라이언트 호출을 관리하고 처리하여 트래픽 관리, 요청 필터링, 라우팅, 캐싱, 인증 및 권한 부여와 같은 기능을 처리합니다.

마이크로서비스 구현

AWS 는 컨테이너 오케스트레이션 엔진의 선택 항목으로 Amazon ECS 및 Amazon EKS와 호스팅 옵션으로 AWS Fargate EC2를 포함한 마이크로서비스를 개발하기 위한 구성 요소를 제공합니다. AWS Lambda 는 마이크로서비스를 구축하는 또 다른 서버리스 방법입니다 AWS. 이러한 호스팅 옵션 중에서 선택하는 것은 기본 인프라를 관리하기 위한 고객의 요구 사항에 따라 달라집니다.

AWS Lambda 를 사용하면 코드를 업로드하여 고가용성으로 실행을 자동으로 조정하고 관리할 수 있습니다. 따라서 인프라 관리가 필요하지 않으므로 빠르게 이동하고 비즈니스 로직에 집중할 수 있습니다. Lambda는 여러 프로그래밍 언어를 지원하며 다른 AWS 서비스에서 트리거하거나 웹 또는 모바일 애플리케이션에서 직접 호출할 수 있습니다.

컨테이너 기반 애플리케이션은 이식성, 생산성 및 효율성으로 인해 인기를 얻었습니다. AWS 는 컨테이너를 빌드, 배포 및 관리하기 위한 여러 서비스를 제공합니다.

- App2Container는 Java 및 .NET 웹 애플리케이션을 컨테이너 형식으로 마이그레이션하고 현대화하기 위한 명령줄 도구입니다. AWS A2C는 베어 메탈, 가상 머신, Amazon Elastic Compute Cloud(EC2) 인스턴스 또는 클라우드에서 실행되는 애플리케이션의 인벤토리를 분석하고 빌드합니다.

- Amazon Elastic Container Service(Amazon ECS) 및 Amazon Elastic Kubernetes Service(Amazon EKS)는 컨테이너 인프라를 관리하므로 컨테이너화된 애플리케이션을 더 쉽게 시작하고 유지 관리할 수 있습니다.
  - Amazon EKS는 AWS 클라우드 및 온프레미스 데이터 센터(Amazon EKS Anywhere)에서 Kubernetes를 실행하는 관리형 Kubernetes 서비스입니다. 이렇게 하면 지연 시간이 짧은 로컬 데이터 처리, 높은 데이터 전송 비용 또는 데이터 레지던시 요구 사항을 위해 클라우드 서비스가 온프레미스 환경으로 확장됩니다("Amazon EKS Anywhere를 사용하여 하이브리드 컨테이너 워크로드 실행" 백서 참조). Kubernetes 커뮤니티의 모든 기존 플러그인 및 도구를 EKS와 함께 사용할 수 있습니다.

- Amazon Elastic Container Service(Amazon ECS)는 컨테이너화된 애플리케이션의 배포, 관리 및 규모 조정을 간소화하는 완전 관리형 컨테이너 오케스트레이션 서비스입니다. 고객은 간소화 및 AWS 서비스와의 심층 통합을 위해 ECS를 선택합니다.

자세한 내용은 Amazon ECS vs Amazon EKS: AWS 컨테이너 서비스 이해 블로그를 참조하세요.

- AWS App Runner는 사전 인프라 또는 컨테이너 경험 없이 컨테이너화된 웹 애플리케이션 및 API 서비스를 빌드, 배포 및 실행할 수 있는 완전 관리형 컨테이너 애플리케이션 서비스입니다.
- AWS Fargate서버리스 컴퓨팅 엔진인는 Amazon ECS 및 Amazon EKS와 함께 작동하여 컨테이너 애플리케이션의 컴퓨팅 리소스를 자동으로 관리합니다.
- Amazon ECR은 고성능 호스팅을 제공하는 완전 관리형 컨테이너 레지스트리이므로 애플리케이션 이미지와 아티팩트를 어디서나 안정적으로 배포할 수 있습니다.

지속적 통합 및 지속적 배포(CI/CD)

지속적 통합 및 지속적 전달(CI/CD)은 신속한 소프트웨어 변경을 위한 DevOps 이니셔티브의 중요한 부분입니다.는 마이크로서비스용 CI/CD를 구현하기 위한 서비스를 AWS 제공하지만 자세한 설명은이 문서의 범위를 벗어납니다. 자세한 내용은 지속적인 통합 및 지속적 전달 연습 AWS 백서를 참조하세요.

프라이빗 네트워킹

AWS PrivateLink 는 Virtual Private Cloud(VPC)와 지원되는 AWS 서비스 간의 프라이빗 연결을 허용하여 마이크로서비스의 보안을 강화하는 기술입니다. 마이크로서비스 트래픽을 격리하고 보호하여 퍼블릭 인터넷을 통과하지 않도록 합니다. 이는 PCI 또는 HIPAA와 같은 규정을 준수하는 데 특히 유용합니다.

데이터 스토어

데이터 스토어는 마이크로서비스에 필요한 데이터를 유지하는 데 사용됩니다. 세션 데이터의 인기 저장소는 Memcached 또는 Redis와 같은 인 메모리 캐시입니다.는 관리형 Amazon ElastiCache 서비스의 일부로 두 기술을 모두 AWS 제공합니다.

애플리케이션 서버와 데이터베이스 사이에 캐시를 넣는 것은 데이터베이스의 읽기 부하를 줄이는 일반적인 메커니즘으로, 더 많은 쓰기를 지원하는 데 리소스를 사용할 수 있습니다. 캐시는 지연 시간도 개선할 수 있습니다.

관계형 데이터베이스는 여전히 구조화된 데이터 및 비즈니스 객체를 저장하는 데 매우 인기가 있습니다.는 Amazon Amazon Relational Database Service (Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL 및 Amazon Aurora)을 관리형 서비스로 AWS 제공합니다.

그러나 관계형 데이터베이스는 무한 규모로 설계되지 않았으므로 많은 수의 쿼리를 지원하는 기술을 적용하는 것이 어렵고 시간이 많이 걸릴 수 있습니다.

NoSQL 데이터베이스는 관계형 데이터베이스의 일관성보다 확장성, 성능 및 가용성을 높이도록 설계되었습니다. NoSQL 데이터베이스의 중요한 요소 중 하나는 일반적으로 엄격한 스키마를 적용하지 않는다는 것입니다. 데이터는 수평적으로 확장할 수 있는 파티션에 분산되며 파티션 키를 사용하여 검색됩니다.

개별 마이크로서비스는 한 가지를 잘 수행하도록 설계되었으므로 일반적으로 NoSQL 지속성에 적합할 수 있는 간소화된 데이터 모델을 갖추고 있습니다. NoSQL 데이터베이스의 액세스 패턴은 관계형 데이터베이스와 다르다는 점을 이해하는 것이 중요합니다. 예를 들어 테이블을 조인할 수 없습니다. 필요한 경우 애플리케이션에서 로직을 구현해야 합니다. Amazon DynamoDB를 사용하여 원하는 양의 데이터를 저장 및 검색하고 모든 수준의 요청 트래픽을 처리할 수 있는 데이터베이스 테이블을 생성할 수 있습니다. DynamoDB는 한 자릿수 밀리초의 성능을 제공하지만 응답 시간이 마이크로초 단위로 필요한 특정 사용 사례가 있습니다. DynamoDB Accelerator (DAX)는 데이터 액세스를 위한 캐싱 기능을 제공합니다.

또한 DynamoDB는 실제 트래픽에 따라 처리량 용량을 동적으로 조정하는 자동 조정 기능을 제공합니다. 그러나 애플리케이션에서 짧은 기간의 대규모 활동 급증으로 인해 용량 계획이 어렵거나 불가능한 경우가 있습니다. 이러한 상황에서 DynamoDB는 pay-per-request 요금을 제공하는 온디맨드 옵션을 제공합니다. DynamoDB 온디맨드는 용량 계획 없이 초당 수천 개의 요청을 즉시 처리할 수 있습니다.

자세한 내용은 분산 데이터 관리 및 데이터베이스를 선택하는 방법을 참조하세요.

작업 간소화

마이크로서비스를 실행, 유지 관리 및 모니터링하는 데 필요한 운영 노력을 더욱 간소화하기 위해 완전한 서버리스 아키텍처를 사용할 수 있습니다.

Lambda 기반 애플리케이션 배포

`zip` 파일 아카이브를 업로드하거나 유효한 Amazon ECR 이미지 URI를 사용하여 콘솔 UI를 통해 컨테이너 이미지를 생성하고 업로드하여 Lambda 코드를 배포할 수 있습니다. 그러나 Lambda 함수가 복잡해져 계층, 종속성 및 권한이 있는 경우 코드 변경 시 UI를 통한 업로드가 불안정해질 수 있습니다.

AWS CloudFormation 및 AWS Serverless Application Model (AWS SAM) AWS Cloud Development Kit (AWS CDK)또는 Terraform을 사용하면 서버리스 애플리케이션을 정의하는 프로세스가 간소화됩니다. CloudFormation에서 기본적으로 지원하는 AWS SAM은 서버리스 리소스를 지정하기 위한 간소화된 구문을 제공합니다. AWS Lambda 계층은 여러 Lambda 함수에서 공유 라이브러리를 관리하고, 함수 공간을 최소화하고, 테넌트 인식 라이브러리를 중앙 집중화하고, 개발자 경험을 개선하는 데 도움이 됩니다. Java용 Lambda SnapStart는 지연 시간에 민감한 애플리케이션의 시작 성능을 향상시킵니다.

배포하려면 CloudFormation 템플릿에서 리소스 및 권한 정책을 지정하고, 배포 아티팩트를 패키징하고, 템플릿을 배포합니다. AWS CLI 도구인 SAM Local은 Lambda에 업로드하기 전에 서버리스 애플리케이션의 로컬 개발, 테스트 및 분석을 허용합니다.

AWS Cloud9 IDE AWS CodeBuild, AWS CodeDeploy등의 AWS CodePipeline 도구와 통합하면 SAM 기반 애플리케이션의 작성, 테스트, 디버깅 및 배포가 간소화됩니다.

다음 다이어그램은 CloudFormation 및 AWS CI/CD 도구를 사용하여 AWS Serverless Application Model 리소스를 배포하는 방법을 보여줍니다.

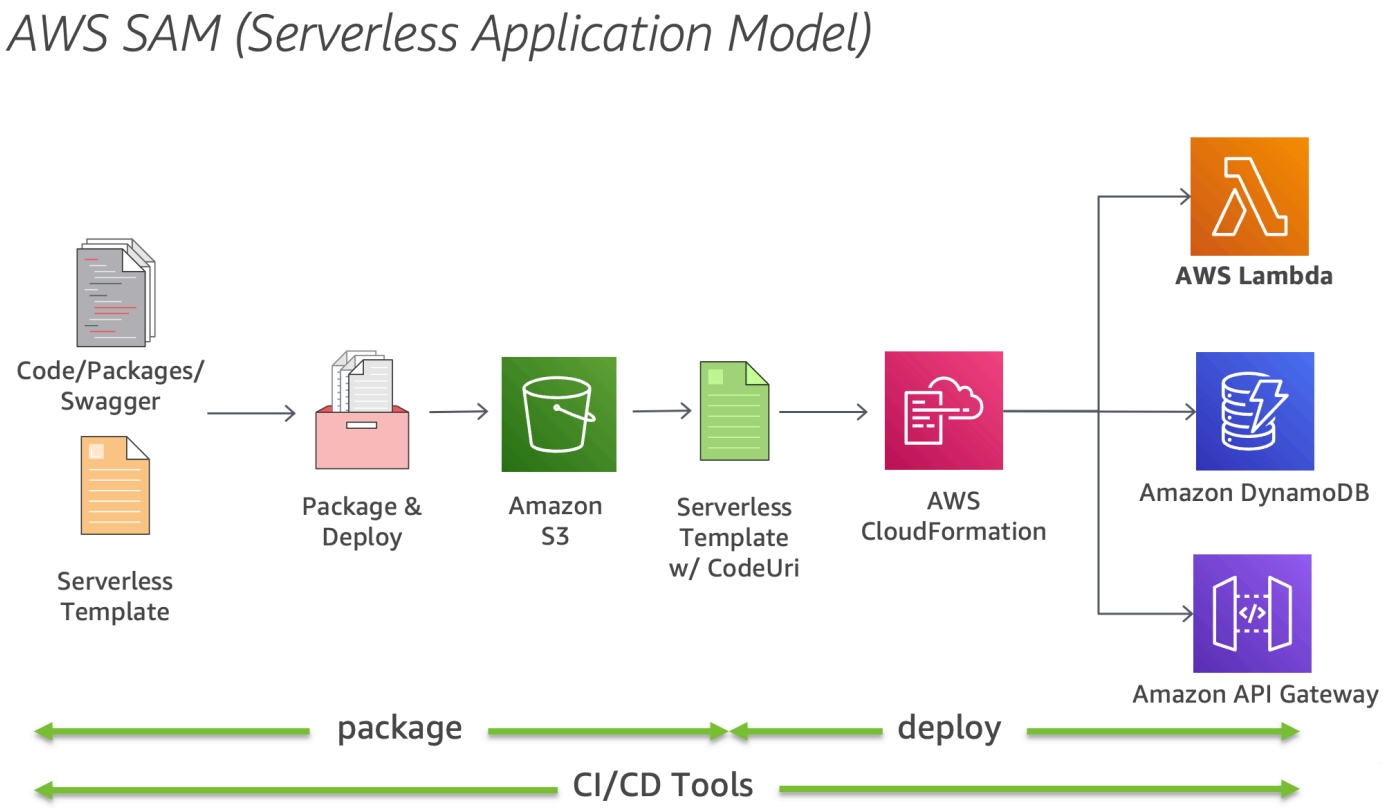

그림 2: AWS Serverless Application Model (AWS SAM)

다중 테넌시 복잡성 추상화

SaaS 플랫폼과 같은 다중 테넌트 환경에서는 개발자가 기능 개발에 집중할 수 있도록 다중 테넌시와 관련된 복잡성을 간소화하는 것이 중요합니다. 이는 교차 커팅 문제를 해결하기 위한 공유 라이브러리를 제공하는 AWS Lambda Layers와 같은 도구를 사용하여 달성할 수 있습니다.이 접근 방식의 근거는 공유 라이브러리와 도구를 올바르게 사용하면 테넌트 컨텍스트를 효율적으로 관리하는 것입니다.

그러나 이로 인해 발생할 수 있는 복잡성과 위험으로 인해 비즈니스 로직을 캡슐화하는 것으로 확장해서는 안 됩니다. 공유 라이브러리의 근본적인 문제는 업데이트를 둘러싼 복잡성이 증가하여 표준 코드 복제에 비해 관리하기가 더 어려워진다는 것입니다. 따라서 가장 효과적인 추상화를 위해 공유 라이브러리 사용과 중복 간의 균형을 맞추는 것이 중요합니다.

API 관리

APIs 관리는 특히 여러 버전, 개발 주기의 단계, 권한 부여 및 제한 및 캐싱과 같은 기타 기능을 고려할 때 시간이 많이 걸릴 수 있습니다. API Gateway 외에도 일부 고객은 API 관리를 위해 ALB(Application Load Balancer) 또는 NLB(Network Load Balancer)를 사용합니다. Amazon API Gateway는 RESTful APIs. 이를 통해 프로그래밍 방식으로 APIs를 생성할 수 있고, 백엔드 서비스의 데이터, 비즈니스 로직 또는 기능, 권한 부여 및 액세스 제어, 속도 제한, 캐싱, 모니터링 및 트래픽 관리에 액세스할 수 있는 “정문” 역할을 하며, 서버를 관리하지 않고도 APIs.

그림 3은 API Gateway가 API 호출을 처리하고 다른 구성 요소와 상호 작용하는 방법을 보여줍니다. 모바일 디바이스, 웹 사이트 또는 기타 백엔드 서비스의 요청은 지연 시간을 줄이고 최적의 사용자 경험을 제공하기 위해 가장 가까운 CloudFront PoP(Point of Presence) 로 라우팅됩니다.

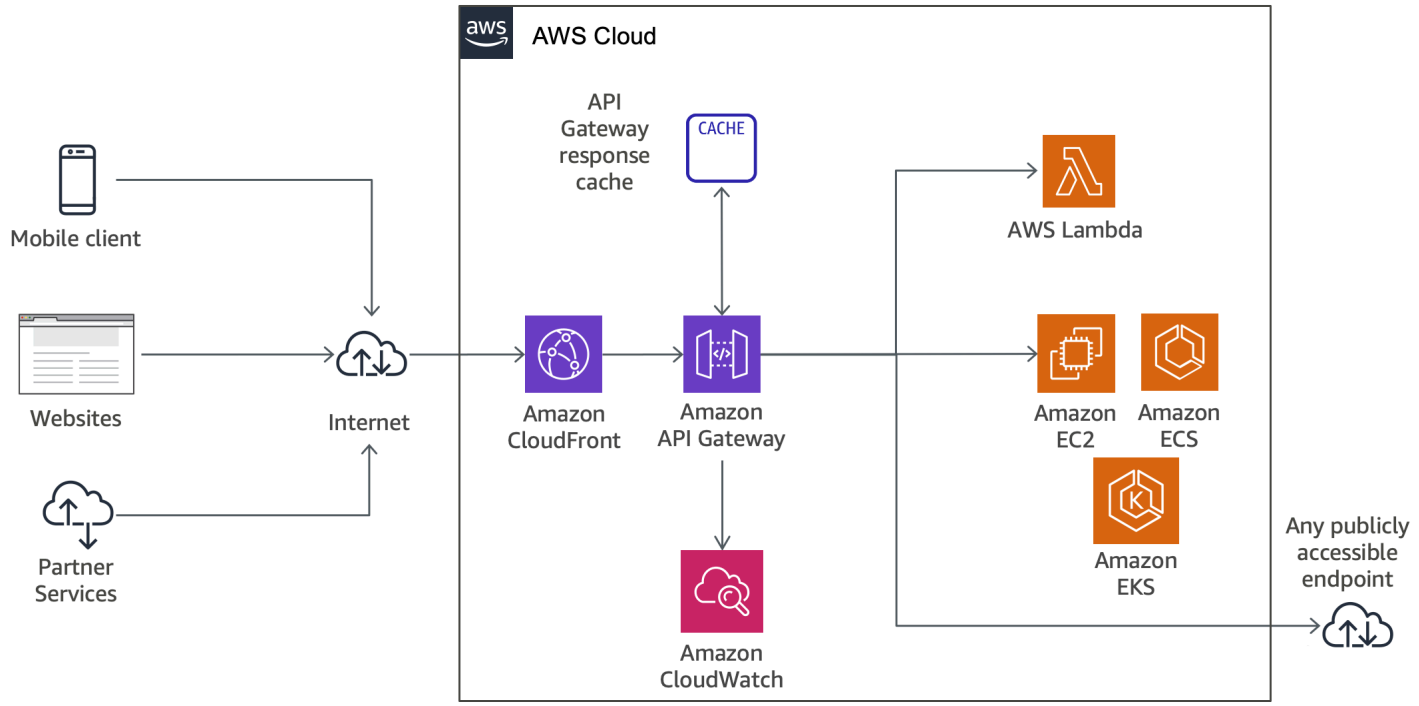

그림 3: API Gateway 호출 흐름

서버리스 기술의 마이크로서비스

서버리스 기술과 함께 마이크로서비스를 사용하면 운영 복잡성을 크게 줄일 수 있습니다. AWS Lambda 또한 API Gateway와 AWS Fargate통합되어 완전한 서버리스 애플리케이션을 생성할 수 있습니다. 2023년 4월 7일부터 Lambda 함수는 응답 페이로드를 클라이언트로 점진적으로 스트리밍하여 웹 및 모바일 애플리케이션의 성능을 향상시킬 수 있습니다. 이전에는 기존 요청-응답 호출 모델을 사용하는 Lambda 기반 애플리케이션이 클라이언트에 응답을 반환하기 전에 응답을 생성하고 버퍼링해야 했으므로 첫 번째 바이트까지 걸리는 시간이 지연될 수 있었습니다. 응답 스트리밍을 사용하면 함수가 준비되면 부분 응답을 클라이언트로 다시 전송하여 웹 및 모바일 애플리케이션이 특히 민감한 첫 번째 바이트까지의 시간을 크게 개선할 수 있습니다.

그림 4는 AWS Lambda 및 관리형 서비스를 사용하는 서버리스 마이크로서비스 아키텍처를 보여줍니다. 이 서버리스 아키텍처는 확장 및 고가용성을 위해 설계할 필요성을 줄이고 기본 인프라를 실행하고 모니터링하는 데 필요한 노력을 줄입니다.

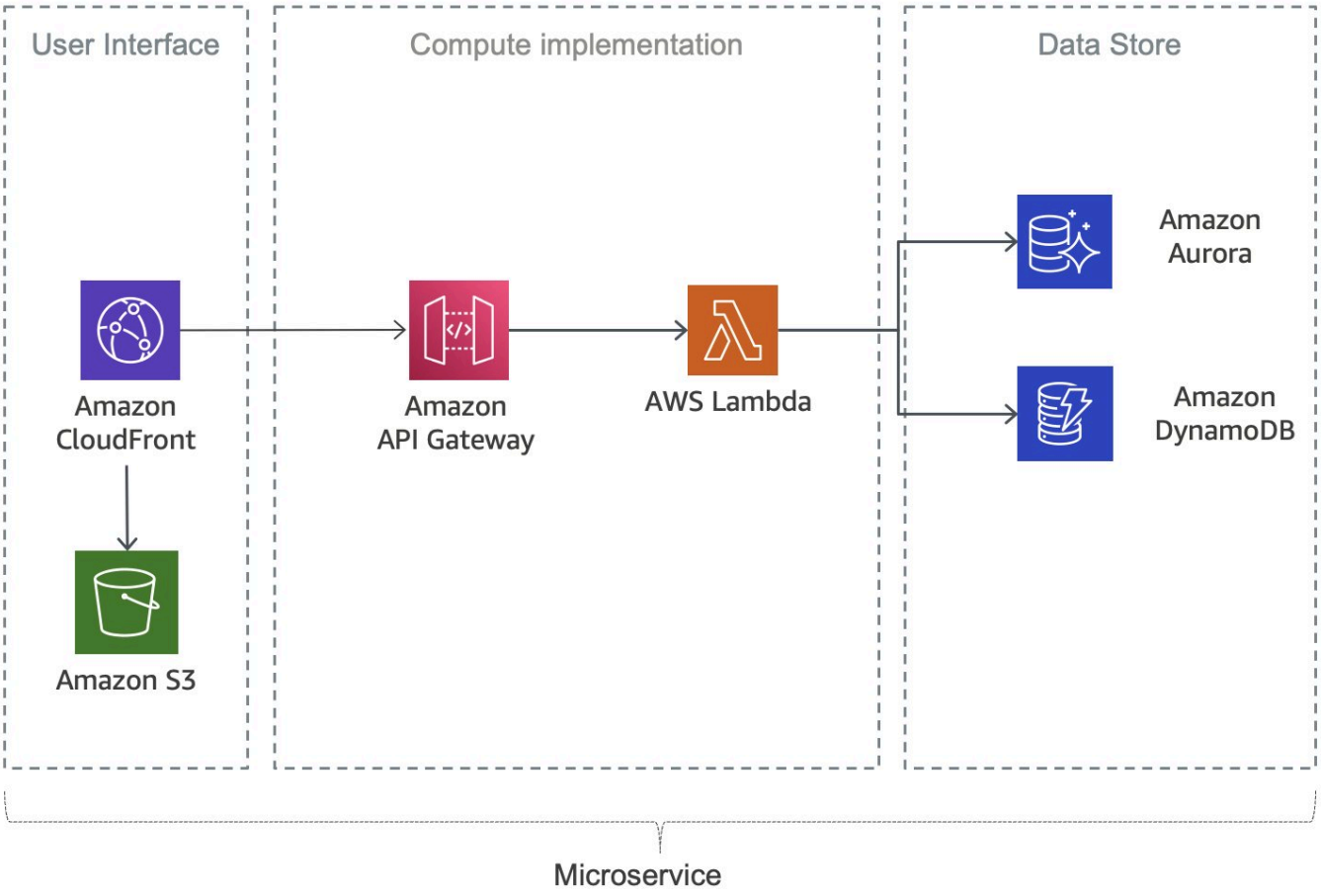

그림 4:를 사용하는 서버리스 마이크로서비스 AWS Lambda

그림 5는가 있는 컨테이너를 사용하는 유사한 서버리스 구현을 표시 AWS Fargate하여 기본 인프라에 대한 우려를 제거합니다. 또한 애플리케이션의 요구 사항에 따라 용량을 자동으로 조정하는 온디맨드 자동 크기 조정 데이터베이스인 Amazon Aurora Serverless가 특징입니다.

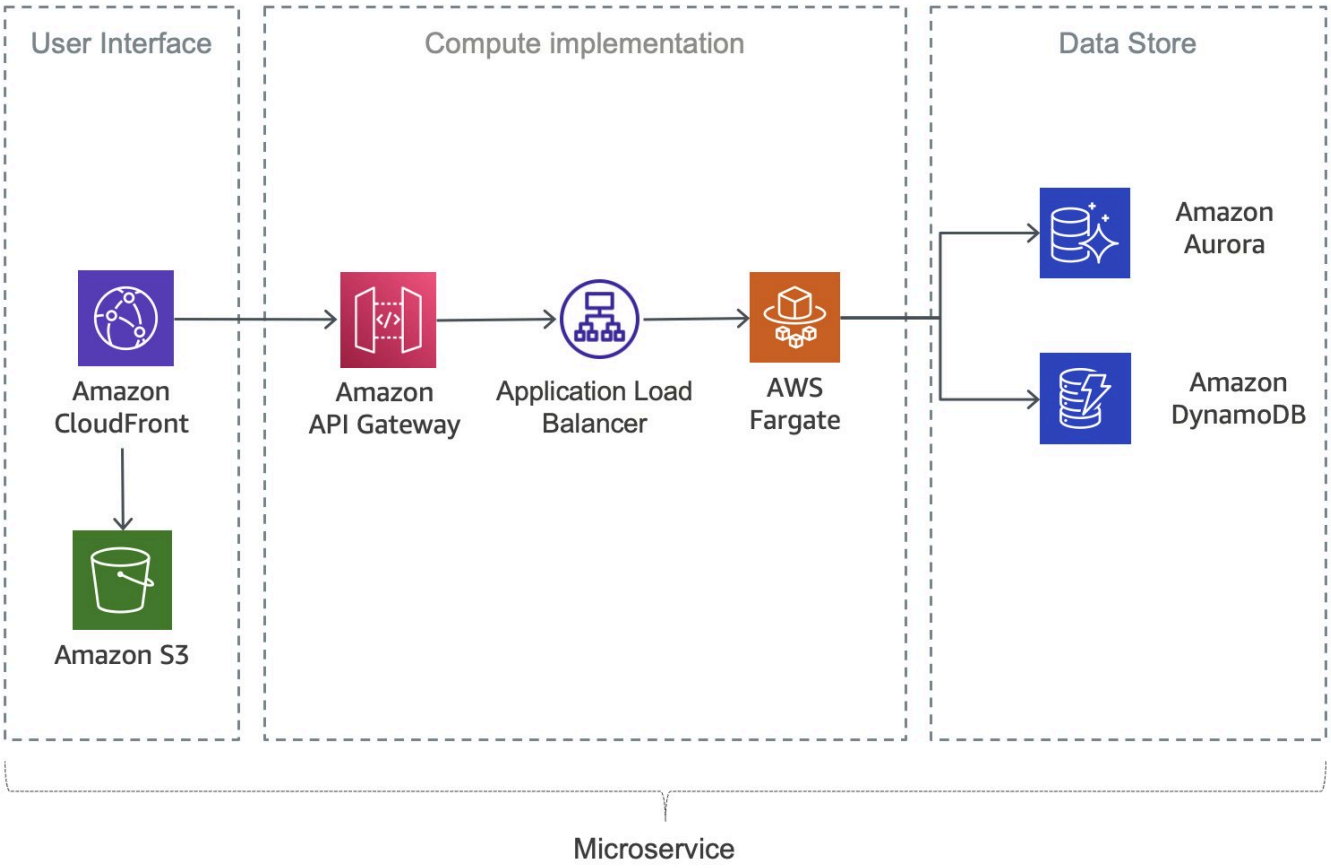

그림 5:를 사용한 서버리스 마이크로서비스 AWS Fargate

탄력적이고 효율적인 시스템

재해 복구(DR)

마이크로서비스 애플리케이션은 프로세스가 상태 비저장이고 영구 데이터가 데이터베이스와 같은 상태 저장 백업 서비스에 저장되는 12단계 애플리케이션 패턴을 따르는 경우가 많습니다. 이렇게 하면 서비스가 실패할 경우 새 인스턴스를 시작하여 기능을 복원하기 쉬우므로 재해 복구(DR)가 간소화됩니다.

마이크로서비스에 대한 재해 복구 전략은 파일 시스템, 데이터베이스 또는 대기열과 같이 애플리케이션의 상태를 유지하는 다운스트림 서비스에 중점을 두어야 합니다. 조직은 복구 시간 목표(RTO) 및 복구 시점 목표(RPO)를 계획해야 합니다. RTO는 서비스 중단과 복원 간에 허용되는 최대 지연이며, RPO는 마지막 데이터 복구 시점 이후의 최대 시간입니다.

재해 복구 전략에 대한 자세한 내용은 클라우드의 재해 워크로드 복구: AWS복구 백서를 참조하세요.

고가용성(HA)

마이크로서비스 아키텍처의 다양한 구성 요소에 대해 고가용성(HA)을 검사합니다.

Amazon EKS는 여러 가용 영역에서 Kubernetes 제어 및 데이터 영역 인스턴스를 실행하여 고가용성을 제공합니다. 비정상 컨트롤 플레인 인스턴스를 자동으로 감지 및 교체하고 자동화된 버전 업그레이드 및 패치를 제공합니다.

Amazon ECR은 스토리지에 Amazon Simple Storage Service(Amazon S3)를 사용하여 컨테이너 이미지를 가용성과 접근성이 높게 만듭니다. Amazon EKS, Amazon ECS 및와 함께 작동하여 프로덕션 워크플로 개발을 AWS Lambda간소화합니다.

Amazon ECS는 리전 내 여러 가용 영역에서 고가용성 방식으로 컨테이너 실행을 간소화하는 리전 서비스로서, 리소스 요구 사항 및 가용성 요구 사항에 맞는 컨테이너를 배치하는 여러 예약 전략을 제공합니다.

AWS Lambda 는 여러 가용 영역에서 작동하므로 단일 영역에서 서비스 중단 시 가용성을 보장합니다. 함수를 VPC에 연결하는 경우 고가용성을 위해 여러 가용 영역에서 서브넷을 지정합니다.

분산 시스템 구성 요소

마이크로서비스 아키텍처에서 서비스 검색은 분산 시스템 내에서 개별 마이크로서비스의 네트워크 위치(IP 주소 및 포트)를 동적으로 찾고 식별하는 프로세스를 말합니다.

접근 방식을 선택할 때는 다음과 같은 요소를 AWS고려하세요.

- 코드 수정: 코드를 수정하지 않고도 이점을 얻을 수 있나요?
- 교차 VPC 또는 교차 계정 트래픽: 필요한 경우 시스템에서 다른 VPCs 간 통신을 효율적으로 관리해야 합니까 AWS 계정? 아니면
- 배포 전략: 시스템이 블루-그린 또는 카나리 배포와 같은 고급 배포 전략을 사용하거나 사용할 계획입니까?
- 성능 고려 사항: 아키텍처가 외부 서비스와 자주 통신하는 경우 전반적인 성능에 어떤 영향을 미칠까요?

AWS 는 마이크로서비스 아키텍처에서 서비스 검색을 구현하기 위한 몇 가지 방법을 제공합니다.

- Amazon ECS Service Discovery:Amazon ECS는 DNS 기반 메서드를 사용하거나와 통합하여 서비스 검색을 지원합니다 AWS Cloud Map (ECS 서비스 검색 참조). ECS Service Connect는 연결 관리를 더욱 개선하여 여러 상호 작용 서비스가 있는 대규모 애플리케이션에 특히 유용할 수 있습니다.
- Amazon Route 53:Route 53은 ECS 및 EKS와 같은 기타 AWS 서비스와 통합되어 서비스 검색을 용이하게 합니다. ECS 컨텍스트에서 Route 53는 Auto Naming API를 활용하여 서비스를 자동으로 등록 및 등록 취소하는 ECS 서비스 검색 기능을 사용할 수 있습니다.
- AWS Cloud Map:이 옵션은 서비스 전체에 변경 사항을 전파하는 동적 API 기반 서비스 검색을 제공합니다.

고급 통신 요구 사항을 위해 Amazon VPC Lattice는 서비스 간의 통신을 일관되게 연결, 모니터링 및 보호하는 애플리케이션 네트워킹 서비스로서, 개발자가 비즈니스에 중요한 기능을 구축하는 데 집중할 수 있도록 생산성을 개선하는 데 도움이 됩니다. 네트워크 트래픽 관리, 액세스 및 모니터링에 대한 정책을 정의하여 인스턴스, 컨테이너 및 서버리스 애플리케이션 전반에서 간소화되고 일관된 방식으로 컴퓨팅 서비스를 연결할 수 있습니다.

HashiCorp Consul 또는 Netflix Eureka와 같은 타사 소프트웨어를 서비스 검색에 이미 사용하고 있는 경우 마이그레이션할 때 이러한 소프트웨어를 계속 사용하여 보다 원활한 전환을 AWS지원하는 것이 좋습니다.

이러한 옵션 중에서 선택하는 것은 특정 요구 사항에 부합해야 합니다. 더 간단한 요구 사항을 위해 Amazon ECS 또는와 같은 DNS 기반 솔루션으로 충분할 AWS Cloud Map 수 있습니다. 더 복잡하거나 더 큰 시스템의 경우 Amazon VPC Lattice와 같은 서비스 메시가 더 적합할 수 있습니다.

결과적으로에서 마이크로서비스 아키텍처를 설계하는 AWS 것은 특정 요구 사항에 맞는 올바른 도구를 선택하는 것입니다. 논의된 고려 사항을 염두에 두고 정보에 입각한 결정을 내려 시스템의 서비스 검색 및 서비스 간 통신을 최적화할 수 있습니다.

분산 데이터 관리

기존 애플리케이션에서는 모든 구성 요소가 단일 데이터베이스를 공유하는 경우가 많습니다. 반면 마이크로서비스 기반 애플리케이션의 각 구성 요소는 자체 데이터를 유지 관리하여 독립성과 분산화를 촉진합니다. 분산 데이터 관리라고 하는이 접근 방식은 새로운 문제를 야기합니다.

이러한 문제 중 하나는 분산 시스템의 일관성과 성능 간의 장단점에서 발생합니다. 즉각적인 업데이트(즉각적인 일관성)를 주장하는 것보다 데이터 업데이트(이벤트 일관성)의 약간의 지연을 수용하는 것이 더 실용적일 수 있습니다.

경우에 따라 비즈니스 운영에는 여러 마이크로서비스가 함께 작동해야 합니다. 한 부분이 실패하면 완료된 일부 작업을 실행 취소해야 할 수 있습니다. Saga 패턴은 일련의 보정 작업을 조정하여 이를 관리하는 데 도움이 됩니다.

마이크로서비스가 동기화 상태를 유지하도록 돕기 위해 중앙 집중식 데이터 스토어를 사용할 수 있습니다. AWS Lambda AWS Step Functions및 Amazon EventBridge와 같은 도구로 관리되는이 스토어는 데이터를 정리하고 중복 제거하는 데 도움이 될 수 있습니다.

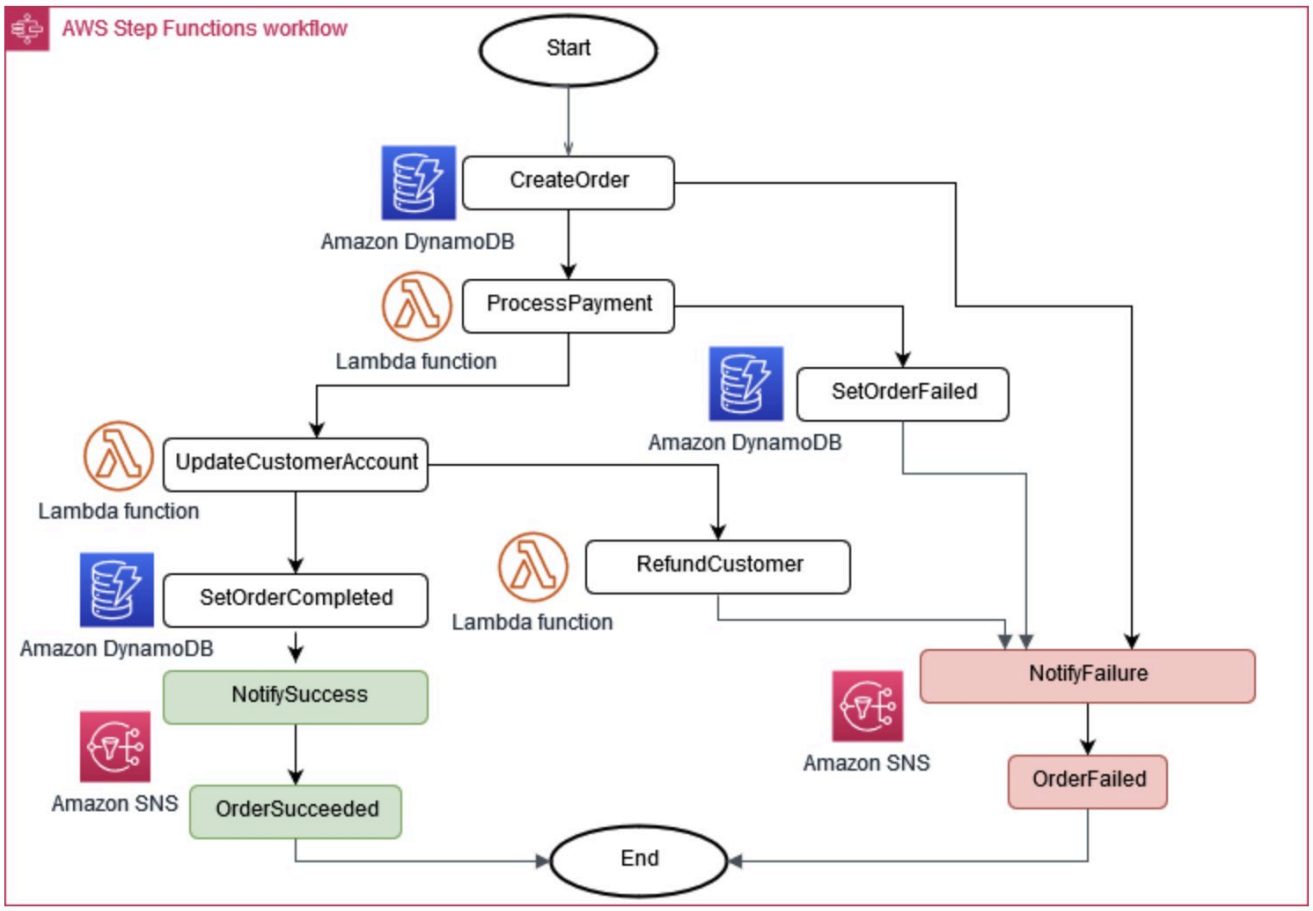

그림 6: Saga 실행 조정자

마이크로서비스 전반의 변경 사항을 관리하는 일반적인 접근 방식은 이벤트 소싱입니다. 애플리케이션의 모든 변경 사항은 이벤트로 기록되어 시스템 상태의 타임라인을 생성합니다. 이 접근 방식은 디버깅 및 감사에 도움이 될 뿐만 아니라 애플리케이션의 다양한 부분이 동일한 이벤트에 대응할 수 있도록 합니다.

이벤트 소싱은 더 나은 성능과 보안을 위해 데이터 수정과 데이터 쿼리를 서로 다른 모듈로 분리하는 명령 쿼리 책임 분리(CQRS) 패턴과 hand-in-hand 작동하는 경우가 많습니다.

에서는 서비스 조합을 사용하여 이러한 패턴을 구현 AWS할 수 있습니다. 그림 7에서 볼 수 있듯이 Amazon Kinesis Data Streams는 중앙 이벤트 스토어 역할을 할 수 있는 반면, Amazon S3는 모든 이벤트 레코드에 대한 내구성 있는 스토리지를 제공합니다. AWS Lambda, Amazon DynamoDB 및 Amazon API Gateway는 이러한 이벤트를 처리하고 처리하기 위해 함께 작동합니다.

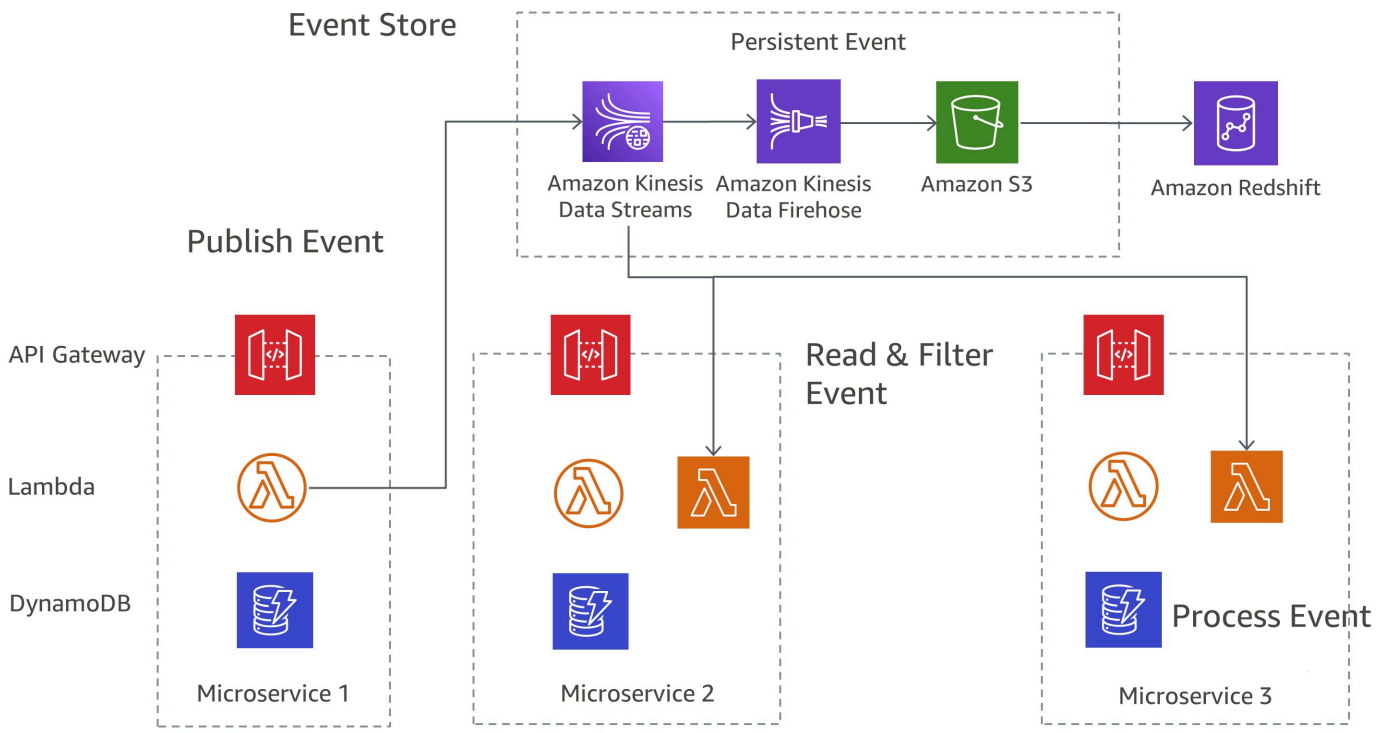

그림 7:의 이벤트 소싱 패턴 AWS

분산 시스템에서는 재시도로 인해 이벤트가 여러 번 전달될 수 있으므로 이를 처리하도록 애플리케이션을 설계하는 것이 중요합니다.

구성 관리

마이크로서비스 아키텍처에서 각 서비스는 데이터베이스, 대기열 및 기타 서비스와 같은 다양한 리소스와 상호 작용합니다. 각 서비스의 연결 및 운영 환경을 구성하는 일관된 방법이 중요합니다. 애플리케이션은 다시 시작할 필요 없이 새 구성에 적응하는 것이 가장 좋습니다. 이 접근 방식은 환경 변수에 구성을 저장하는 것을 권장하는 12단계 앱 원칙의 일부입니다.

다른 접근 방식은 AWS App Config를 사용하는 것입니다. Systems AWS Manager의 기능으로, 고객이 기능 플래그와 애플리케이션 구성을 빠르고 안전하게 구성, 검증 및 배포할 수 있습니다. 배포 전 단계에서 기능 플래그 및 구성 데이터를 구문적으로 또는 의미적으로 검증할 수 있으며, 구성한 경보가 트리거되면 모니터링되고 자동으로 롤백될 수 있습니다. AppConfig 에이전트를 사용하여 AWS AppConfig를 Amazon ECS 및 Amazon EKS와 통합할 수 있습니다. 에이전트는 Amazon ECS 및 Amazon EKS 컨테이너 애플리케이션과 함께 실행되는 사이드카 컨테이너 역할을 합니다. Lambda 함수에서 AppConfig 기능 플래그 또는 기타 동적 구성 데이터를 사용하는 AWS 경우 AWS AppConfig Lambda 확장을 Lambda 함수에 계층으로 추가하는 것이 좋습니다.

GitOps는 Git을 모든 구성 변경의 정확한 소스로 사용하는 구성 관리에 대한 혁신적인 접근 방식입니다. 즉, 구성 파일에 대한 모든 변경 사항은 Git을 통해 자동으로 추적, 버전 관리 및 감사됩니다.

보안 암호 관리

보안이 가장 중요하므로 자격 증명을 일반 텍스트로 전달해서는 안 됩니다.는AWS Systems Manager Parameter Store 및와 같은 보안 서비스를 AWS 제공합니다 AWS Secrets Manager. 이러한 도구는 Amazon EKS의 컨테이너에 보안 암호를 볼륨으로 전송하거나 Amazon ECS에 환경 변수로 전송할 수 있습니다. 에서 AWS Lambda환경 변수는 코드에 자동으로 제공됩니다. Kubernetes 워크플로의 경우 외부 보안 암호 운영자는 AWS Secrets Manager와 같은 서비스에서 직접 보안 암호를 가져와서 해당 Kubernetes 보안 암호를 생성합니다. 이를 통해 Kubernetes 네이티브 구성과 원활하게 통합할 수 있습니다.

비용 최적화 및 지속 가능성

마이크로서비스 아키텍처는 비용 최적화 및 지속 가능성을 향상시킬 수 있습니다. 애플리케이션을 더 작은 부분으로 분할하면 더 많은 리소스가 필요한 서비스만 확장하여 비용과 낭비를 줄일 수 있습니다. 이는 가변 트래픽을 처리할 때 특히 유용합니다. 마이크로서비스는 독립적으로 개발됩니다. 따라서 개발자는 더 작은 업데이트를 수행하고 엔드 투 엔드 테스트에 소요되는 리소스를 줄일 수 있습니다. 업데이트하는 동안 모놀리스가 아닌 특성의 하위 집합만 테스트해야 합니다.

아키텍처의 상태 비저장 구성 요소(로컬 데이터 스토어 대신 외부 데이터 스토어에 상태를 저장하는 서비스)는 AWS 클라우드에서 미사용 EC2 용량을 제공하는 Amazon EC2 스팟 인스턴스를 사용할 수 있습니다. EC2 이러한 인스턴스는 온디맨드 인스턴스보다 비용 효율적이며 중단을 처리할 수 있는 워크로드에 적합합니다. 이렇게 하면 고가용성을 유지하면서 비용을 더욱 절감할 수 있습니다.

격리된 서비스를 사용하면 각 오토 스케일링 그룹에 대해 비용 최적화 컴퓨팅 옵션을 사용할 수 있습니다. 예를 들어 AWS ,Graviton은 ARM 기반 인스턴스에 적합한 워크로드에 대해 비용 효율적인 고성능 컴퓨팅 옵션을 제공합니다.

비용 및 리소스 사용량을 최적화하면 Well-Architected Framework의 지속 가능성 원칙에 따라 환경 영향을 최소화하는 데 도움이 됩니다. AWS Customer Carbon Footprint 도구를 사용하여 탄소 배출량 감소 진행 상황을 모니터링할 수 있습니다. 이 도구는 AWS 사용의 환경 영향에 대한 인사이트를 제공합니다.

통신 메커니즘

마이크로서비스 패러다임에서는 애플리케이션의 다양한 구성 요소가 네트워크를 통해 통신해야 합니다. 이에 대한 일반적인 접근 방식에는 REST 기반, GraphQL 기반, gRPC 기반 및 비동기식 메시징이 포함됩니다.

REST 기반 통신

마이크로서비스 간의 동기 통신에 광범위하게 사용되는 HTTP/S 프로토콜은 종종 RESTful APIs 통해 작동합니다. API Gateway는 트래픽 관리, 권한 부여, 모니터링 및 버전 관리와 같은 작업을 처리하여 백엔드 서비스에 대한 중앙 집중식 액세스 포인트 역할을 하는 API를 구축하는 간소화된 방법을 제공합니다.

GraphQL 기반 통신

마찬가지로 GraphQL은 REST와 동일한 프로토콜을 사용하지만 단일 엔드포인트에 대한 노출을 제한하여 동기 통신을 위한 광범위한 방법입니다. 를 사용하면 AWS 서비스 및 데이터 스토어와 직접 상호 작용하는 GraphQL 애플리케이션을 생성 및 게시하거나 비즈니스 로직을 위한 Lambda 함수를 통합할 AWS AppSync수 있습니다.

gRPC 기반 통신

gRPC는 동기식, 경량, 고성능, 오픈 소스 RPC 통신 프로토콜입니다. gRPC는 HTTP/2를 사용하고 압축 및 스트림 우선 순위 지정과 같은 더 많은 기능을 지원하여 기본 프로토콜을 개선합니다. 이진 인코딩된 Protobuf 인터페이스 정의 언어(IDL)를 사용하므로 HTTP/2 이진 프레이밍을 활용합니다.

비동기식 메시징 및 이벤트 전달

비동기식 메시징을 사용하면 서비스가 대기열을 통해 메시지를 전송하고 수신하여 통신할 수 있습니다. 이를 통해 서비스는 느슨하게 연결된 상태를 유지하고 서비스 검색을 촉진할 수 있습니다.

메시징은 다음 세 가지 유형으로 정의할 수 있습니다.

- 메시지 대기열: 메시지 대기열은 메시지의 발신자(생산자)와 수신자(소비자)를 분리하는 버퍼 역할을 합니다. 생산자는 메시지를 대기열에 대기열에 추가하고 소비자는 메시지를 대기열에서 제거하고 처리합니다. 이 패턴은 비동기 통신, 로드 레벨링 및 트래픽 버스트 처리에 유용합니다.

- 게시-구독: 게시-구독 패턴에서 메시지가 주제에 게시되고 관심 있는 여러 구독자가 메시지를 수신합니다. 이 패턴을 사용하면 이벤트 또는 메시지를 여러 소비자에게 비동기적으로 브로드캐스팅할 수 있습니다.
- 이벤트 기반 메시징: 이벤트 기반 메시징에는 시스템에서 발생하는 이벤트를 캡처하고 이에 대응하는 작업이 포함됩니다. 이벤트는 메시지 브로커에 게시되며 관심 있는 서비스는 특정 이벤트 유형을 구독합니다. 이 패턴을 사용하면 느슨한 결합이 가능하고 서비스가 직접 종속성 없이 이벤트에 대응할 수 있습니다.

이러한 각 메시지 유형을 구현하기 위해는 Amazon SQS, Amazon SNS, Amazon EventBridge, Amazon MQ 및 Amazon MSK와 같은 다양한 관리형 서비스를 AWS 제공합니다. 이러한 서비스에는 특정 요구 사항에 맞게 조정된 고유한 기능이 있습니다.

- Amazon Simple Queue Service(Amazon SQS) 및 Amazon Simple Notification Service(Amazon SNS): 그림 8에서 볼 수 있듯이이 두 서비스는 서로 보완하며, Amazon SQS는 메시지를 저장할 수 있는 공간을 제공하고 Amazon SNS는 여러 구독자에게 메시지를 전달할 수 있습니다. 동일한 메시지를 여러 대상으로 전송해야 할 때 유효합니다.

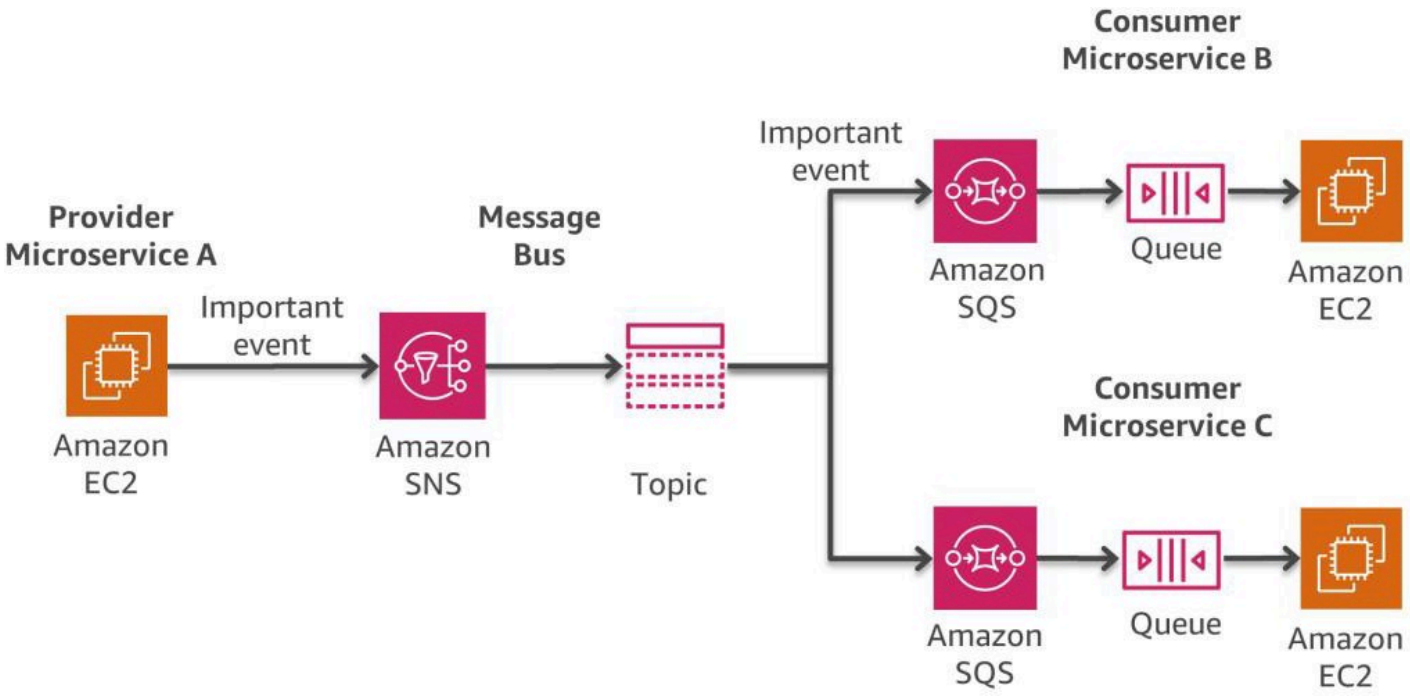

그림 8:의 메시지 버스 패턴 AWS

- Amazon EventBridge: 이벤트를 사용하여 애플리케이션 구성 요소를 함께 연결하는 서버리스 서비스로, 확장 가능한 이벤트 기반 애플리케이션을 더 쉽게 구축할 수 있습니다. 이를 사용하여 자체 개발 애플리케이션, AWS 서비스 및 타사 소프트웨어와 같은 소스에서 조직 전체의 소비자 애플리케이션으로 이벤트를 라우팅할 수 있습니다. EventBridge는 이벤트를 수집, 필터링, 변환 및 전달하는 간

단하고 일관된 방법을 제공하므로 새 애플리케이션을 빠르게 구축할 수 있습니다. EventBridge 이벤트 버스는 이벤트 기반 서비스 간 이벤트의 many-to-many 라우팅에 매우 적합합니다.

- Amazon MQ: JMS, AMQP 등과 같은 표준 프로토콜을 사용하는 기존 메시징 시스템이 있는 경우 좋은 선택입니다. 이 관리형 서비스는 작업을 중단하지 않고 시스템을 대체합니다.
- Amazon MSK(Managed Kafka): 메시지를 저장하고 읽기 위한 메시징 시스템으로, 메시지를 여러 번 처리해야 하는 경우에 유용합니다. 또한 실시간 메시지 스트리밍을 지원합니다.
- Amazon Kinesis: 스트리밍 데이터의 실시간 처리 및 분석. 이를 통해 실시간 애플리케이션을 개발하고 AWS 에코시스템과 원활하게 통합할 수 있습니다.

최상의 서비스는 특정 요구 사항에 따라 달라지므로 각 서비스가 제공하는 내용과 요구 사항에 맞는 방식을 이해하는 것이 중요합니다.

오케스트레이션 및 상태 관리

마이크로서비스 오케스트레이션은 오케스트레이터라고 하는 중앙 구성 요소가 마이크로서비스 간의 상호 작용을 관리하고 조정하는 중앙 집중식 접근 방식을 말합니다. 여러 마이크로서비스에서 워크플로를 오케스트레이션하는 것은 어려울 수 있습니다. 오케스트레이션 코드를 서비스에 직접 임베딩하는 것은 더 엄격한 결합을 도입하고 개별 서비스 교체를 방해하므로 권장되지 않습니다.

Step Functions는 오류 처리 및 직렬화와 같은 서비스 오케스트레이션 복잡성을 관리하는 워크플로 엔진을 제공합니다. 이를 통해 조정 코드를 추가하지 않고도 애플리케이션을 빠르게 확장하고 변경할 수 있습니다. Step Functions는 AWS 서버리스 플랫폼의 일부이며 Lambda 함수, Amazon EC2, Amazon EKS, Amazon ECS, SageMaker AI AWS Glue등을 지원합니다.

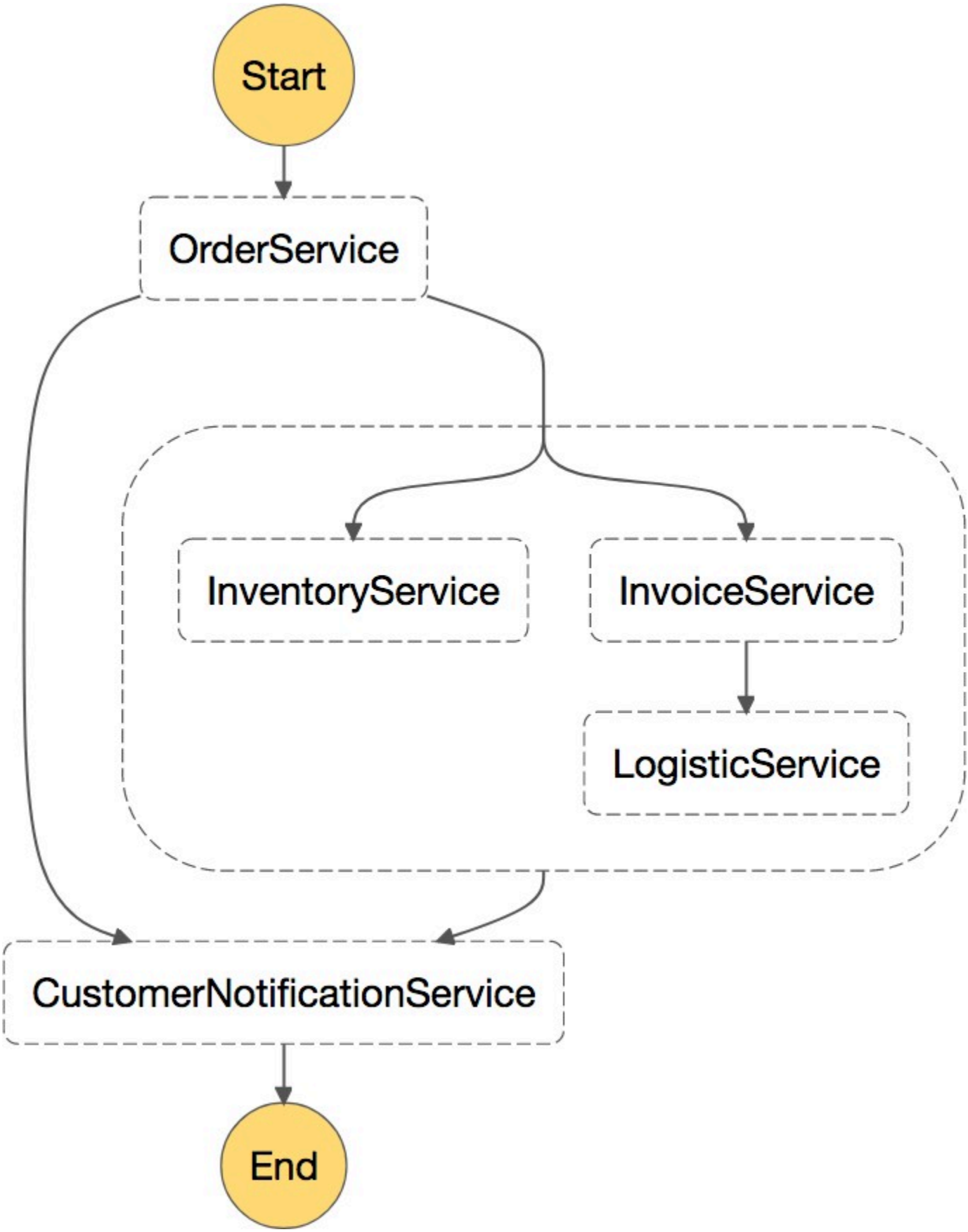
Start
OrderService
InventoryService
InvoiceService
LogisticService
CustomerNotificationService
End

그림 9:에서 호출한 병렬 및 순차 단계가 있는 마이크로서비스 워크플로의 예 AWS Step Functions

Amazon Managed Workflows for Apache Airflow(Amazon MWAA)는 Step Functions의 대안입니다. 오픈 소스 및 이식성을 우선시하는 경우 Amazon MWAA를 사용해야 합니다. Airflow에는 새로운 기능과 통합에 정기적으로 기여하는 대규모의 활성 오픈 소스 커뮤니티가 있습니다.

관찰성

마이크로서비스 아키텍처는 본질적으로 많은 분산 구성 요소로 구성되므로 이러한 모든 구성 요소에서 관찰성이 중요합니다. Amazon CloudWatch를 사용하면 지표를 수집 및 추적하고, 로그 파일을 모니터링하고, AWS 환경의 변경 사항에 대응할 수 있습니다. 애플리케이션 및 서비스에서 생성된 AWS 리소스 및 사용자 지정 지표를 모니터링할 수 있습니다.

주제

- 모니터링
- 로그 중앙 집중화
- 분산 추적
- 에 대한 로그 분석 AWS
- 기타 분석 옵션

모니터링

CloudWatch는 리소스 사용률, 애플리케이션 성능 및 운영 상태에 대한 시스템 전반의 가시성을 제공합니다. 마이크로서비스 아키텍처에서는 개발자가 수집할 지표를 선택할 수 있으므로 CloudWatch를 통한 사용자 지정 지표 모니터링이 유용합니다. 동적 조정은 이러한 사용자 지정 지표를 기반으로 할 수도 있습니다.

CloudWatch Container Insights는이 기능을 확장하여 CPU, 메모리, 디스크 및 네트워크와 같은 많은 리소스에 대한 지표를 자동으로 수집합니다. 컨테이너 관련 문제를 진단하고 해결을 간소화하는 데 도움이 됩니다.

Amazon EKS의 경우 포괄적인 모니터링 및 알림 기능을 제공하는 오픈 소스 플랫폼인 Prometheus가 자주 선호됩니다. 일반적으로 직관적인 지표 시각화를 위해 Grafana와 결합됩니다. Amazon Managed Service for Prometheus(AMP)는 Prometheus와 완벽하게 호환되는 모니터링 서비스를 제공하므로 컨테이너화된 애플리케이션을 손쉽게 감독할 수 있습니다. 또한 Amazon Managed Grafana(AMG)는 지표의 분석 및 시각화를 간소화하므로 기본 인프라를 관리할 필요가 없습니다.

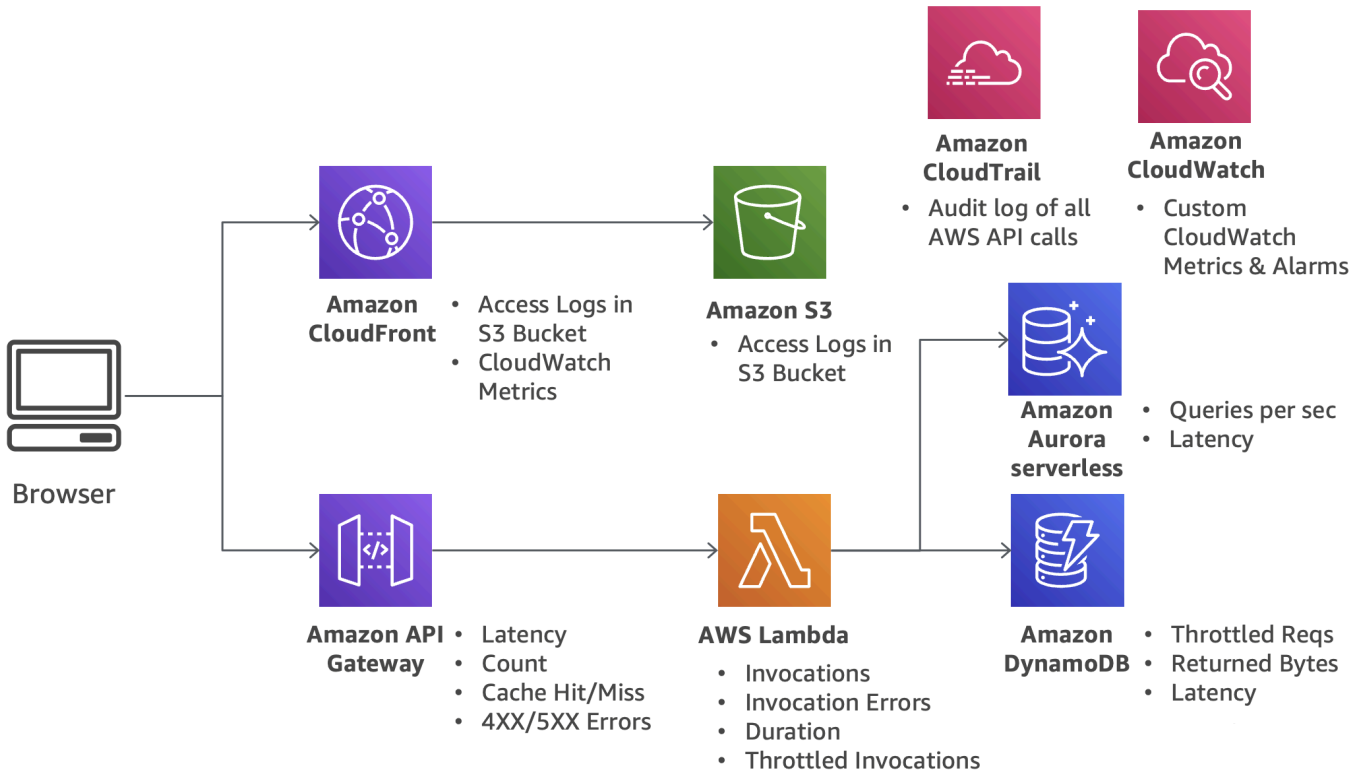

그림 10: 모니터링 구성 요소가 있는 서버리스 아키텍처

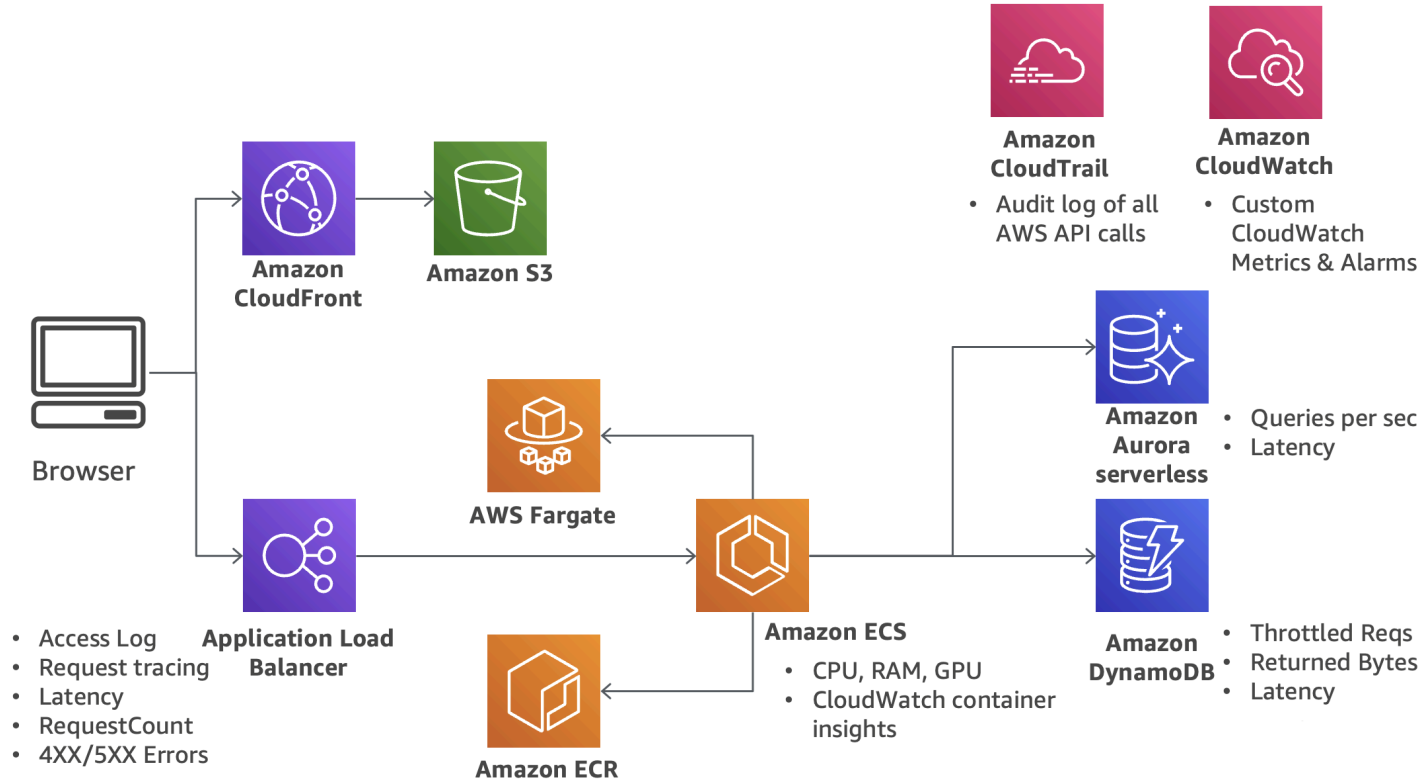

그림 11: 모니터링 구성 요소가 있는 컨테이너 기반 아키텍처

로그 중앙 집중화

로깅은 문제를 정확히 찾아 해결하는 데 중요합니다. 마이크로서비스를 사용하면 더 자주 릴리스하고 새로운 기능을 실험할 수 있습니다.는 Amazon S3, CloudWatch Logs 및 Amazon OpenSearch Service와 같은 서비스를 AWS 제공하여 로그 파일을 중앙 집중화합니다. Amazon EC2는 로그를 CloudWatch로 전송하는 데몬을 사용하는 반면, Lambda와 Amazon ECS는 기본적으로 로그 출력을 CloudWatch로 전송합니다. Amazon EKS의 경우 Fluent Bit 또는 Fluentd를 사용하여 OpenSearch 및 Kibana를 사용하여 보고하기 위해 CloudWatch에 로그를 전달할 수 있습니다. CloudWatch OpenSearch 그러나 설치 공간 및 성능 이점이 작기 때문에 Fluentd보다 Fluent Bit를 사용하는 것이 좋습니다.

그림 12는 다양한 AWS 서비스의 로그가 Amazon S3 및 CloudWatch로 전송되는 방법을 보여줍니다. 이러한 중앙 집중식 로그는 데이터 시각화를 위한 Kibana를 포함한 Amazon OpenSearch Service를 사용하여 추가로 분석할 수 있습니다. 또한 Amazon Athena는 Amazon S3에 저장된 로그에 대한 임시 쿼리에 사용할 수 있습니다.

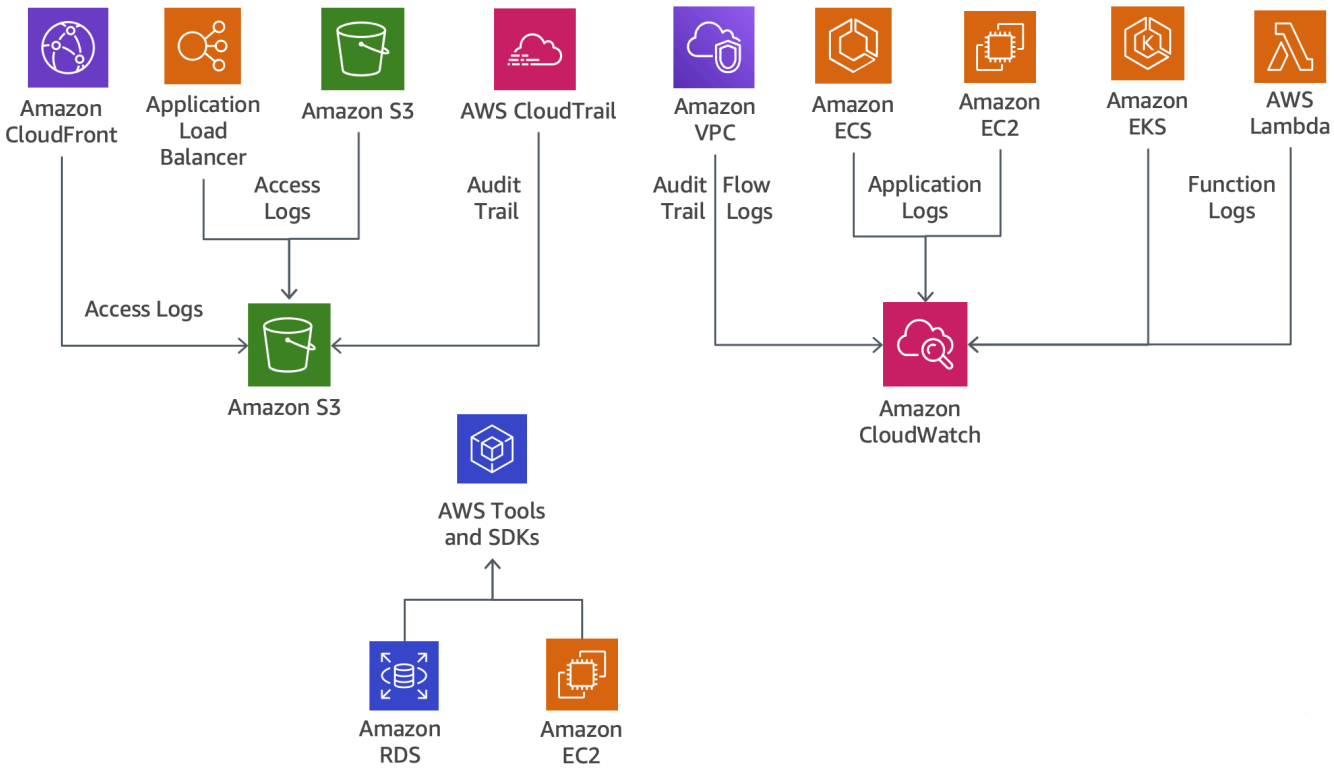

그림 12: AWS 서비스의 로깅 기능

분산 추적

마이크로서비스는 요청을 처리하는 데 함께 작동하는 경우가 많습니다.는 상관관계 IDs AWS X-Ray 사용하여 이러한 서비스 전반의 요청을 추적합니다. X-Ray는 Amazon EC2, Amazon ECS, Lambda 및 Elastic Beanstalk에서 작동합니다.

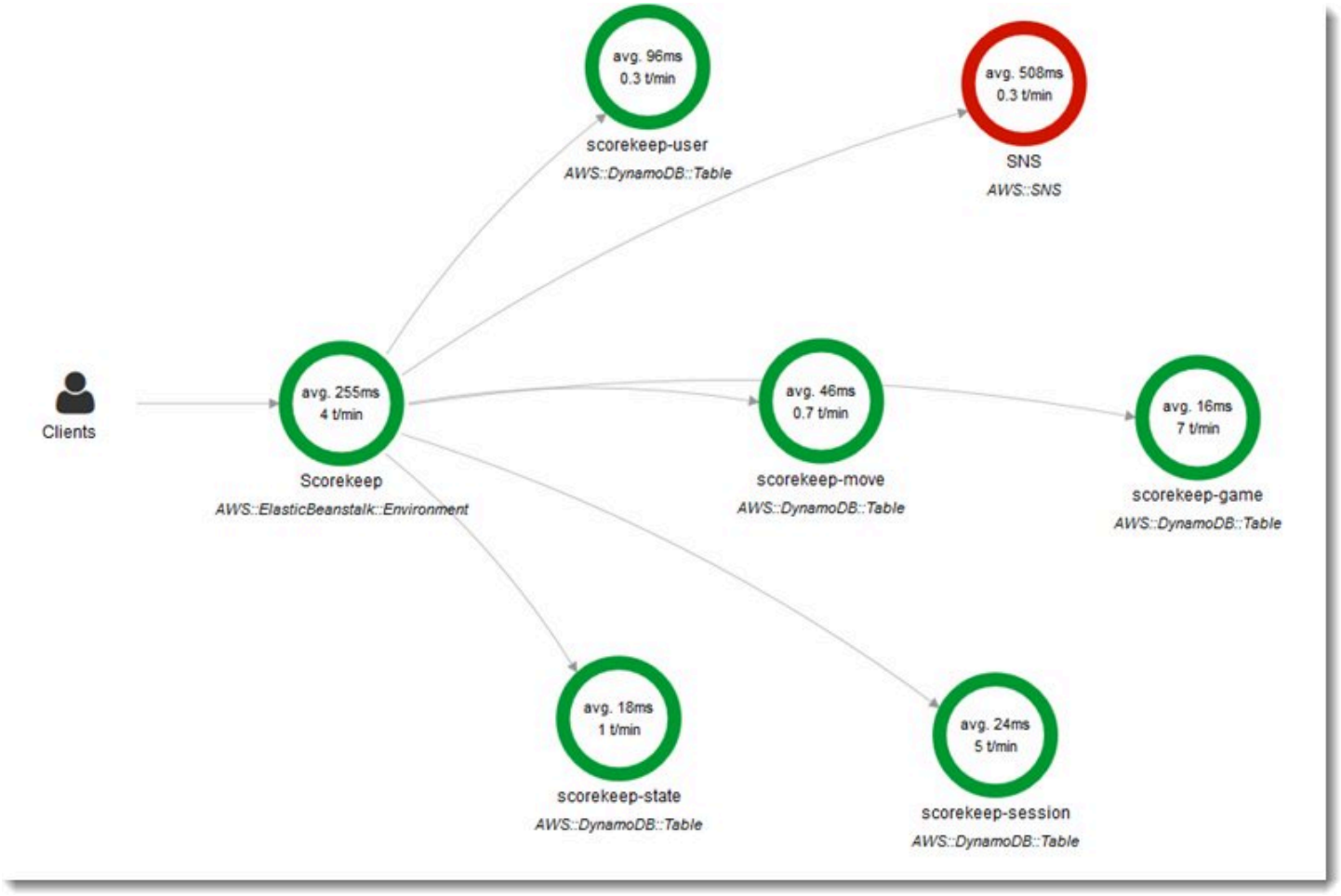

그림 13: AWS X-Ray 서비스 맵

AWS Distro for OpenTelemetry는 OpenTelemetry 프로젝트의 일부이며 분산 추적 및 지표를 수집하여 애플리케이션 모니터링을 개선하기 위한 오픈 소스 APIs 및 에이전트를 제공합니다. 또한 지표 및 추적을 여러 AWS 및 파트너 모니터링 솔루션으로 전송합니다. AWS 리소스에서 메타데이터를 수집하여 애플리케이션 성능을 기본 인프라 데이터에 맞게 조정하여 문제 해결을 가속화합니다. 또한 다양한 AWS 서비스와 호환되며 온프레미스에서 사용할 수 있습니다.

에 대한 로그 분석 AWS

Amazon CloudWatch Logs Insights를 사용하면 실시간 로그 탐색, 분석 및 시각화가 가능합니다. 추가 로그 파일 분석을 위해 Kibana를 포함하는 Amazon OpenSearch Service는 강력한 도구입니다. CloudWatch Logs는 로그 항목을 OpenSearch Service로 실시간으로 스트리밍할 수 있습니다. OpenSearch와 원활하게 통합된 Kibana는이 데이터를 시각화하고 직관적인 검색 인터페이스를 제공합니다.

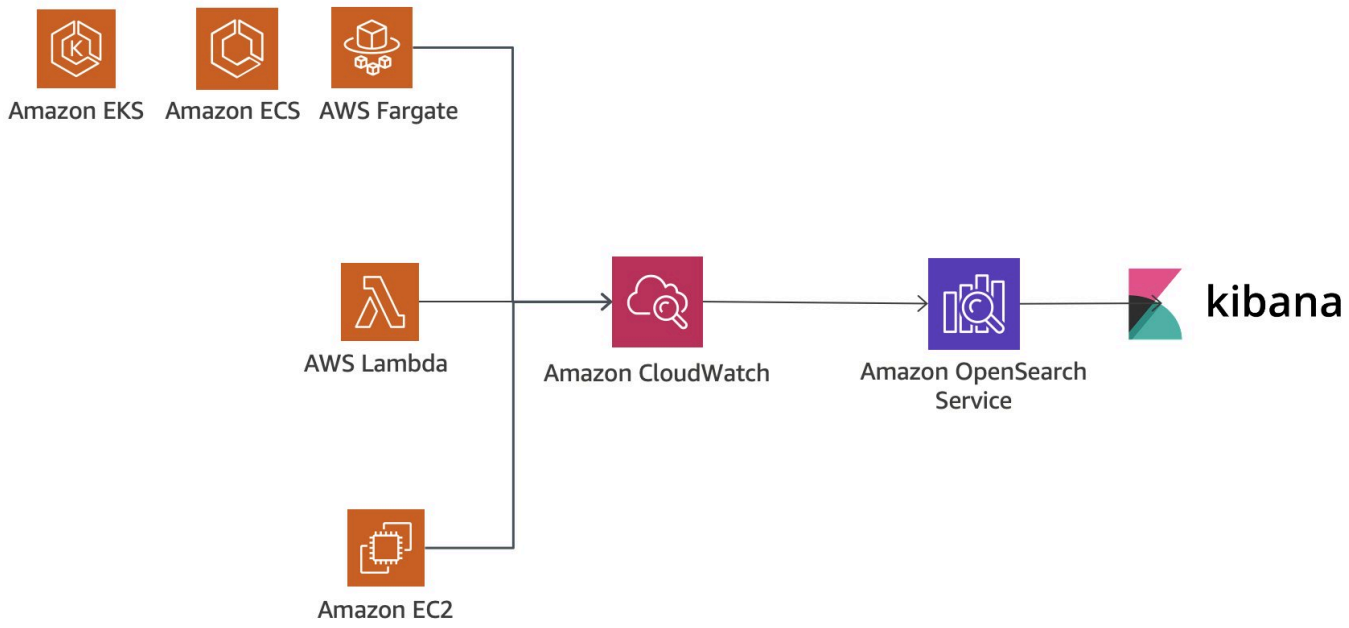

그림 14: Amazon OpenSearch Service를 사용한 로그 분석

기타 분석 옵션

추가 로그 분석을 위해 완전 관리형 데이터 웨어하우스 서비스인 Amazon Redshift와 확장 가능한 비즈니스 인텔리전스 서비스인 Quick은 효과적인 솔루션을 제공합니다. QuickSight는 Redshift, RDS, Aurora, EMR, DynamoDB, Amazon S3, Kinesis와 같은 다양한 AWS 데이터 서비스에 쉽게 연결하여 데이터 액세스를 간소화합니다.

CloudWatch Logs는 실시간 스트리밍 데이터를 제공하는 서비스인 Amazon Data Firehose로 로그 항목을 스트리밍할 수 있습니다. 그런 다음 QuickSight는 Redshift에 저장된 데이터를 사용하여 포괄적인 분석, 보고 및 시각화를 수행합니다.

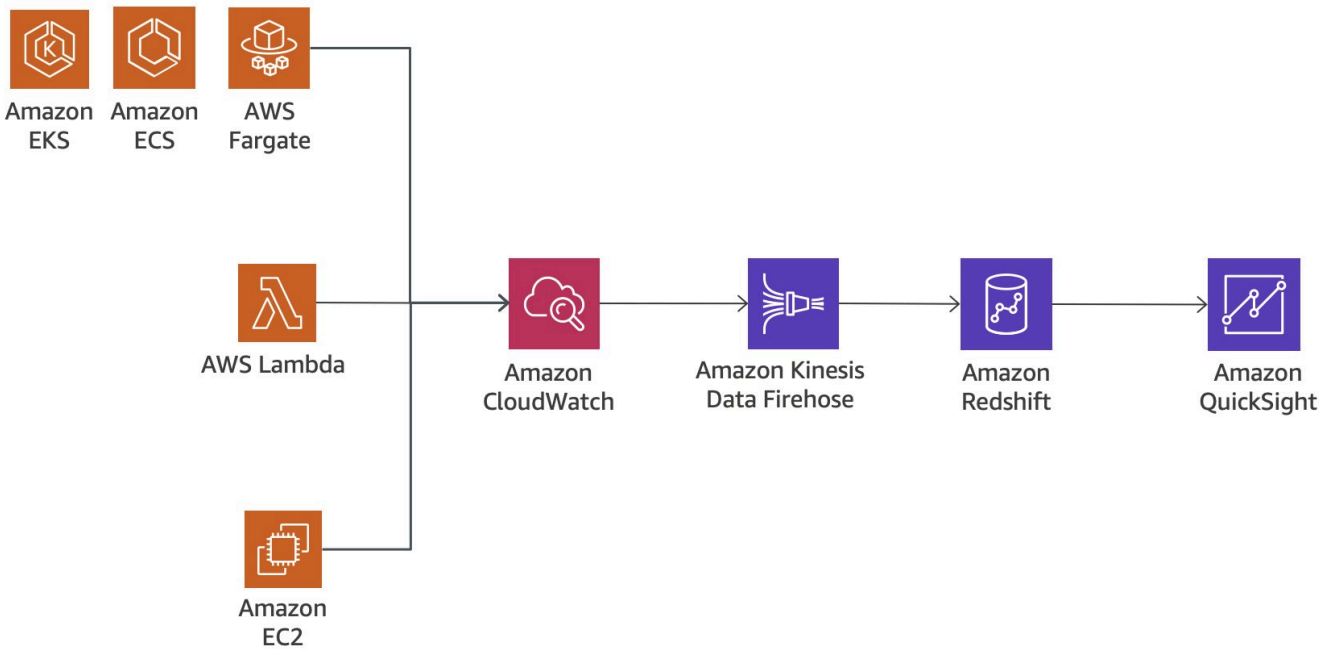

그림 15: Amazon Redshift 및 Quick을 사용한 로그 분석

또한 로그가 객체 스토리지 서비스인 S3 버킷에 저장되면 데이터를 클라우드 기반 빅 데이터 플랫폼인 Redshift 또는 EMR과 같은 서비스에 로드하여 저장된 로그 데이터를 철저히 분석할 수 있습니다.

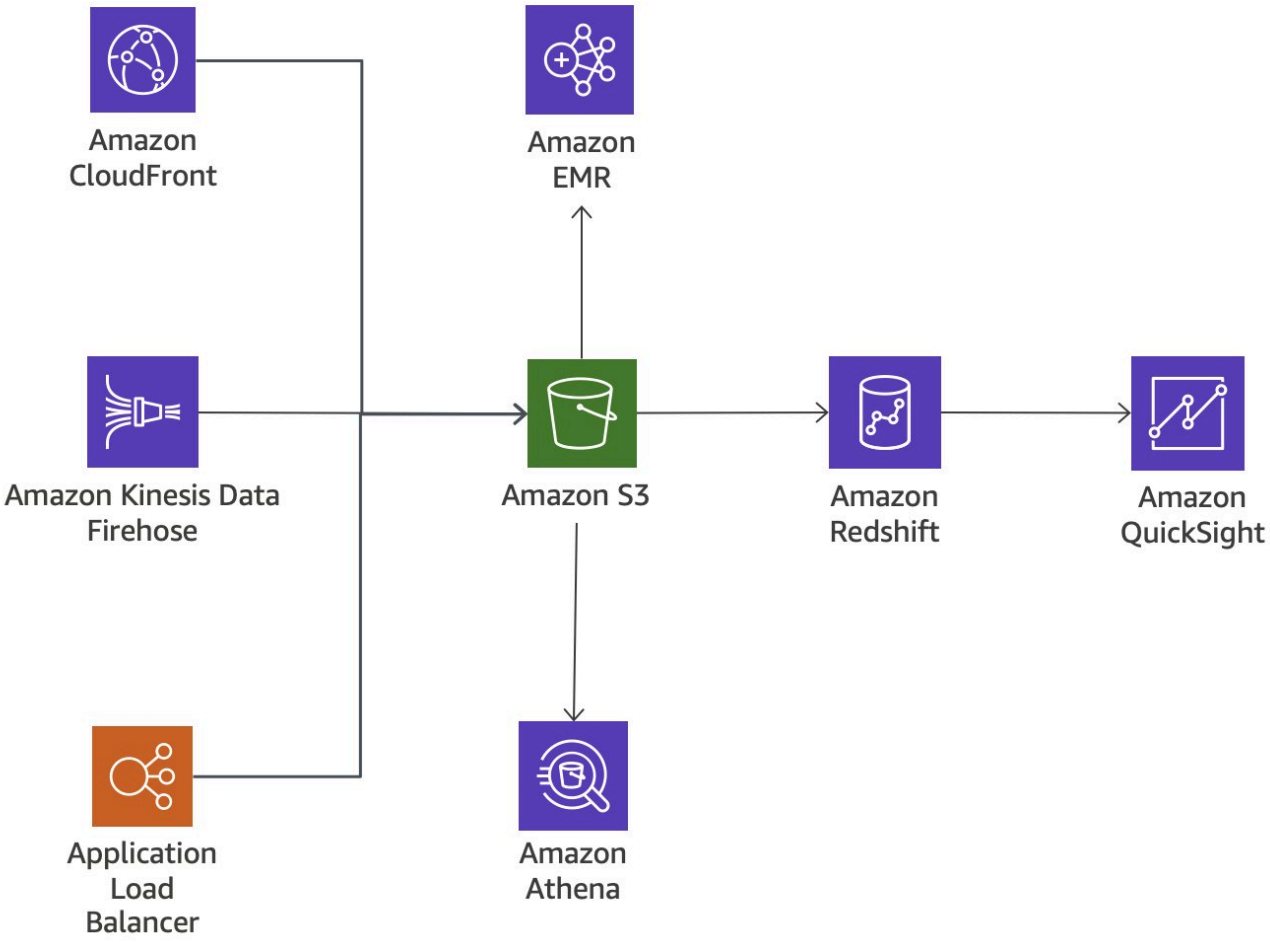

그림 16: 로그 분석 간소화: AWS 서비스에서 QuickSight로

마이크로서비스 통신의 채팅 관리

Chattiness는 마이크로서비스 간의 과도한 통신을 의미하며, 네트워크 지연 시간 증가로 인해 비효율성이 발생할 수 있습니다. 잘 작동하는 시스템의 경우 채팅을 효과적으로 관리하는 것이 중요합니다.

채팅을 관리하기 위한 몇 가지 주요 도구는 REST APIs, HTTP APIs 및 gRPC APIs. REST APIs API 키, 클라이언트별 제한, 요청 검증, AWS WAF 통합 또는 프라이빗 API 엔드포인트와 같은 다양한 고급 기능을 제공합니다. HTTP APIs는 최소한의 기능으로 설계되었으므로 더 저렴한 가격으로 제공됩니다. 이 주제에 대한 자세한 내용과 REST APIs 및 HTTP API에서 사용할 수 있는 핵심 기능 목록은 REST API와 HTTP API 중에서 선택을 APIs 참조하세요. APIs

마이크로서비스는 널리 사용되므로 통신을 위해 HTTP를 통한 REST를 사용하는 경우가 많습니다. 그러나 대용량 상황에서는 REST의 오버헤드로 인해 성능 문제가 발생할 수 있습니다. 이는 통신이 모든 새 요청에 필요한 TCP 핸드셰이크를 사용하기 때문입니다. 이러한 경우 gRPC API가 더 나은 선택입니다. gRPC는 단일 TCP 연결을 통해 여러 요청을 허용하므로 지연 시간을 줄입니다. 또한 gRPC는 양방향 스트리밍을 지원하므로 클라이언트와 서버가 동시에 메시지를 보내고 받을 수 있습니다. 이로 인해 특히 대규모 또는 실시간 데이터 전송의 경우 통신이 더 효율적입니다.

올바른 API 유형을 선택했지만 채팅이 지속되는 경우 마이크로서비스 아키텍처를 재평가해야 할 수 있습니다. 서비스를 통합하거나 도메인 모델을 수정하면 채팅을 줄이고 효율성을 개선할 수 있습니다.

프로토콜 및 캐싱 사용

마이크로서비스는 종종 통신을 위해 gRPC 및 REST와 같은 프로토콜을 사용합니다(의 이전 섹션 참조통신 메커니즘). gRPC는 전송을 위해 HTTP/2를 사용하는 반면, REST는 일반적으로 HTTP/1.1을 사용합니다. gRPC는 직렬화를 위해 프로토콜 버퍼를 사용하는 반면, REST는 일반적으로 JSON 또는 XML을 사용합니다. 지연 시간과 통신 오버헤드를 줄이기 위해 캐싱을 적용할 수 있습니다. Amazon ElastiCache 또는 API Gateway의 캐싱 계층과 같은 서비스는 마이크로서비스 간의 호출 수를 줄이는 데 도움이 될 수 있습니다.

감사

마이크로서비스 아키텍처에서는 모든 서비스의 사용자 작업을 파악하는 것이 중요합니다.는에서 수행된 모든 API 호출을 로깅 AWS CloudTrail하는 AWS와 같은 도구와 애플리케이션 로그를 캡처하는 데 사용되는 AWS CloudWatch를 AWS 제공합니다. 이를 통해 변경 사항을 추적하고 마이크로서비스 전반의 동작을 분석할 수 있습니다. Amazon EventBridge는 시스템 변경에 신속하게 대응하여 적절한 사용자에게 알리거나 워크플로를 자동으로 시작하여 문제를 해결할 수 있습니다.

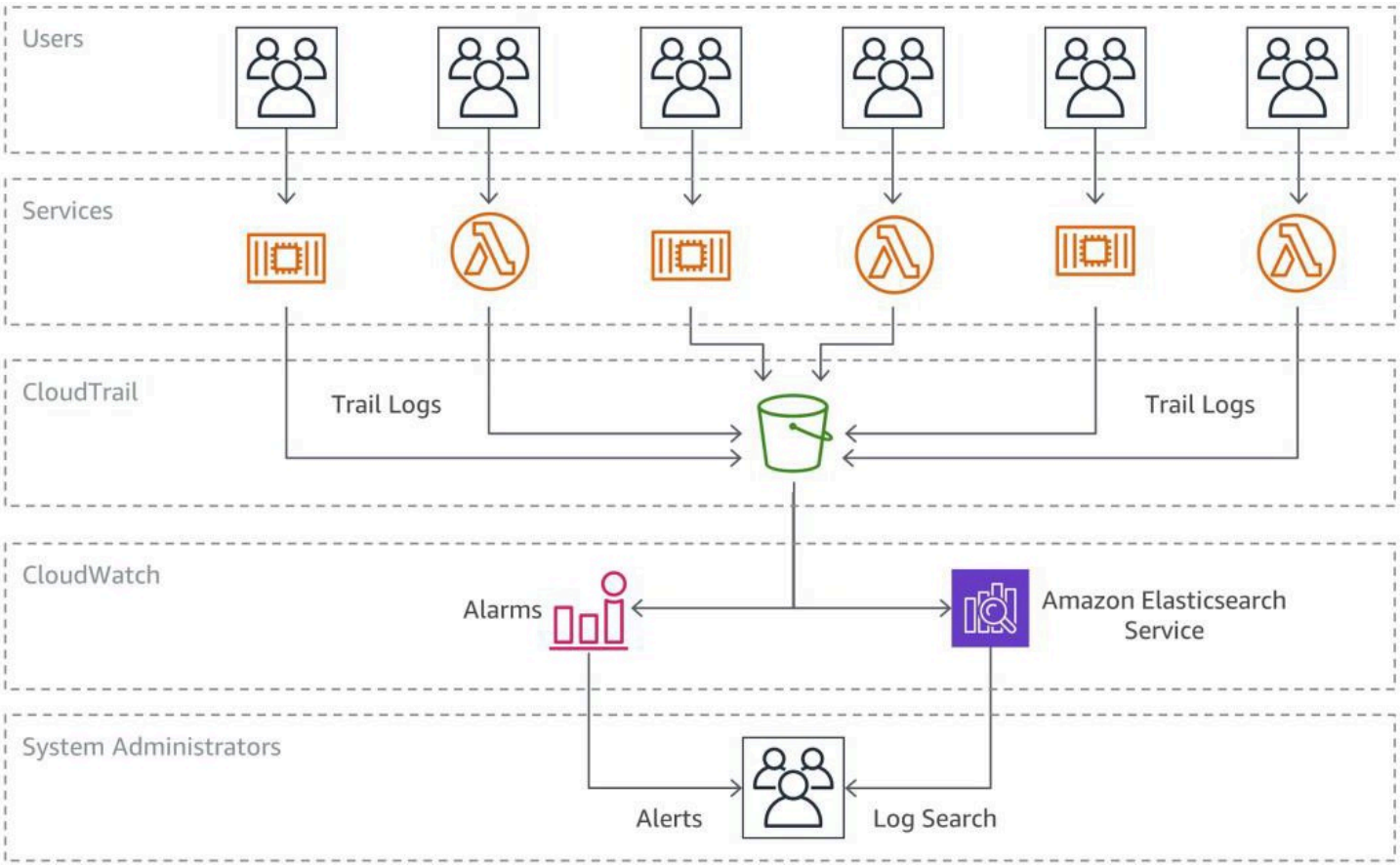

그림 17: 마이크로서비스 전반의 감사 및 문제 해결

리소스 인벤토리 및 변경 관리

빠르게 진화하는 인프라 구성이 있는 민첩한 개발 환경에서는 자동화된 감사 및 제어가 중요합니다. 마이크로서비스 전반에서 이러한 변경 사항을 모니터링하는 관리형 접근 방식을 AWS Config 규칙 제공합니다. 이를 통해 정책 위반에 대한 알림을 자동으로 감지, 추적 및 전송하는 특정 보안 정책을 정의할 수 있습니다.

예를 들어 마이크로서비스의 API Gateway 구성이 HTTPS 요청만 아닌 인바운드 HTTP 트래픽을 허용하도록 변경되면 사전 정의된 AWS Config 규칙이이 보안 위반을 감지할 수 있습니다. 감사에 대한 변경 사항을 기록하고 SNS 알림을 트리거하여 규정 준수 상태를 복원합니다.

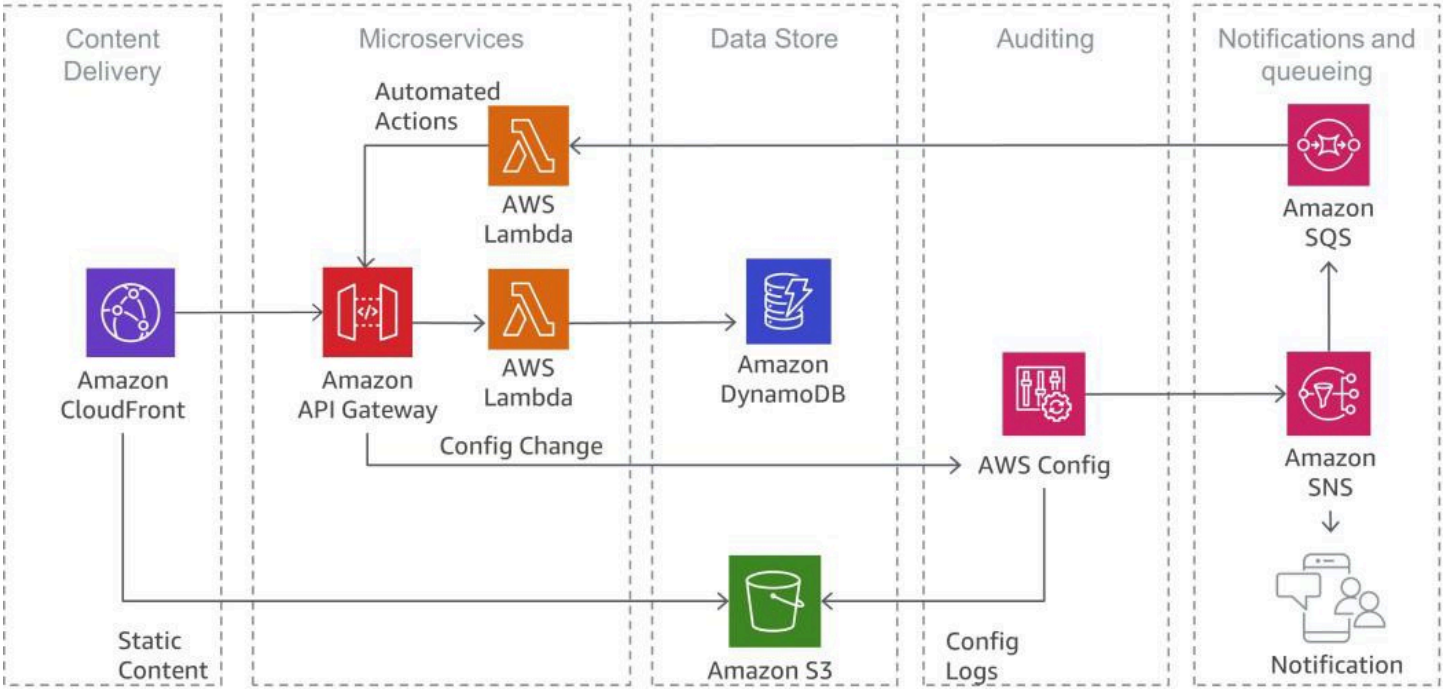

그림 18:를 사용하여 보안 위반 감지 AWS Config

결론

기존 모놀리식 시스템의 대안을 제공하는 다목적 설계 접근 방식인 Microservices 아키텍처는 애플리케이션 규모 조정, 개발 속도 향상, 조직 성장 촉진을 지원합니다. 적응성을 통해 컨테이너, 서버리스 접근 방식 또는 두 가지의 조합을 사용하여 특정 요구 사항에 맞게 조정할 수 있습니다.

그러나 이는 one-size-fits-all 솔루션이 아닙니다. 아키텍처 복잡성 및 운영 요구가 증가할 가능성이 있으므로 각 사용 사례에는 세심한 평가가 필요합니다. 그러나 전략적으로 접근하면 마이크로서비스의 이점이 이러한 문제를 크게 능가할 수 있습니다. 특히 관찰성, 보안 및 변경 관리 영역에서는 사전 예방적인 계획이 중요합니다.

또한 마이크로서비스 외에도 Retrieval Augmented Generation(RAG)과 같은 생성형 AI 아키텍처와 같은 아키텍처 프레임워크가 완전히 다르므로 필요에 가장 적합한 다양한 옵션을 제공합니다.

AWS는 강력한 관리형 서비스 제품군을 통해 팀이 효율적인 마이크로서비스 아키텍처를 구축하고 복잡성을 효과적으로 최소화할 수 있도록 지원합니다. 이 백서는 관련 AWS 서비스와 주요 패턴 구현을 안내하는 것을 목표로 했습니다. 목표는 마이크로서비스의 성능을 활용할 수 있는 지식을 제공하여 이점을 활용하고 애플리케이션 개발 여정을 혁신 AWS하는 것입니다.

기여자

다음 개인과 조직이 이 문서에 기여했습니다.

- Sascha Möllering, 솔루션 아키텍처, Amazon Web Services
- Christian Müller, 솔루션 아키텍처, Amazon Web Services
- MatthiasJun, Solutions Architecture, Amazon Web Services
- Peter Dalbhanjan, 솔루션 아키텍처, Amazon Web Services
- Peter Chapman, Solutions Architecture, Amazon Web Services
- Christoph Kassen, 솔루션 아키텍처, Amazon Web Services
- Umair Ishaq, 솔루션 아키텍처, Amazon Web Services
- Rajiv Kumar, 솔루션 아키텍처, Amazon Web Services
- Ramesh Dwarakanath, 솔루션 아키텍처, Amazon Web Services
- Andrew Watkins, 솔루션 아키텍처, Amazon Web Services
- Yann Stoneman, 솔루션 아키텍처, Amazon Web Services
- Mainak Chaudhuri, 솔루션 아키텍처, Amazon Web Services
- Gaurav Acharya, 솔루션 아키텍처, Amazon Web Services

문서 이력

이 백서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
메이저 업데이트	AWS 고객 탄소 발자국 도구, Amazon EventBridge, AWS AppSync (GraphQL), AWS Lambda 레이어, Lambda SnapStart, 대규모 언어 모델(LLMs), Amazon Managed Streaming for Apache Kafka(MSK), Amazon Managed Workflows for Apache Airflow(MWAA), Amazon VPC Lattice, AWS AppConfig에 대한 정보가 추가되었습니다. 비용 최적화 및 지속 가능성에 대한 별도의 섹션을 추가했습니다.	2023년 7월 31일
마이너 업데이트	추상화에 Well-Architected를 추가했습니다.	2022년 4월 13일
백서 업데이트	Amazon EventBridge, AWS OpenTelemetry, AMP, AMG, Container Insights, 사소한 텍스트 변경의 통합.	2021년 11월 9일
마이너 업데이트	조정된 페이지 레이아웃	2021년 4월 30일
마이너 업데이트	사소한 텍스트 변경.	2019년 8월 1일
백서 업데이트	Amazon EKS, AWS Fargate, Amazon MQ, AWS PrivateLi	2019년 6월 1일

	nk, AWS App Mesh, AWS Cloud Map 통합	
백서 업데이트	AWS Step Functions, AWS X-Ray 및 ECS 이벤트 스트림의 통합.	2017년 9월 1일
최초 게시	AWS에서 마이크로서비스 구현이 게시되었습니다.	2016년 1월 12일

> **Note**
>
> RSS 업데이트를 구독하려면 사용 중인 브라우저에서 RSS 플러그인을 활성화해야 합니다.

고지 사항

고객은 본 문서의 정보를 독립적으로 평가할 책임이 있습니다. 이 문서: (a) 정보 제공 목적으로만 사용되며, (b) 예고 없이 변경될 수 있는 현재 AWS 제품 제공 및 관행을 나타내며, (c) AWS 및 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 약속이나 보장도 생성하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 표현 또는 조건 없이 “있는 그대로” 제공됩니다. 고객에 AWS 대한의 책임과 책임은 AWS 계약에 의해 관리되며,이 문서는 AWS 와 고객 간의 계약의 일부이거나 수정하지 않습니다.

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.