AWS 백서

AWS에서 마이크로서비스 구현

AWS에서 마이크로서비스 구현: AWS 백서

Table of Contents

AWS에서 마이크로서비스 구현

게시 날짜: 2021년 11월 9일(문서 이력 및 기여자)

요약

마이크로서비스는 배포 주기를 단축하고 혁신 및 책임 의식을 강화하며, 소프트웨어 애플리케이션의 유지 관리 및 확장성을 개선하며, 팀이 서로 독립적으로 일할 수 있도록 지원하는 애자일 접근 방식을 사용하여 소프트웨어 및 서비스를 제공하는 조직을 확장하기 위해 생성된 소프트웨어 개발에 대한 아키텍처 및 조직적 접근 방식입니다. 마이크로서비스 접근 방식을 사용할 경우 소프트웨어는 독립적으로 배포 가능한 정의가 잘된 애플리케이션 프로그래밍 인터페이스(API)를 통해 통신하는 여러 개의 작은 서비스로 구성됩니다. 이러한 서비스는 자율적인 소규모 팀에서 소유하게 됩니다. 이러한 애자일 접근 방식은 조직을 성공적으로 확장하는 데에서 핵심적인 역할을 합니다.

AWS 고객이 마이크로서비스를 구축할 때 API 기반, 이벤트 기반 및 데이터 스트리밍이라는 세 가지 공통 패턴이 관찰되었습니다. 본 백서에서는 세 가지 방식을 모두 소개하고, 마이크로서비스의 공통적인 특징을 알아보고, 마이크로서비스를 구축하는 데 해결해야 할 주요 과제를 제품 팀이 Amazon Web Services(AWS)를 사용하여 어떻게 해결할 수 있는지를 설명합니다.

본 백서에 설명된 데이터 스토어, 비동기 통신 및 서비스 검색과 같은 다양한 주제의 특성이 다소 영향을 주기 때문에 아키텍처를 선택하기 전에 제공된 지침과 사용자별 애플리케이션의 특정 요구 사항 및 사용 사례를 함께 고려하는 것이 좋습니다.

소개

마이크로서비스 아키텍처는 소프트웨어 엔지니어링 분야에서 새로 등장한 개념이 아니라 이미 성공 사례를 통해 검증된 다음과 같은 다양한 개념의 조합입니다.

- 애자일 소프트웨어 개발
- 서비스 중심 아키텍처
- API 위주 디자인
- 지속적 통합 및 지속적 전달(CI/CD)

마이크로서비스에는 Twelve-Factor App의 설계 패턴이 많이 사용됩니다.

본 백서에서는 먼저 확장성과 내결함성이 뛰어난 마이크로서비스 아키텍처(사용자 인터페이스, 마이크로서비스 구현 및 데이터 스토어)의 다양한 측면을 설명하고 컨테이너 기술을 활용하여 AWS를 기반

으로 마이크로서비스를 구축하는 방법을 알아봅니다. 그런 다음 운영의 복잡성을 줄이기 위해 일반적인 서버리스 마이크로서비스 아키텍처를 구현하는 데 적합한 AWS 서비스를 추천합니다.

서버리스란 다음을 원칙으로 하는 운영 모델이라고 정의할 수 있습니다.

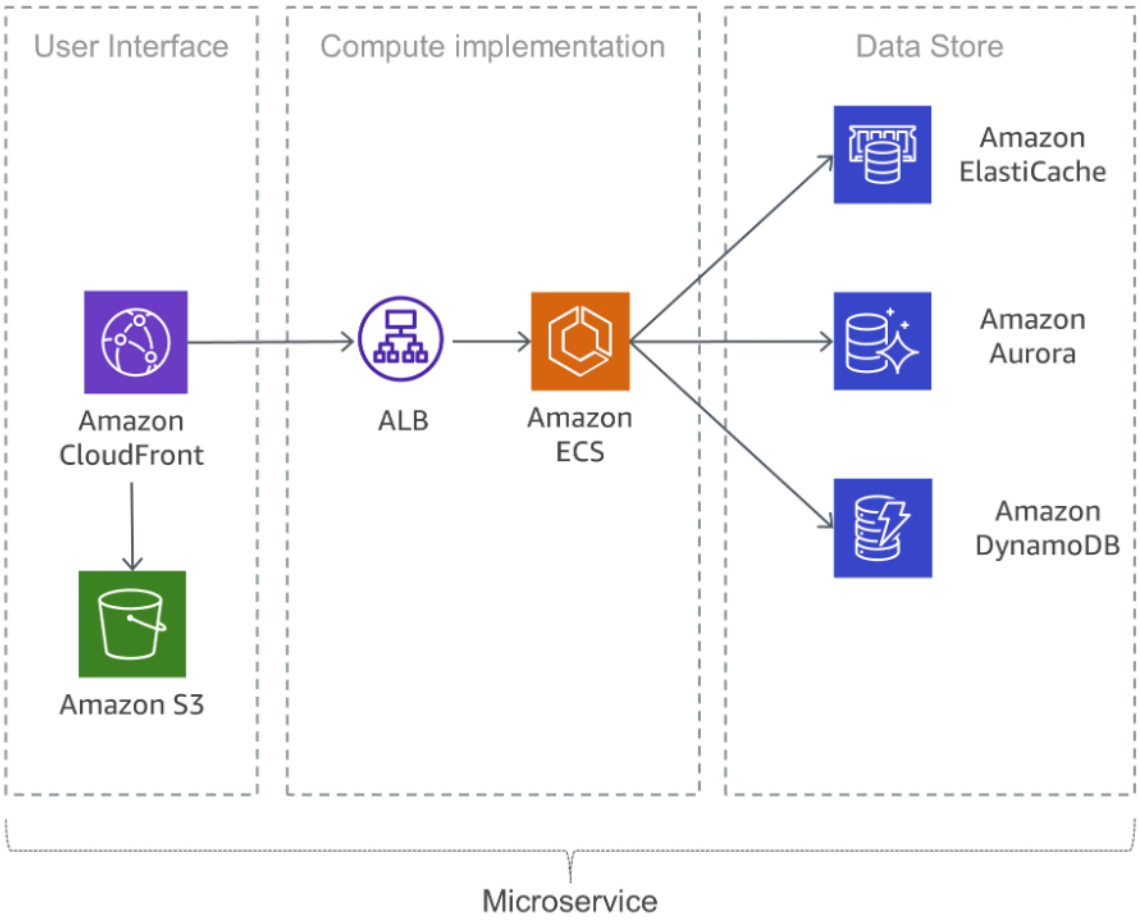
- 프로비저닝 또는 관리할 인프라가 없음
- 소비 단위를 기준으로 자동 확장
- 가치에 대한 비용 지불 결제 모델
- 기본적으로 보장되는 가용성 및 내결함성

마지막으로 본 백서에서는 전반적인 시스템을 살펴보고 분산 모니터링과 감사, 데이터 일관성, 비동기식 통신 등의 마이크로서비스 아키텍처 서비스 간의 기능을 설명합니다.

본 백서는 AWS 클라우드에서 실행되는 워크로드만 집중적으로 다룹니다. 하이브리드 시나리오나 마이그레이션 전략은 다루지 않습니다. 마이그레이션에 대한 자세한 내용은 컨테이너 마이그레이션 방법론 백서를 참조하세요.

AWS 기반의 마이크로서비스 아키텍처

과거에는 일반적인 모놀리식 애플리케이션이 사용자 인터페이스(UI) 계층, 비즈니스 계층 및 지속성 계층과 같은 다양한 계층을 사용하여 구축되었습니다. 마이크로서비스 아키텍처의 중심이 되는 원리는 기술 계층이 아니라 특정 도메인을 구현하여 기능을 서로 연관된 여러 분야로 나누는 것입니다. 다음 그림은 AWS를 기반으로 한 일반적인 마이크로서비스 애플리케이션의 참조 아키텍처를 보여줍니다.



AWS 기반의 일반적인 마이크로서비스 애플리케이션

주제

- 사용자 인터페이스
- 마이크로서비스
- 데이터 스토어

사용자 인터페이스

현대적 웹 애플리케이션의 경우 JavaScript 프레임워크를 사용하여 REST(Representational State Transfer) 또는 RESTful API를 통해 통신하는 단일 페이지 애플리케이션을 주로 구현합니다. Amazon Simple Storage Service(S3) 및 Amazon CloudFront를 사용하여 정적 웹 콘텐츠를 제공할 수 있습니다.

마이크로서비스의 클라이언트는 가장 가까운 엣지 로케이션에서 서비스를 제공받고 오리진에 대한 최적화된 연결을 통해 캐시 또는 프록시 서버에서 응답을 받으므로 대기 시간이 크게 감소할 수 있습니다. 그러나 서로 가까이에서 실행되는 마이크로서비스는 콘텐츠 전송 네트워크의 혜택을 받지 못합니다. 경우에 따라 이 접근 방식은 실제로는 추가 지연 시간을 증가시킬 수 있습니다. 다른 캐싱 메커니즘과 구현하여 통신 수를 줄이고 지연 시간을 최소화하는 것이 좋습니다. 자세한 내용은 the section called “통신 수” 주제를 참조하세요.

마이크로서비스

API는 마이크로서비스의 프런트 도어입니다. 즉, API는 프로그래밍 방식 인터페이스 세트(일반적으로 RESTful 웹 서비스 API) 뒤에 있는 애플리케이션 로직의 진입점 역할을 합니다. 이 API는 클라이언트의 호출을 수락하고 처리하며 트래픽 관리, 요청 필터링, 라우팅, 캐싱, 인증 및 권한 부여와 같은 기능을 구현할 수 있습니다.

마이크로서비스 구현

AWS는 마이크로서비스의 개발을 지원하는 빌딩 블록을 통합했습니다. 인기 있는 두 가지의 접근 방식은 AWS Lambda를 사용하는 방식과 Docker 컨테이너를 AWS Fargate와 함께 사용하는 방식입니다.

AWS Lambda를 사용할 경우 코드를 업로드하기만 하면 뛰어난 가용성으로 실제 수요 변화에 맞춰 서비스를 실행하고 확장하는 데 필요한 모든 작업을 Lambda에서 자동으로 처리합니다. 인프라 관리가 필요하지 않습니다. Lambda는 여러 프로그래밍 언어를 지원하며 다른 AWS 서비스에서 호출되거나 웹 또는 모바일 애플리케이션에서 직접 호출될 수 있습니다. AWS Lambda의 가장 큰 장점 중 하나는 비즈니스 환경의 변화에 신속하게 대응할 수 있다는 점입니다. 즉, AWS에서 보안과 확장이 관리되므로 고객은 비즈니스 로직에 집중할 수 있습니다. Lambda만의 편향적 접근 방식 덕분에 확장성이 뛰어난 플랫폼이 실현됩니다.

배포와 관련한 운영 작업의 부담을 줄이는 일반적인 접근 방식은 컨테이너 기반 배포입니다. Docker와 같은 컨테이너 기술은 지난 몇 년 동안 이식성, 생산성 및 효율성과 같은 이점 때문에 인기를 끌었습니다. 컨테이너는 익히기가 어려울 수 있고 Docker 이미지와 모니터링의 보안 수정 사항도 고려해야 합니다. Amazon Elastic Container Service(Amazon ECS) 및 Amazon Elastic Kubernetes

Service(Amazon EKS)로 인해 자체 클러스터 관리 인프라를 설치, 운영, 확장할 필요가 없습니다. API 호출을 통해 Docker를 사용하는 애플리케이션을 시작 및 중지하고, 클러스터의 전체 상태를 쿼리하며, 보안 그룹, 로드 밸런싱, Amazon Elastic Block Store(Amazon EBS) 볼륨, AWS Identity and Access Management(IAM) 역할 등 여러 가지 익숙한 기능에 액세스할 수 있습니다.

AWS Fargate는 Amazon ECS와 Amazon EKS 모두에서 작동하는 컨테이너용 서버리스 컴퓨팅 엔진입니다. Fargate를 사용하면 컨테이너 애플리케이션을 위한 충분한 컴퓨팅 리소스 프로비저닝에 대해 더 이상 걱정할 필요가 없습니다. Fargate는 수만 개의 컨테이너를 시작하고 사용자의 작업에 매우 중요한 애플리케이션의 실행을 위해 쉽게 확장할 수 있습니다.

Amazon ECS는 컨테이너 배치 전략과 작업 배치와 종료 방식을 사용자 지정 가능한 제약 조건을 지원합니다. 작업 배치 제약 조건은 작업 배치 시에 고려되는 규칙입니다. 기본적으로 키-값 페어인 속성을 컨테이너 인스턴스에 연결한 다음 제약 조건을 사용하여 이 같은 속성을 기준으로 작업을 배치할 수 있습니다. 예를 들어 제약 조건을 사용하여 GPU로 구동되는 인스턴스와 같은 인스턴스 유형 또는 인스턴스 기능을 기준으로 특정 마이크로서비스를 배치할 수 있습니다.

Amazon EKS는 오픈 소스 Kubernetes 소프트웨어의 최신 버전을 실행하므로 Kubernetes 커뮤니티의 모든 기존 플러그 인과 도구를 사용할 수 있습니다. Amazon EKS에서 실행되는 애플리케이션은 온프레미스 데이터 센터 혹은 공공 클라우드에서 실행되는 모든 표준 Kubernetes 환경에서 실행되는 애플리케이션과 완전하게 호환됩니다. Amazon EKS는 IAM을 Kubernetes와 통합하여 사용자가 Kubernetes의 기본 인증 시스템이 포함되어 있는 IAM 엔터티에 등록할 수 있도록 해줍니다. Kubernetes 제어 영역으로 인증하기 위해 수동으로 자격 증명을 설정할 필요가 없습니다. IAM 통합에서는 IAM을 사용하여 제어 영역 자체로 직접 인증하고 Kubernetes 제어 영역의 퍼블릭 엔드포인트를 세부적으로 제어할 수 있습니다.

Amazon ECS 및 Amazon EKS에서 사용하는 Docker 이미지는 Amazon Elastic Container Registry(Amazon ECR)에 저장할 수 있습니다. Amazon ECR을 사용하면 컨테이너 레지스트리를 지원하는 데 필요한 인프라를 운영 및 확장할 필요가 없습니다.

지속적 통합 및 지속적 전달(CI/CD)은 DevOps 이니셔티브의 모범 사례이자 가장 중요한 부분으로, 시스템 안전성과 보완성을 유지하면서 소프트웨어를 신속하게 변경할 수 있습니다. 그러나 이는 본 백서의 범위를 벗어납니다. 자세한 내용은 AWS에서의 지속적 통합 및 지속적 전달 적용 백서를 참조하세요.

프라이빗 링크

AWS PrivateLink는 Virtual Private Cloud(VPC)를 지원되는 AWS 서비스, 다른 AWS 계정에서 호스팅되는 서비스(VPC 엔드포인트 서비스), 지원되는 AWS Marketplace 파트너 서비스에 프라이빗으로 연결할 수 있는 가용성과 확장성이 뛰어난 기술입니다. 인터넷 게이트웨이, NAT(Network Address

Translation) 디바이스, 퍼블릭 IP 주소, AWS Direct Connect 연결 또는 VPN 연결 없이도 서비스와 통신할 수 있습니다. VPC와 서비스 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

프라이빗 링크는 마이크로서비스 아키텍처의 격리 및 보안을 강화하는 좋은 방법입니다. 예를 들어 마이크로서비스는 완전히 분리된 VPC에 배포되고, 로드 밸런서가 앞에 배치되고, AWS PrivateLink 엔드포인트를 통해 다른 마이크로서비스에 노출될 수 있습니다. 이 설정에서는 AWS PrivateLink를 사용하여 마이크로서비스와 주고받는 네트워크 트래픽이 퍼블릭 인터넷을 통과하지 않습니다. 이러한 격리의 한 가지 사용 사례에는 PCI, HIPPA 및 EU/US US Privacy Shield와 같은 민감한 데이터를 처리하는 서비스에 대한 규정 준수가 포함됩니다. 또한 AWS PrivateLink에서는 방화벽 규칙, 경로 정의 또는 라우팅 테이블 없이도 여러 계정과 Amazon VPC에 걸쳐 서비스를 연결할 수 있으므로 네트워크 관리가 간소화됩니다. PrivateLink를 활용하여 서비스형 소프트웨어(SaaS) 제공업체 및 ISV는 완벽한 운영 격리 및 보안 액세스를 갖춘 마이크로서비스 기반 솔루션을 제공할 수 있습니다.

데이터 스토어

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

애플리케이션 서버와 데이터베이스 사이에 캐시를 배치하는 것은 데이터베이스에 대한 읽기 로드를 줄여 쓰기를 지원하는 데 리소스를 더 많이 사용할 수 있도록 하는 일반적인 메커니즘입니다. 캐시를 사용하면 대기 시간을 개선할 수 있습니다.

관계형 데이터베이스는 정형 데이터와 비즈니스 객체를 저장하는 데 여전히 많이 사용됩니다. AWS는 Amazon Relational Database Service(Amazon RDS)를 통해 관리형 서비스로 6개의 데이터베이스 엔진(Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL 및 Amazon Aurora)을 제공합니다.

그러나 관계형 데이터베이스는 무한한 확장이 가능하도록 설계되지 않았으므로, 대량의 쿼리를 지원하도록 기술을 적용하는 작업은 매우 어렵고 시간이 오래 걸릴 수 있습니다.

NoSQL 데이터베이스는 관계형 데이터베이스의 일관성보다 확장성, 성능 및 가용성을 높이도록 설계되었습니다. NoSQL의 한 가지 중요한 요소는 NoSQL 데이터베이스가 일반적으로 엄격한 스키마를 적용하지 않는다는 것입니다. 데이터는 수평으로 확장할 수 있는 파티션에 분산되며 파티션 키를 사용하여 검색됩니다.

개별 마이크로서비스는 한 가지 작업을 잘 수행하도록 설계되었으므로 일반적으로 NoSQL 지속성에 적합한 단순화된 데이터 모델을 사용합니다. NoSQL 데이터베이스는 관계형 데이터베이스와 액세스 패턴이 다르다는 것을 이해하는 것이 중요합니다. 예를 들어, 테이블 조인을 수행할 수 없습니다. 테이블 조인이 필요한 경우, 애플리케이션에 해당 로직을 구현해야 합니다. Amazon DynamoDB를

사용하여 데이터 규모에 관계없이 데이터를 저장 및 검색하고, 어떤 수준의 요청 트래픽이라도 처리할 수 있는 데이터베이스 테이블을 생성할 수 있습니다. DynamoDB는 한 자릿수 밀리초 성능을 제공하지만 마이크로초의 응답 시간을 필요로 하는 일부 사용 사례가 있습니다. Amazon DynamoDB Accelerator(DAX)는 데이터 액세스에 캐싱 기능을 제공합니다.

DynamoDB는 또한 실제 트래픽에 대해 응답하여 처리량을 동적으로 조절하는 자동 확장 기능을 제공합니다. 그러나 애플리케이션에서 활동이 단기간에 급증함으로 인해 용량 계획이 어렵거나 불가능한 사례가 있습니다. 이러한 상황을 위해 DynamoDB는 단순한 요청당 지불 방식을 제공하는 온디멘드 옵션을 제공합니다. DynamoDB 온디맨드는 용량 계획 없이 초당 수천 개의 요청을 즉각적으로 처리할 수 있습니다.

운영 복잡성 최소화

본 백서의 앞에서 설명한 아키텍처는 이미 관리형 서비스를 사용하고 있지만 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에 대한 관리는 계속되어야 합니다. 완전한 서버리스 아키텍처를 사용하면 마이크로서비스를 실행하고 유지 관리하고 모니터링하는 데 필요한 운영 작업을 더 줄일 수 있습니다.

주제

- API 구현
- 서버리스 마이크로서비스
- 재해 복구
- 고가용성
- Lambda 기반 애플리케이션 배포

API 구현

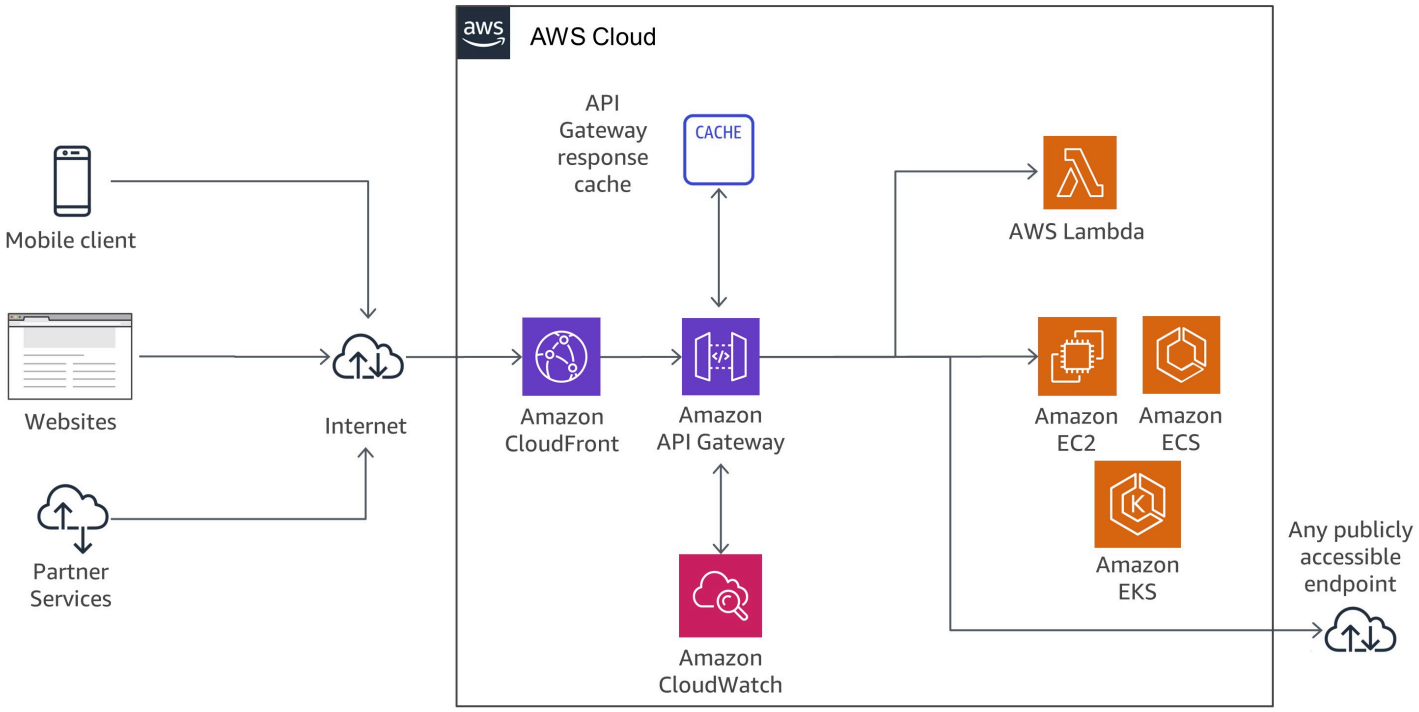
API를 설계하고 배포하고 모니터링하고 지속적으로 개선하며 유지 관리하는 작업에는 시간이 많이 소요될 수 있습니다. 모든 클라이언트를 지원하기 위해 이전 버전과의 호환성을 보장하려는 목적으로 API의 여러 버전을 실행해야 하는 경우도 있습니다. 개발 주기의 다양한 단계(예: 개발, 테스트 및 프로덕션) 때문에 몇 배의 운영 노력이 필요합니다.

권한 부여는 모든 API에 중요한 기능이지만 일반적으로 구축하기가 복잡하고 반복적인 작업을 요구합니다. API가 게시되고 성공적으로 적용되면, 다음 과제는 해당 API를 활용하는 서드 파티 개발자의 에코시스템을 관리하고 모니터링하고 수익화하는 것이 됩니다.

다른 중요한 기능 및 과제로는 백엔드 서비스를 보호하기 위한 요청 제한, API 응답 캐싱, 요청 및 응답 변환 처리, Swagger 등의 도구를 사용한 API 정의 및 문서 생성 등이 있습니다.

Amazon API Gateway는 이러한 문제를 해결하고 RESTful API를 생성하고 유지 관리하는 운영상의 복잡성을 줄입니다. API Gateway는 AWS API 또는 AWS 관리 콘솔을 사용하여 Swagger 정의를 가져와 API를 프로그래밍 방식으로 생성할 수 있도록 지원합니다. API Gateway는 Amazon EC2, Amazon ECS, AWS Lambda 또는 모든 온프레미스 환경에서 실행되는 모든 웹 애플리케이션에 대한 프런트 도어 역할을 합니다. 기본적으로 API Gateway는 서버 관리 없이 API를 실행할 수 있게 합니다.

다음 그림은 API Gateway가 어떻게 API 호출을 처리하고 다른 구성 요소와 상호 작용하는지 보여줍니다. 모바일 디바이스, 웹 사이트 또는 기타 백엔드 서비스에서 발생한 요청은 가장 가까운 CloudFront 상호 접속 위치(POP)로 라우팅되어 대기 시간을 최소화하고 최적의 사용자 경험을 제공합니다.



API Gateway 호출 흐름

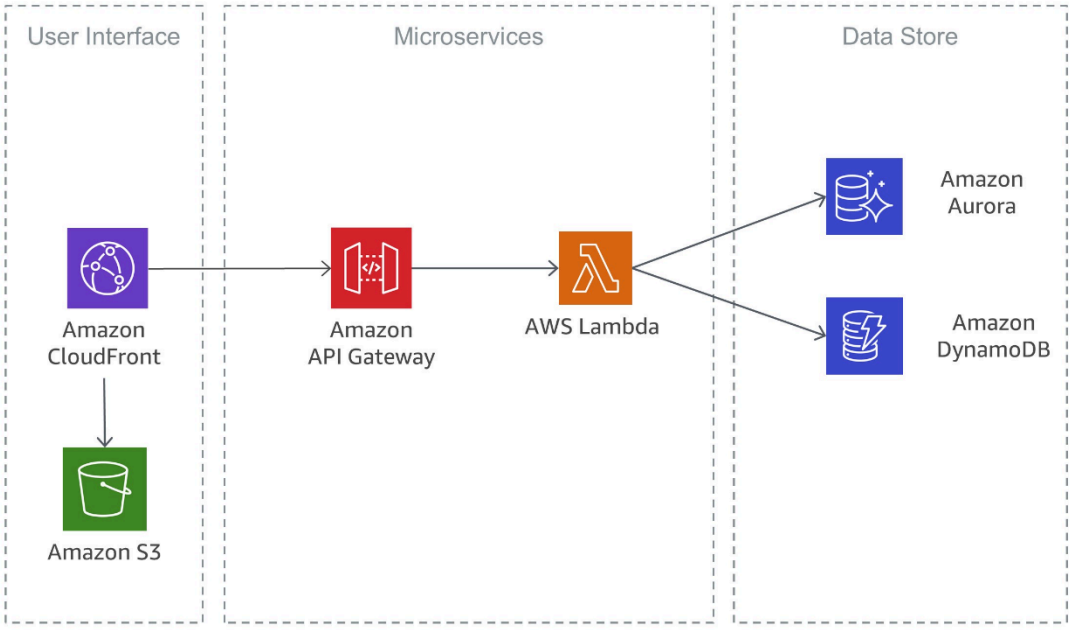
서버리스 마이크로서비스

"서버가 없는 것이 가장 간단한 서버 관리 방법입니다".

운영 복잡성을 해소하는 효과적인 방법은 서버를 없애는 것입니다.

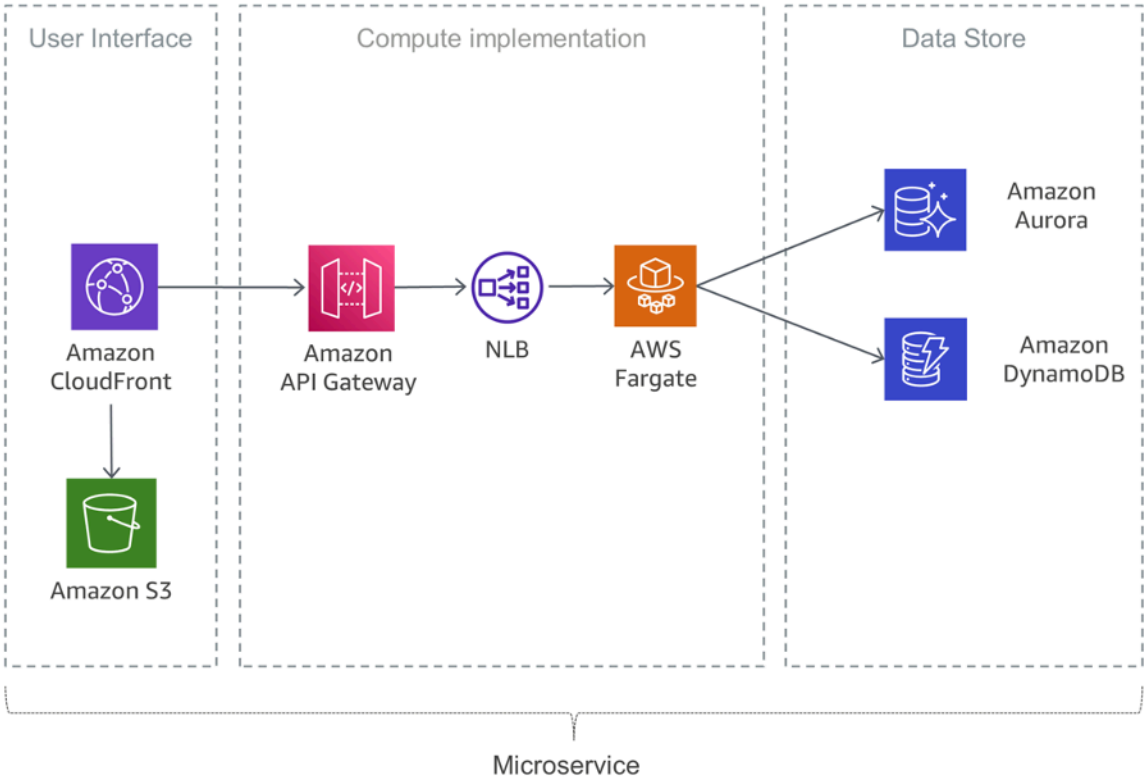
Lambda는 API Gateway와 긴밀하게 통합됩니다. API Gateway에서 Lambda로 동기식 호출을 수행할 수 있으므로 완전한 서버리스 애플리케이션을 생성할 수 있습니다. 자세한 내용은 Amazon API Gateway 개발자 안내서에 나와 있습니다.

다음 그림은 전체 서비스가 관리형 서비스로 구축되는 AWS Lambda를 사용한 서버리스 마이크로서비스의 아키텍처를 보여줍니다. 이 아키텍처는 확장 및 고가용성을 보장하도록 아키텍처를 설계해야 하는 부담이 없고 마이크로서비스의 기반 인프라를 실행하고 모니터링해야 하는 운영상의 작업 부담도 없습니다.



AWS Lambda를 사용한 서버리스 마이크로서비스

다음 그림에는 서버리스 서비스를 기반으로 하는 유사한 구현이 나와 있습니다. 이 아키텍처에서는 Docker 컨테이너가 Fargate와 함께 사용되므로 기반 인프라에 신경 쓸 필요가 없습니다. DynamoDB 외에 Amazon Aurora의 온디맨드 자동 크기 조정 구성(MySQL 호환 버전)인 Amazon Aurora Serverless도 사용되었습니다. 이 구성에서는 애플리케이션의 요구 사항에 따라 자동으로 데이터베이스가 시작 및 종료되고 데이터베이스 용량이 확장 및 축소됩니다.



Fargate를 사용한 서버리스 마이크로서비스

재해 복구

본 백서 앞부분인 소개에서 언급한 것처럼 일반적인 마이크로서비스 애플리케이션은Twelve-Factor App 패턴을 사용하여 구현됩니다. 프로세스 섹션에는 “Twelve-factor 프로세스는 무상태이며 공유가 없습니다. 유지해야 하는 모든 데이터는 상태 저장 지원 서비스(일반적으로 데이터베이스)에 저장해야 합니다.”라고 언급되어 있습니다.

일반적인 마이크로서비스 아키텍처의 경우 재해 복구의 주요 초점은 애플리케이션의 상태를 유지 관리하는 다운스트림 서비스에 있어야 함을 의미합니다. 이러한 서비스의 예로 파일 시스템, 데이터베이스, 대기열 등이 있습니다. 재해 복구 전략을 수립할 때 조직은 일반적으로 복구 시간 목표 및 복구 지점 목표를 계획합니다.

복구 시간 목표는 서비스 중단 시점과 서비스 복원 시점 간에 허용되는 최대 지연 시간입니다. 이 목표는 서비스 불가능 상태가 허용되는 기간을 고려하여 결정되며, 조직에서 정의합니다.

복구 시점 목표는 마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이 목표는 마지막 복구 시점과 서비스 중단 시점 사이에 허용되는 데이터 손실량을 고려하여 결정되며, 조직에서 정의합니다.

자세한 내용은 AWS에서 워크로드의 재해 복구: 클라우드에서의 복구 백서를 참조하세요.

고가용성

이 섹션에서는 다양한 컴퓨팅 옵션에 대한 고가용성을 자세히 살펴봅니다.

Amazon EKS에서는 여러 가용 영역에서 Kubernetes 제어 및 데이터 영역 인스턴스를 실행하여 고가용성을 보장합니다. Amazon EKS에서는 비정상 제어 영역 인스턴스를 자동으로 검색하여 교체하고 이에 대한 버전 업그레이드 및 패치를 자동으로 실행합니다. 이 제어 영역은 한 리전 내 3개의 가용 영역에서 실행되는 3개의 etcd 노드와 2개 이상의 API 서버 인스턴스로 구성됩니다. Amazon EKS는 AWS 리전의 아키텍처를 사용하여 고가용성을 유지합니다.

Amazon ECR은 이미지를 가용성이 뛰어난 고성능 아키텍처에 호스팅함으로써 컨테이너 애플리케이션용의 이미지를 가용 영역 전체에 안정적으로 배포할 수 있습니다. Amazon ECR은 Amazon EKS, Amazon ECS 및 AWS Lambda와 통합되어 개발에서 프로덕션까지의 워크플로를 간소화할 수 있습니다.

Amazon ECS는 AWS 리전 내의 여러 가용 영역에 걸쳐 고가용성 방식으로 애플리케이션 컨테이너를 실행하는 과정을 간소화하는 리전 서비스입니다. Amazon ECS에는 리소스에 필요한 사항(예: CPU 또는 RAM)과 가용성 요구 사항에 따라 클러스터 전체에 컨테이너를 배치하는 여러 일정 예약 전략이 포함되어 있습니다.

AWS Lambda는 함수를 여러 가용 영역에서 실행하여 단일 영역 내 서비스 중단 발생 시 이벤트를 처리할 수 있도록 합니다. 사용자의 계정에서 사용자의 함수가 Virtual Private Cloud(VPC)로 연결되도록 구성하는 경우에는 여러 가용 영역의 서브넷을 지정하여 고가용성을 보장합니다.

Lambda 기반 애플리케이션 배포

AWS CloudFormation을 사용하여 서버리스 애플리케이션을 정의, 배포 및 구성할 수 있습니다.

AWS Serverless Application Model(AWS SAM)은 서버리스 애플리케이션을 정의하는 편리한 방법입니다. AWS SAM은 기본적으로 CloudFormation에서 지원되며 서버리스 리소스를 표현하기 위한 간소화된 구문을 정의합니다. 애플리케이션을 배포하려면 CloudFormation 템플릿에서 관련 권한 정책과 함께 애플리케이션의 일부로서 필요한 리소스를 지정하고, 배포 아티팩트를 패키징한 후 템플릿을 배포하면 됩니다. AWS SAM을 기반으로 하는 SAM Local은 서버리스 애플리케이션을 Lambda 런타임에 업로드하기 전에 로컬로 개발, 테스트 및 분석할 수 있는 환경을 제공하는 AWS Command Line Interface(AWS CLI) 도구입니다. AWS SAM Local을 사용하여 AWS 런타임 환경을 시뮬레이션하는 로컬 테스트 환경을 생성할 수 있습니다.

분산 시스템 구성 요소

지금까지 AWS가 개별 마이크로서비스와 관련된 당면 과제를 어떻게 해결할 수 있는지 살펴보았습니다. 이제 서비스 검색, 데이터 일관성, 비동기 통신, 분산 모니터링, 감사와 같은 서비스 간 당면 과제에 집중해 보겠습니다.

주제

서비스 검색

마이크로서비스 아키텍처의 주요 당면 과제 중 하나는 서비스가 서로 검색하고 상호 작용할 수 있도록 하는 것입니다. 마이크로서비스 아키텍처의 분산 특성 때문에 서비스 간의 통신이 더 어려워질 뿐만 아니라, 이러한 시스템의 상태 확인하기 및 새로운 애플리케이션이 사용 가능한 상태가 될 때 알리기 등의 과제도 수반됩니다. 또한 애플리케이션에서 사용할 수 있는 구성 데이터와 같은 메타 스토어 정보를 저장하는 방법과 위치를 결정해야 합니다. 이 섹션에서는 AWS에서 마이크로서비스 기반 아키텍처의 서비스 검색을 수행하는 몇 가지 기법을 살펴봅니다.

DNS 기반 서비스 검색

이제 Amazon ECS에는 컨테이너화된 서비스가 서로 손쉽게 검색하고 연결할 수 있도록 지원하는 통합 서비스 검색 기능이 포함되어 있습니다.

이전에는 서비스가 서로 검색하고 연결할 수 있도록 보장하려면 Amazon Route 53, AWS Lambda 및 ECS 이벤트 스트림을 기반으로 자체 서비스 검색 시스템을 구성 및 실행하거나 모든 서비스를 로드 밸런서에 연결해야 했습니다.

Amazon ECS는 Route 53 Auto Naming API을 이용하여 서비스 이름 등록을 만들고 관리합니다. 이름이 DNS에 자동으로 매핑되기 때문에 코드의 이름으로 서비스를 참조하고 DNS 쿼리를 작성하여 실시간으로 서비스 엔드포인트에 이름을 삭제할 수 있다. 서비스 작업 정의에서 상태 확인 조건을 지정할 수 있으며 Amazon ECS는 정상적인 서비스 엔드포인트만이 서비스 조회 시 반환되게 합니다.

또한 Kubernetes에서 관리되는 서비스에 통합 서비스 검색 기능을 사용할 수 있습니다. 이러한 통합을 지원하기 위해 AWS는 Kubernetes 인큐베이터 프로젝트인 외부 DNS 프로젝트에 기여했습니다.

또 다른 옵션은 AWS Cloud Map의 기능을 활용하는 것입니다. AWS Cloud Map은 인터넷 프로토콜(IP), URL(Uniform Resource Locator), Amazon 리소스 이름(ARN) 등의 리소스를 위한 서비스 레지스트리를 제공하고 신속한 변경 사항 전파 기능과 속성을 사용하여 검색되는 리소스 세트의 범위를 좁히는 기능을 갖춘 API 기반 서비스 검색 메커니즘을 제공함으로써 자동 이름 지정 API의 기능을 확장합니다. 기존 Route 53 자동 이름 지정 리소스는 AWS Cloud Map으로 자동 업그레이드됩니다.

서드 파티 소프트웨어

서비스 검색을 구현하는 또 다른 접근 방식은 HashiCorp Consul, etcd 또는 Netflix Eureka와 같은 서드 파티 소프트웨어를 사용하는 것입니다. 세 가지 예 모두 신뢰할 수 있는 분산 키 값 스토어입니다. HashiCorp Consul의 경우 유연하고 확장 가능한 AWS 클라우드 환경을 설정하고, 선택한 구성으로 HashiCorp Consul을 자동으로 시작하는 AWS Quick Start가 있습니다.

서비스 메시

최신 마이크로서비스 아키텍처에서는 실제 애플리케이션이 수백 개, 심지어 수천 개의 서비스로 구성될 수 있습니다. 애플리케이션에서 가장 복잡한 부분은 실제 서비스 자체가 아니라 이러한 서비스 간의 통신인 경우가 많습니다. 서비스 메시는 서비스 간 통신을 처리하기 위한 추가 계층으로, 마이크로서비스 아키텍처에서 트래픽을 모니터링하고 제어하는 역할을 합니다. 이를 통해 서비스 검색과 같은 작업을 이 계층에서 전적으로 처리할 수 있습니다.

일반적으로 서비스 메시는 데이터 영역과 제어 영역으로 분할됩니다. 데이터 영역은 애플리케이션 코드와 함께 특수 사이드카 프록시로 분배되는 지능형 프록시 세트로 구성됩니다. 이 특수 사이드카 프록시는 마이크로서비스 사이의 모든 네트워크 통신에 개입합니다. 제어 영역은 프록시와의 통신을 담당합니다.

서비스 메시는 투명합니다. 이는 애플리케이션 개발자들은 이 추가적인 계층에 대해 알아야 할 필요가 없으며 기존 애플리케이션 코드를 변경할 필요가 없습니다. AWS App Mesh는 애플리케이션 수준의 네트워킹을 통해 서비스에서 여러 유형의 컴퓨팅 인프라와 서로 원활하게 통신할 수 있도록 지원하는 서비스 메시입니다. App Mesh는 서비스의 통신 방식을 표준화하여 완벽한 가시성을 제공하고 애플리케이션의 고가용성을 보장합니다.

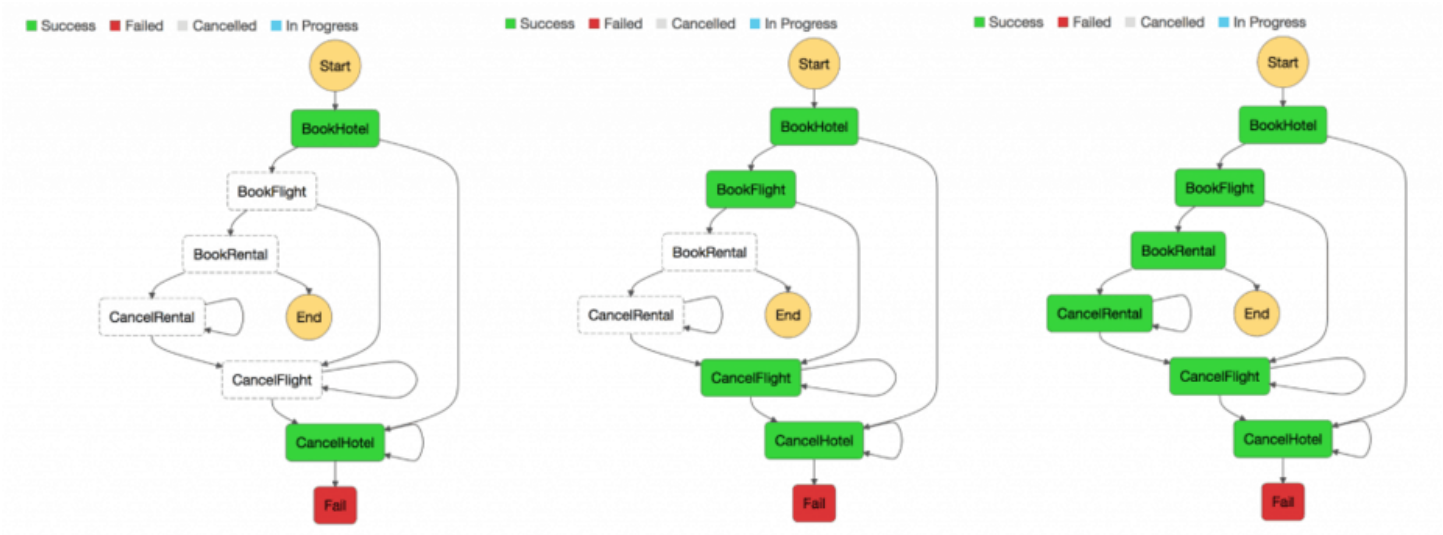
AWS Fargate, Amazon ECS, Amazon EKS, AWS의 자체 관리형 Kubernetes에서 실행 중인 기존 또는 새로운 마이크로서비스에 AWS App Mesh를 사용할 수 있습니다. App Mesh는 여러 클러스터, 오케스트레이션 시스템 또는 VPC에서 단일 애플리케이션으로 실행되는 마이크로서비스에 대한 통신을 코드 변경 없이 모니터링하고 제어할 수 있습니다.

분산 데이터 관리

일반적으로 대규모 관계형 데이터베이스에서 모놀리식 애플리케이션을 지원하여 모든 애플리케이션 구성 요소에 공통적으로 적용되는 단일 데이터 모델을 정의하게 됩니다. 마이크로서비스 접근 방식에서 이러한 중앙 데이터베이스는 분산되고 독립적인 구성 요소를 구축하는 목표에 방해가 됩니다. 각 마이크로서비스 구성 요소에는 고유한 데이터 지속성 계층이 있어야 합니다.

하지만 분산 데이터 관리로 인해 새로운 문제가 발생합니다. CAP 이론을 사용하는 데 따른 결과로서 분산 마이크로서비스 아키텍처는 기본적으로 일관성보다는 높은 성능을 보장하도록 설계되었기 때문에 최종 일관성을 수용해야 합니다.

분산 시스템에서는 비즈니스 트랜잭션이 여러 마이크로서비스에 걸쳐 구현될 수 있습니다. 이러한 마이크로서비스에서 단일 ACID 트랜잭션을 사용할 수 없으므로 부분적으로 실행할 수 있습니다. 이러한 경우 이미 처리된 트랜잭션을 다시 실행하기 위해 몇 가지 제어 로직이 필요합니다. 이를 위해 일반적으로 분산 Saga 패턴이 사용됩니다. 비즈니스 트랜잭션이 실패할 경우 Saga는 이전 트랜잭션에 의한 변경을 취소하는 일련의 보상 트랜잭션을 오케스트레이션합니다. AWS Step Functions를 사용하면 다음 그림과 같이 Saga 실행 조정자를 쉽게 구현할 수 있습니다.



Saga 실행 조정자

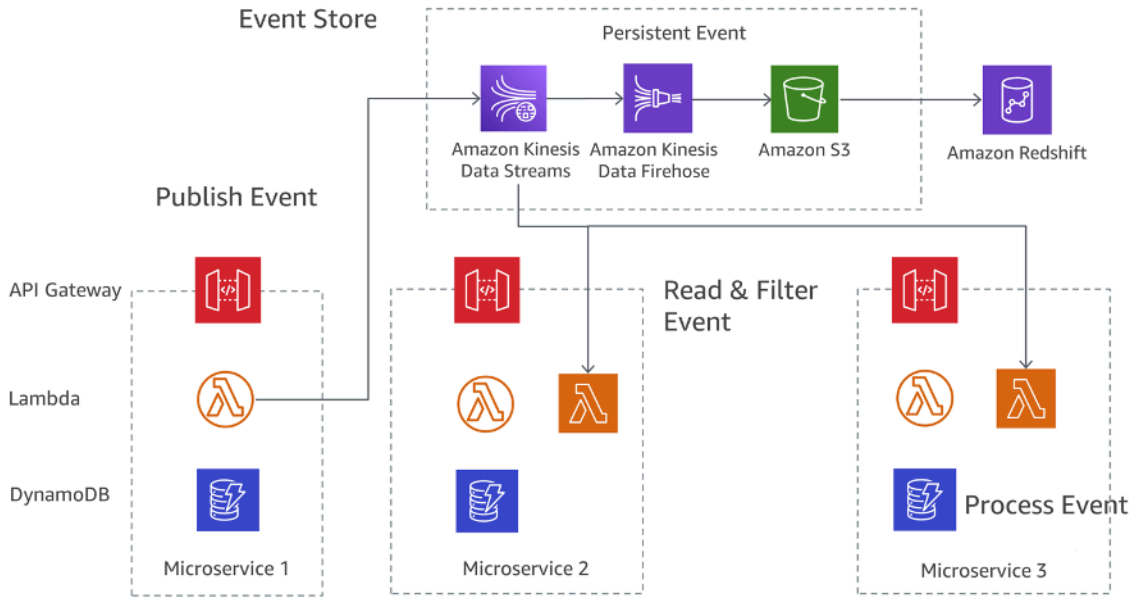
핵심 데이터 관리 도구 및 절차로 큐레이팅되는 중요 참조 데이터의 중앙 집중식 스토어를 구축하면 마이크로서비스에서 중요 데이터를 동기화하고 상태를 롤백할 수 있습니다. Lambda를 예약된 Amazon CloudWatch Events와 함께 사용하면 간단한 정리 및 중복 제거 메커니즘을 구축할 수 있습니다.

상태 변경이 둘 이상의 마이크로서비스에 영향을 미치는 것은 매우 일반적입니다. 이러한 경우 이벤트 소싱이 유용한 패턴인 것으로 입증되었습니다. 이벤트 소싱의 핵심 아이디어는 모든 애플리케이션 변

경을 이벤트 레코드로 표시하고 유지하는 것입니다 애플리케이션 상태를 유지하는 대신 데이터가 이벤트 스트림으로 저장됩니다. 데이터베이스 트랜잭션 로깅과 버전 제어 시스템은 이벤트 소싱의 두 가지 잘 알려진 예입니다. 이벤트 소싱에는 몇 가지 이점이 있습니다. 즉, 어느 시점에서든 상태를 확인하고 재구성할 수 있습니다. 이렇게 하면 자연스럽게 지속적인 감사 트레일이 생성되고 디버깅도 용이해집니다.

마이크로서비스 아키텍처의 맥락에서 이벤트 소싱은 게시/구독 패턴을 사용하여 애플리케이션의 여러 부분을 분리할 수 있으며, 동일한 이벤트 데이터를 각각 서로 다른 마이크로서비스를 지원하는 여러 데이터 모델에 공급합니다. 쓰기 워크로드에서 읽기 워크로드를 분리하고 성능, 확장성 및 보안에 맞게 이 두 워크로드를 모두 최적화하기 위해 이벤트 소싱과 CQRS(Command Query Responsibility Segregation) 패턴은 자주 함께 사용됩니다. 기존의 데이터 관리 시스템에서는 명령 및 쿼리가 동일한 데이터 리포지토리에 대해 실행됩니다.

다음 그림은 이벤트 소싱 패턴을 AWS에서 구현하는 방법을 보여줍니다. Amazon Kinesis Data Streams는 애플리케이션 변경 사항을 이벤트로 캡처하여 Amazon S3에 유지하는 중앙 이벤트 스토어의 주요 구성 요소 역할을 합니다. 다음 그림은 Amazon API Gateway, AWS Lambda 및 Amazon DynamoDB로 구성된 세 가지 서로 다른 마이크로서비스를 보여줍니다. 화살표는 이벤트의 흐름을 나타냅니다. 마이크로서비스 1에서 이벤트 상태가 변경되면 메시지를 Kinesis Data Streams에 기록하며 이벤트를 게시합니다. 모든 마이크로서비스는 메시지 복사본을 읽고 마이크로서비스의 관련성을 기준으로 이를 필터링한 다음 추가 처리를 위해 전달할 수 있는 자체 Kinesis Data Streams 애플리케이션을 AWS Lambda에서 실행합니다. 함수가 오류를 반환하면 Lambda는 처리가 성공할 때까지 또는 데이터가 만료될 때까지 배치를 재시도합니다. 샤드 중지를 방지하기 위해 보다 작은 배치 크기로 재시도하거나, 재시도 횟수를 제한하거나, 너무 오래된 레코드를 폐기하도록 이벤트 소스 매핑을 구성할 수 있습니다. 폐기된 이벤트를 유지하기 위해 실패한 배치에 대한 세부 정보를 Amazon Simple Queue Service(Amazon SQS) 대기열 또는 Amazon Simple Notification Service(Amazon SNS) 주제로 보내도록 이벤트 소스 매핑을 구성할 수 있습니다.



AWS의 이벤트 소싱 패턴

Amazon S3은 모든 마이크로서비스에 걸쳐 모든 이벤트를 안전하게 저장하며, 디버깅, 애플리케이션 상태 복구 또는 애플리케이션 변경 감사를 수행할 때 신뢰할 수 있는 단일 소스입니다. 레코드가 Kinesis Data Streams 애플리케이션에 두 번 이상 전달될 수도 있는 두 가지 주된 이유는 생산자 재시도 및 소비자 재시도입니다. 애플리케이션은 개별 레코드가 여러 번 처리될 것을 예상하고 이 문제를 적절하게 처리해야 합니다.

구성 관리

수십 개의 서로 다른 서비스가 포함된 일반적인 마이크로서비스 아키텍처에서 각 서비스는 서비스에 데이터를 노출하는 여러 다운스트림 서비스 및 인프라 구성 요소에 액세스해야 합니다. 메시지 대기열, 데이터베이스 및 기타 마이크로서비스를 예로 들 수 있습니다. 주요 과제 중 하나는 다운스트림 서비스 및 인프라로의 연결에 대한 정보를 제공하기 위해 각 서비스를 일관된 방식으로 구성하는 것입니다. 또한 구성에는 서비스가 작동 중인 환경에 대한 정보도 포함되어야 하며 새 구성 데이터를 사용하기 위해 애플리케이션을 다시 시작할 필요는 없습니다.

Twelve-Factor App 패턴의 세 번째 원칙에서는 'Twelve-Factor App은 환경 변수('환경'이라는 축약형을 사용하는 경우도 있음)에 구성을 저장합니다.'라는 주제를 다룹니다. Amazon ECS의 경우 Docker 실행을 위한 `--env` 옵션에 매핑되는 환경 컨테이너 정의 파라미터를 사용하여 환경 변수를 컨테이너에 전달할 수 있습니다. 환경 변수를 포함하는 하나 이상의 파일을 나열하는 `environmentFiles` 컨테이너 정의 파라미터를 사용하여 환경 변수를 대량으로 컨테이너에 전달할 수 있습니다. 파일은 Amazon

S3에서 호스팅되어야 합니다. AWS Lambda에서는 런타임을 통해 코드에서 환경 변수를 사용할 수 있으며 함수 및 호출 요청에 대한 정보가 포함된 추가 환경 변수를 설정할 수 있습니다. Amazon EKS의 경우 해당 포드의 구성 매니페스트에 대한 환경 필드에서 환경 변수를 정의할 수 있습니다. 환경 변수를 사용하는 다른 방법은 ConfigMap을 사용하는 것이다.

비동기식 통신 및 간단한 메시징

기존 모놀리식 애플리케이션에서의 통신 방식은 단순합니다. 즉, 애플리케이션의 일부가 메서드 호출 또는 내부 이벤트 배포 메커니즘을 사용하여 다른 부분과 통신합니다. 분리된 마이크로서비스를 사용하여 동일한 애플리케이션을 구현하는 경우 애플리케이션의 서로 다른 부분 간 통신은 네트워크 통신을 사용하여 구현해야 합니다.

REST 기반 통신

HTTP/S 프로토콜이 마이크로서비스 간에 동기식 통신을 구현하는 가장 보편적인 방법입니다. 대부분의 경우 RESTful API는 HTTP를 전송 계층으로 사용합니다. REST 아키텍처 스타일은 무상태 통신, 단일 인터페이스 및 표준 메서드를 사용합니다.

API Gateway에서는 프런트 도어의 역할을 하는 API를 생성할 수 있습니다. 애플리케이션은 이 도어를 통해 백엔드 서비스로부터 데이터, 비즈니스 로직 또는 기능에 액세스할 수 있습니다. API 개발자는 AWS나 다른 웹 서비스는 물론 AWS 클라우드에 저장된 데이터에 액세스하는 API를 생성할 수 있습니다. API Gateway 서비스를 사용해 정의한 API 객체는 리소스 및 메서드 그룹입니다.

리소스는 API 도메인 내에 유형이 지정된 객체이며, 데이터 모델과 연결되어 있거나 다른 리소스와 관계가 있을 수 있습니다. 하나 이상의 메서드, 즉 GET, POST 또는 PUT과 같은 표준 HTTP 동사에 응답하도록 각 리소스를 구성할 수 있습니다. REST API는 각기 다른 단계에 배포하고, 버전을 지정하고, 새로운 버전으로 복제할 수 있습니다.

API Gateway는 트래픽 관리, 권한 부여 및 접근 제어, 모니터링 및 API 버전 관리를 비롯하여 최대 수십만 건의 동시 API 호출을 수락하고 처리하는 데 관련된 모든 작업을 처리합니다.

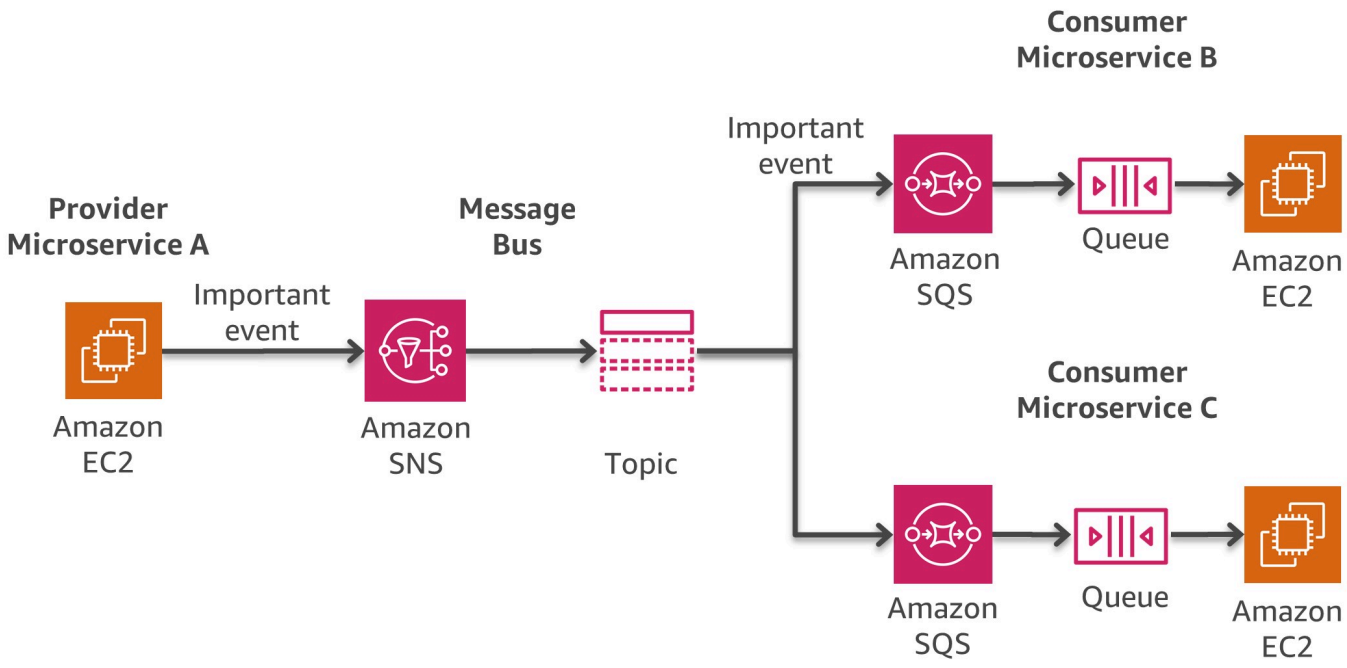
비동기 메시징 및 이벤트 전달

마이크로서비스 간의 통신을 구현하는 또 다른 패턴은 메시지 전달입니다. 서비스는 대기열을 통해 메시지를 교환하는 방식으로 통신합니다. 이러한 통신 방식의 주요 이점 중 하나는 서비스 검색 기능이 필요 없으며 서비스가 느슨하게 연결되어 있다는 점입니다.

동기식 시스템은 긴밀하게 연결되어 있어서 동기식 다운스트림 종속성의 문제가 업스트림 호출자에게 즉각적인 영향을 미칩니다. 업스트림 호출자에서 발생하는 재시도가 빠르게 팬아웃되고 문제를 증폭시킬 수 있습니다.

프로토콜과 같은 특정 요구 사항에 따라 AWS는 이러한 패턴을 구현하는 데 도움이 되는 다양한 서비스를 제공합니다. 가능한 구현 방법 중 하나는 Amazon Simple Queue Service(Amazon SQS) 대기열 또는 Amazon Simple Notification Service(Amazon SNS)의 조합을 사용하는 것입니다.

이 두 서비스는 긴밀하게 연동됩니다. Amazon SNS를 사용하면 애플리케이션에서 푸시 메커니즘을 통해 다수의 구독자에게 메시지를 전송할 수 있습니다. Amazon SNS와 Amazon SQS를 함께 사용하면 하나의 메시지를 다수의 소비자에게 전달할 수 있습니다. 다음 그림은 Amazon SNS와 Amazon SQS의 통합 방식을 보여줍니다.



AWS의 메시지 버스 패턴

Amazon SQS 대기열이 특정 SNS 주제를 구독하도록 설정하면 해당 주제에 메시지를 게시할 수 있고 Amazon SNS는 구독하는 Amazon SQS 대기열에 메시지를 전송합니다. 이 메시지에는 JSON 형식의 메타데이터 정보와 함께 주제에 게시된 제목 및 메시지가 들어 있습니다.

규모에 따라 내부 애플리케이션, 서드 파티 SaaS 애플리케이션 및 AWS 서비스를 포괄하는 이벤트 소스를 사용하여 이벤트 기반 아키텍처를 구축하는 또 다른 옵션은 Amazon EventBridge입니다. 완전관리형 이벤트 버스 서비스인 EventBridge는 서로 다른 소스로부터 이벤트를 수신하고, 라우팅 규칙을 기반으로 대상을 식별하며 AWS Lambda, Amazon SNS, Amazon Kinesis Streams를 포함한 대상에 거의 실시간으로 데이터를 전송합니다. 전송 전에 입력 변환기를 통해 인바운드 이벤트를 사용자 지정할 수도 있습니다.

이벤트 기반 애플리케이션을 훨씬 더 빠르게 개발하기 위해 EventBridge 스키마 레지스트리는 AWS 서비스에서 생성된 모든 이벤트에 대한 스키마를 포함하여 스키마를 수집하고 구성합니다. 또한 고객은 사용자 지정 스키마를 정의하거나 스키마 추론 옵션을 사용하여 스키마를 자동으로 검색할 수 있습니다. 그러나 이러한 모든 기능의 단점으로 EventBridge 전송에 대한 대기 시간 값이 상대적으로 높다는 것입니다. 또한 EventBridge의 기본 처리량 및 할당량이 사용 사례에 따라 지원 요청을 통해 증가할 수 있습니다.

다른 구현 전략이 Amazon MQ를 기반으로 구축될 수 있습니다. 이 서비스는 기존 소프트웨어가 JMS, NMS, AMQP, STOMP, MQTT, WebSocket 등 개방형 표준 API 및 프로토콜을 메시징에 사용하는 경우에 이용할 수 있습니다. Amazon SQS는 고객 API를 드러냅니다. 이는 온프레미스 환경 등에서 AWS로의 마이그레이션을 원하는 기존 애플리케이션이 있는 경우 코드를 변경해야 한다는 것을 의미합니다. Amazon MQ를 사용하면 많은 경우 코드를 변경할 필요가 없습니다.

Amazon MQ는 널리 사용되는 오픈 소스 메시지 브로커인 ActiveMQ의 유지 관리를 담당합니다. 기본 인프라는 애플리케이션의 안정성을 지원하기 위해 고가용성 및 메시지 내구성을 보장하도록 자동으로 프로비저닝됩니다.

오케스트레이션 및 상태 관리

마이크로서비스의 분산 특성으로 인해 여러 개의 마이크로서비스가 사용되는 워크플로를 오케스트레이션하기가 어렵습니다. 개발자는 서비스에 오케스트레이션 코드를 직접 추가하고 싶은 유혹을 느낄 수 있습니다. 하지만 그렇게 하면 결합이 더 단단해지고 개별 서비스를 신속하게 교체하기가 더 어려워지므로 이는 피해야 합니다.

AWS Step Functions를 사용하여 각각 개별 기능을 수행하는 개별 구성 요소로 애플리케이션을 구축할 수 있습니다. Step Functions는 오류 처리, 직렬화, 병렬화와 같은 서비스 오케스트레이션의 복잡성을 숨기는 상태 시스템을 제공합니다. 따라서 서비스 내에 추가적인 조정 코드를 사용하지 않고 애플리케이션을 신속하게 확장 및 변경할 수 있습니다.

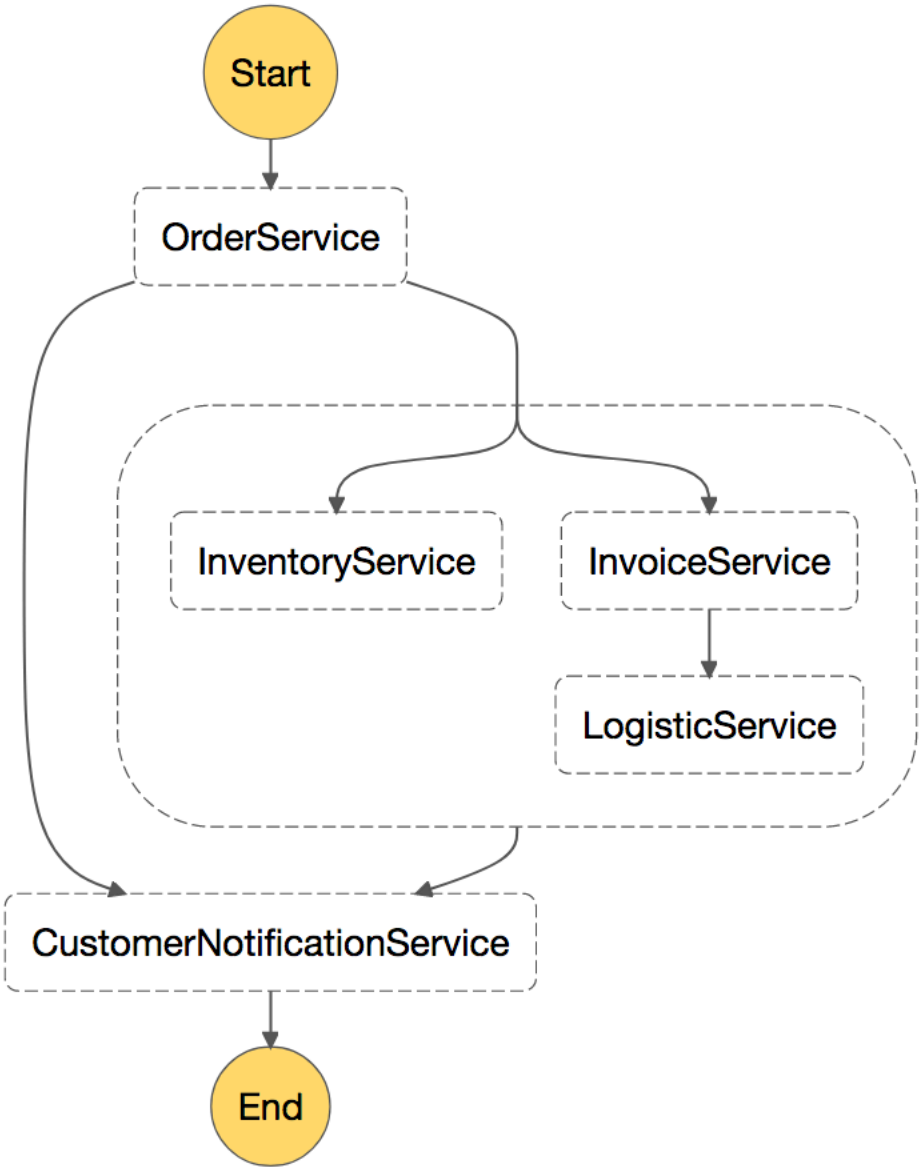
Step Functions는 구성 요소를 조정하고 애플리케이션의 기능을 단계별로 실행하는 신뢰할 수 있는 방법입니다. Step Functions에서는 애플리케이션의 구성 요소를 일련의 단계로 배열하고 시각화할 수 있는 그래픽 콘솔을 제공합니다. 그러므로 손쉽게 분산 서비스를 구축하고 실행할 수 있습니다.

Step Functions가 자동으로 각 단계를 시작 및 추적하고 오류가 발생할 경우 재시도하므로 애플리케이션이 의도대로 정상적으로 실행됩니다. Step Functions는 각 단계의 상태를 기록합니다. 따라서 무언가 잘못된 경우 빠르게 문제를 진단하고 디버깅할 수 있습니다. 코드를 작성하지 않고 단계를 변경 및 추가할 수 있어 애플리케이션을 개선하고 더 빠르게 혁신할 수 있습니다.

Step Functions는 AWS 서버리스 플랫폼의 일부이며 Lambda 함수의 오케스트레이션과 Amazon EC2, Amazon EKS, Amazon ECS와 같은 컴퓨팅 리소스에 기반한 애플리케이션과 Amazon SageMaker, AWS Glue와 같은 추가 서비스를 지원합니다. Step Functions는 어떤 규모에서든 애플리케이션을 사용할 수 있도록 운영 및 기본 인프라를 관리합니다.

Step Functions는 워크플로를 구축하기 위해 Amazon States Language를 사용합니다. 워크플로에는 순차적 또는 병렬 단계뿐만 아니라 분기 단계도 포함될 수 있습니다.

다음 그림은 순차적 단계와 병렬 단계를 결합한 마이크로서비스 아키텍처 워크플로의 예를 보여줍니다. 이러한 워크플로는 Step Functions API를 통해 또는 API Gateway를 사용하여 호출할 수 있습니다.



AWS Step Functions에서 호출한 마이크로서비스 워크플로의 예

분산 모니터링

마이크로서비스 아키텍처는 모니터링해야 하는 여러 분산된 구성 요소로 구성됩니다. Amazon CloudWatch를 사용하여 지표를 수집 및 추적하고, 로그 파일을 중앙 집중화 및 모니터링하며, 경보를 설정하고, AWS 환경의 변경 사항에 자동으로 대응할 수 있습니다. CloudWatch는 Amazon EC2 인스턴스, DynamoDB 테이블, Amazon RDS DB 인스턴스 같은 AWS 리소스뿐만 아니라 애플리케이션과 서비스에서 생성된 사용자 정의 지표 및 애플리케이션에서 생성된 모든 로그 파일을 모니터링할 수 있습니다.

모니터링

CloudWatch를 사용하여 시스템 전반의 리소스 사용률, 애플리케이션 성능, 운영 상태를 파악할 수 있습니다. CloudWatch는 몇 분 안에 시작할 수 있는 안정적이고 확장 가능하며 유연한 모니터링 솔루션을 제공합니다. 더 이상 자체 모니터링 시스템 및 인프라를 설정, 관리 및 확장할 필요가 없습니다. 마이크로서비스 아키텍처에서는 CloudWatch를 사용하여 사용자 지정 지표를 모니터링하는 기능이 추가적인 이점을 제공하는데, 이는 개발자가 각 서비스에 대해 수집해야 하는 지표를 결정할 수 있기 때문입니다. 또한 사용자 지정 지표를 기준으로 동적 확장을 구현할 수 있습니다.

Amazon CloudWatch 외에도 CloudWatch Container Insights를 사용하여 컨테이너화된 애플리케이션 및 마이크로서비스에서 지표 및 로그를 수집, 집계 및 요약할 수 있습니다. CloudWatch Container Insights는 CPU, 메모리, 디스크 및 네트워크와 같은 많은 리소스에 대한 지표를 자동으로 수집하고 클러스터, 노드, 포드, 작업 및 서비스 수준에서 CloudWatch 지표로 집계합니다. CloudWatch Container Insights를 사용하면 CloudWatch Container Insights 대시보드 지표에 액세스할 수 있습니다. 또한 컨테이너 재시작 오류 같은 진단 정보를 제공하여 문제를 격리하고 신속하게 해결할 수 있도록 도와줍니다. Container Insights에서 수집한 지표에 대해 CloudWatch 경보를 설정할 수도 있습니다.

Container Insights는 Amazon ECS, Amazon EKS 및 Amazon EC2 기반의 Kubernetes 플랫폼에 사용할 수 있습니다. Amazon ECS 지원에는 Fargate에 대한 지원이 포함됩니다.

특히, Amazon EKS에서 인기 있는 또 다른 옵션은 Prometheus를 사용하는 것입니다. Prometheus는 오픈 소스 모니터링 및 알림 도구 키트로, 수집된 지표를 시각화하는 데 Grafana와 함께 많이 사용됩니다. 대부분의 Kubernetes 구성 요소는 지표를 `/metrics`에 저장하며 Prometheus는 이러한 지표를 주기적으로 수집할 수 있습니다.

Amazon Managed Service for Prometheus(AMP)는 대규모의 컨테이너형 애플리케이션을 쉽게 모니터링할 수 있는 Prometheus 호환 모니터링 서비스입니다. AMP를 사용하면 운영 지표의 수집, 저장 및 쿼리 작업을 관리하는 데 필요한 기본 인프라를 관리해야 하는 부담 없이 오픈 소스 Prometheus 쿼리 언어(PromQL)를 사용하여 컨테이너형 워크로드의 성능을 모니터링할 수 있습니다. AWS Distro for

OpenTelemetry 또는 Prometheus 서버를 수집 에이전트로 사용하여 Amazon EKS 및 Amazon ECS 환경에서 Prometheus 지표를 수집할 수 있습니다.

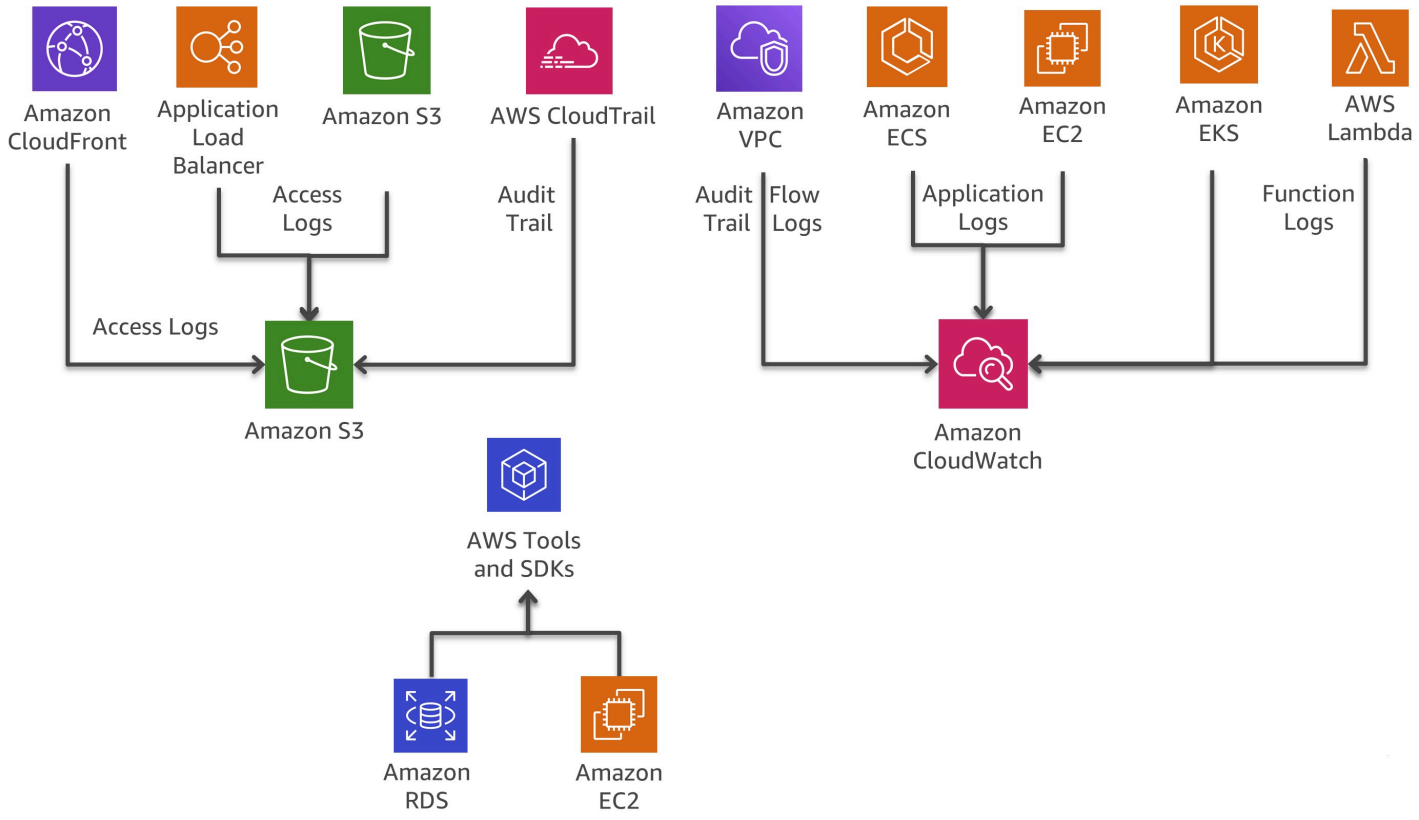
AMP는 종종 Amazon Managed Service for Grafana(AMG)와 함께 사용됩니다. AMG를 사용하면 저장 위치에 관계없이 지표를 쉽게 쿼리, 시각화, 알림 및 이해할 수 있습니다. AMG를 사용하면 서버 프로비저닝 작업, 소프트웨어 구성 및 업데이트 작업 또는 프로덕션 환경에서 Grafana의 보안 및 크기 조정과 관련한 번거로운 작업을 수행할 필요 없이 지표, 로그 및 추적을 분석할 수 있습니다.

로그 중앙 집중화

일관된 로깅은 문제를 파악하고 해결하는 데 중요합니다. 마이크로서비스를 통해 팀은 이전보다 많은 릴리스를 출시할 수 있으며, 엔지니어링 팀은 프로덕션에서 새로운 기능을 실험할 수 있습니다. 애플리케이션을 점진적으로 개선하기 위해서는 고객에게 미치는 영향을 파악하는 것이 중요합니다.

대부분의 AWS 서비스는 기본적으로 로그 파일을 중앙 집중화합니다. AWS의 기본 로그 파일 저장 위치는 Amazon S3 및 Amazon CloudWatch Logs입니다. Amazon EC2 인스턴스에서 실행되는 애플리케이션의 경우 데몬을 사용하여 로그 파일을 CloudWatch Logs로 전송할 수 있습니다. Lambda 함수는 기본적으로 로그 출력을 CloudWatch Logs로 전달하며, Amazon ECS는 컨테이너 로그를 CloudWatch Logs로 중앙 집중화할 수 있는 awslogs 로그 드라이버를 지원합니다. Amazon EKS의 경우 Fluent Bit 또는 Fluentd가 클러스터의 개별 인스턴스에서 중앙 집중식 로깅 CloudWatch Logs로 로그를 전달할 수 있습니다. 이러한 로그는 CloudWatch Logs에서 Amazon OpenSearch Service 및 Kibana를 통해 상위 수준 보고용으로 결합됩니다. 설치 공간이 작고 성능상의 이점이 있으므로 FluentD 대신 Fluent Bit를 사용하는 것이 좋습니다.

다음 그림은 일부 서비스의 로깅 기능을 보여줍니다. 그러면 팀은 Amazon OpenSearch Service 및 Kibana와 같은 도구를 사용하여 이러한 로그를 검색하고 분석할 수 있습니다. Amazon Athena는 Amazon S3의 중앙 집중식 로그 파일에 대해 임시 쿼리를 실행하는 데 사용할 수 있습니다.



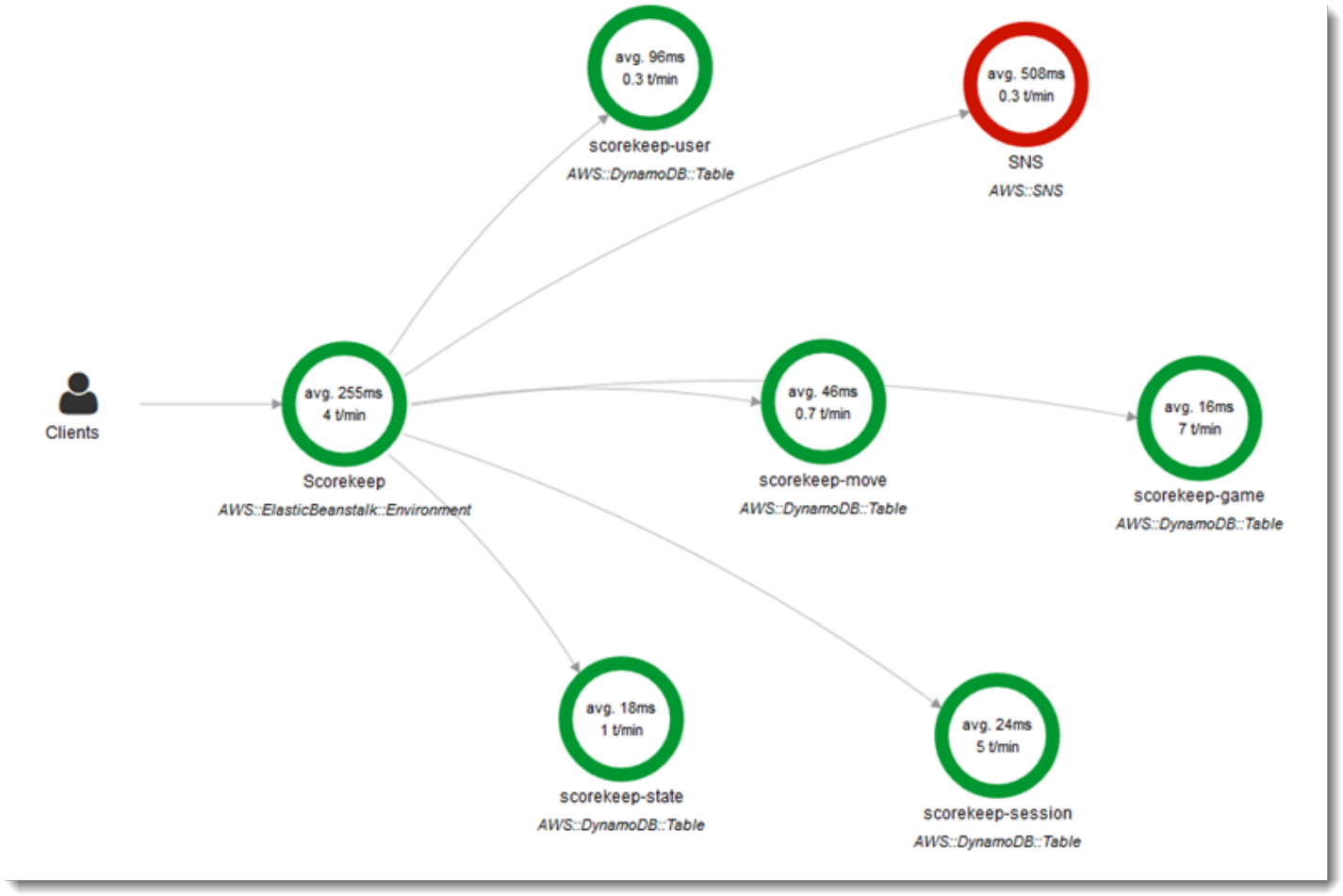
AWS 서비스의 로깅 기능

분산 추적

대부분의 경우, 일련의 마이크로서비스가 함께 연동하여 요청을 처리합니다. 수십 개의 마이크로서비스로 구성된 복잡한 시스템이 있고 호출 체인의 서비스 중 하나에서 오류가 발생하는 상황을 상상해보십시오. 모든 마이크로서비스가 제대로 로깅되고 있고 로그가 중앙 시스템에 통합되더라도, 모든 관련 로그 메시지를 찾는 것은 매우 어려울 수 있습니다.

AWS X-Ray의 핵심 아이디어는 상관 관계 ID를 사용하는 것입니다. 이 ID는 특정 이벤트 체인과 관련된 모든 요청과 메시지에 첨부되는 고유한 식별자입니다. 추적 ID는 요청이 첫 번째 X-Ray 통합 서비스(예: Application Load Balancer 또는 API Gateway)에 도달할 때 `X-Amzn-Trace-Id`라는 특정 추적 헤더의 HTTP 요청에 추가되며, 응답에도 포함됩니다. 모든 마이크로서비스가 X-Ray SDK를 통해 이 헤더를 읽을 수 있고 추가하거나 업데이트할 수도 있습니다.

X-Ray는 Amazon EC2, Amazon ECS, Lambda 및 AWS Elastic Beanstalk와 연동합니다. 이러한 서비스에 배포되었으며 Java, Node.js 및 .NET으로 작성된 애플리케이션에서 X-Ray를 사용할 수 있습니다.



AWS X-Ray 서비스 맵

Epsagon은 모든 AWS 서비스, 서드 파티 API(HTTP 호출 이용) 및 기타 일반 서비스(예: Redis, Kafka 및 Elastic)에 대한 추적을 포함하는 완전관리형 SaaS입니다. Epsagon 서비스에는 모니터링 기능, 가장 일반적인 서비스에 대한 경고 및 코드가 수행하는 모든 호출에 대한 페이로드 가시성이 포함됩니다.

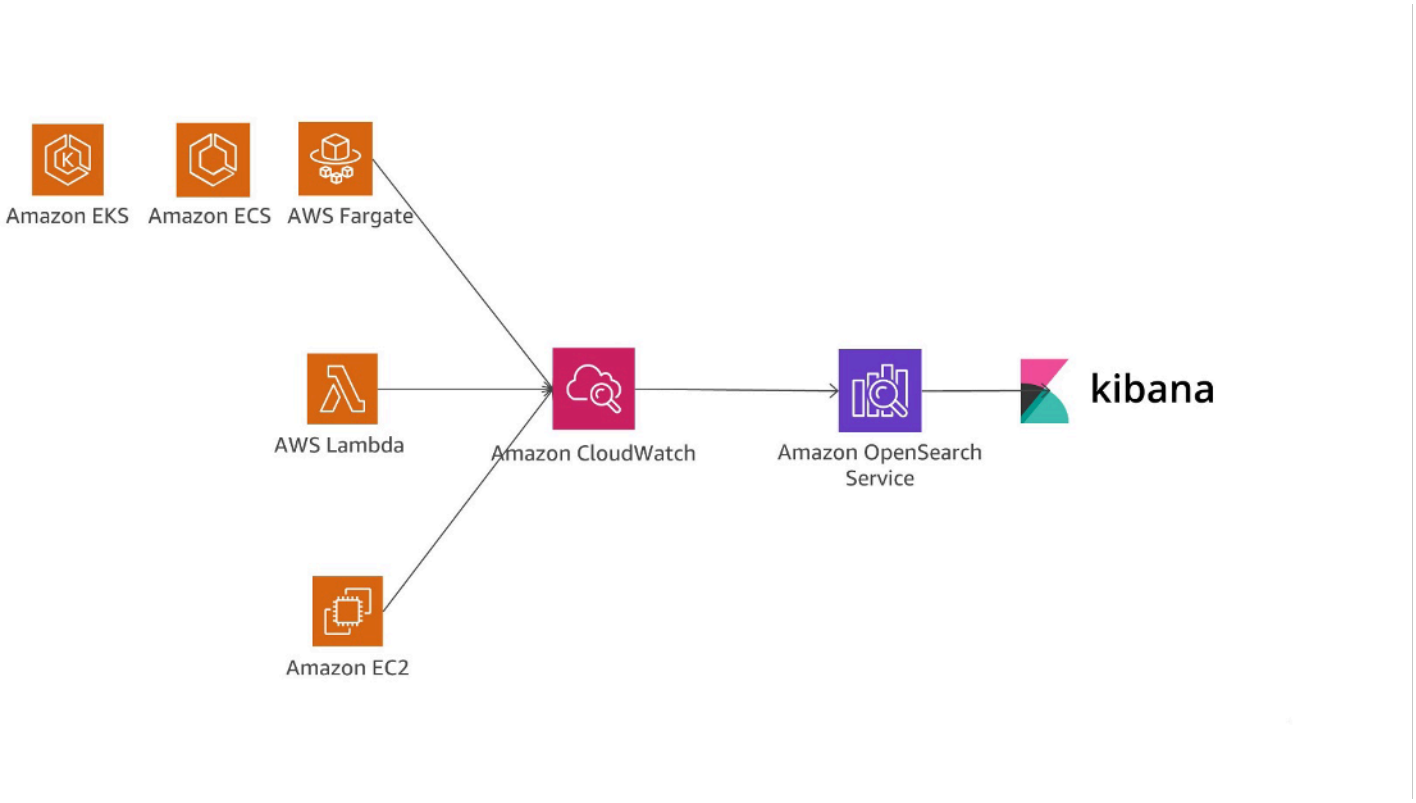
AWS Distro for OpenTelemetry는 OpenTelemetry 프로젝트의 안전한 프로덕션이 준비된 AWS가 지원하는 배포입니다. Cloud Native Computing Foundation의 일부인 AWS Distro for OpenTelemetry는 애플리케이션 모니터링을 위해 분산된 추적 및 지표를 수집하는 오픈 소스 API, 라이브러리 및 에이전트를 제공합니다. AWS Distro for OpenTelemetry를 사용하면 애플리케이션을 한 번만 계측하여 연관된 지표 및 추적을 여러 AWS 및 파트너 모니터링 솔루션으로 전송할 수 있습니다. 자동 계측 에이전트를 사용하면 코드 변경 없이 추적을 수집할 수 있습니다. 또한 AWS Distro for OpenTelemetry는 AWS 리소스 및 관리형 서비스에서 메타데이터를 수집하므로 애플리케이션 성능 데이터를 기반 인프라 데이터에 연관시켜 문제에 대한 평균 해결 시간을 줄일 수 있습니다. AWS Distro for OpenTelemetry를 사용하여 온프레미스뿐만 아니라 Amazon EC2, Amazon ECS, Amazon EC2 기반 Amazon EKS, Fargate 및 AWS Lambda에서 실행되는 애플리케이션을 계측할 수 있습니다.

AWS의 로그 분석 옵션

로그 데이터를 검색, 분석 및 시각화하는 것은 분산 시스템을 파악하는 데 있어 중요한 부분입니다. Amazon CloudWatch Logs Insights를 사용하면 로그를 즉각적으로 살펴보고 분석하고 시각화할 수 있습니다. 이를 통해 운영 문제를 해결할 수 있습니다. 로그 파일 분석을 위한 또 다른 옵션은 Amazon OpenSearch Service를 Kibana와 함께 사용하는 것입니다.

Amazon OpenSearch Service는 전체 텍스트 검색, 구조화 검색, 분석 및 이 세 가지 모두의 조합에 사용할 수 있습니다. Kibana는 Amazon OpenSearch Service와 원활하게 통합되는 오픈 소스 데이터 시각화 플러그 인입니다.

다음 그림은 Amazon OpenSearch Service 및 Kibana를 사용한 로그 분석을 보여줍니다. CloudWatch Logs 구독을 통해 로그 항목을 Amazon OpenSearch Service로 거의 실시간으로 스트리밍하도록 CloudWatch Logs를 구성할 수 있습니다. Kibana는 데이터를 시각화하고 Amazon OpenSearch Service의 데이터 스토어에 편리한 검색 인터페이스를 제공합니다. 이 솔루션은 ElastAlert와 같은 소프트웨어와 함께 사용하여 데이터에서 이상 징후, 사용량 급증 또는 기타 주의가 필요한 패턴이 감지될 경우 SNS 알림과 이메일을 전송하고 JIRA 티켓을 생성하는 등의 작업을 수행하는 알림 시스템을 구현할 수 있습니다.



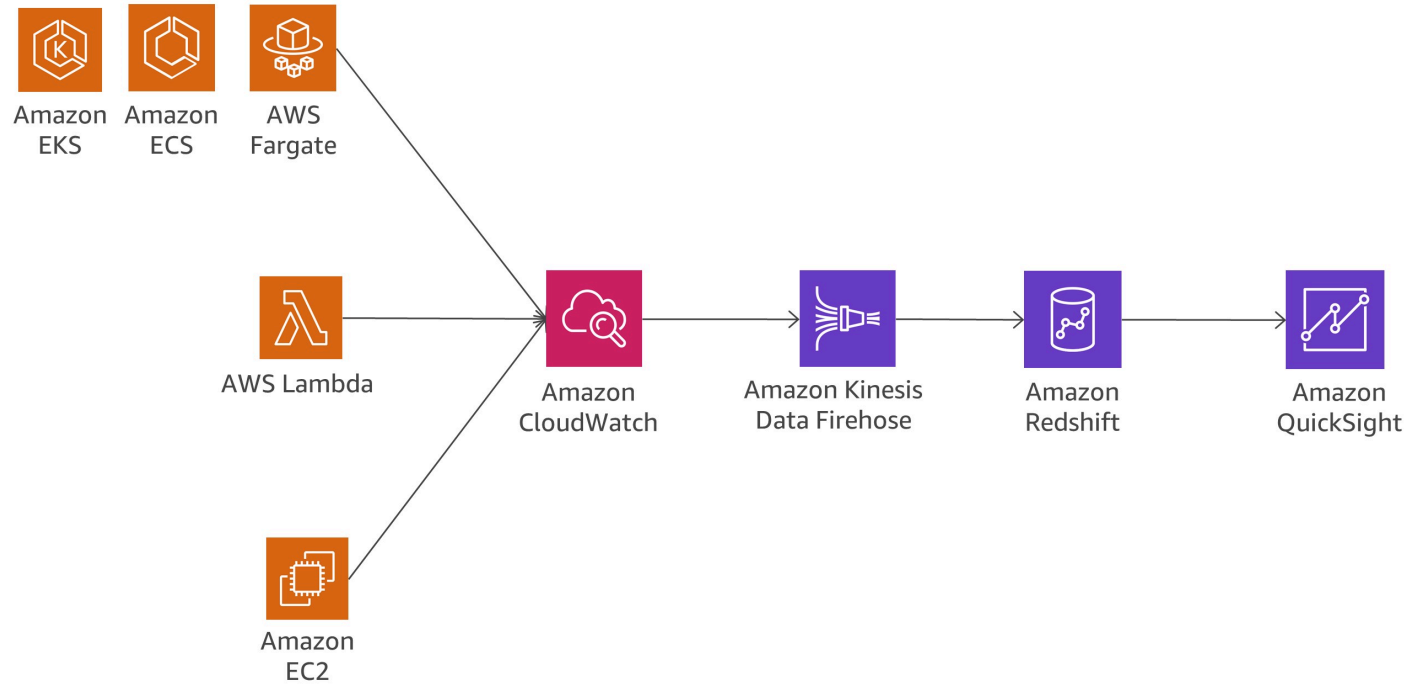
Amazon OpenSearch Service와 Kibana를 사용한 로그 분석

로그 파일 분석을 위한 또 다른 옵션은 Amazon Redshift를 Amazon QuickSight와 함께 사용하는 것입니다.

QuickSight는 Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon EMR, DynamoDB, Amazon S3, Amazon Kinesis 등의 AWS 데이터 서비스에 손쉽게 연결할 수 있습니다.

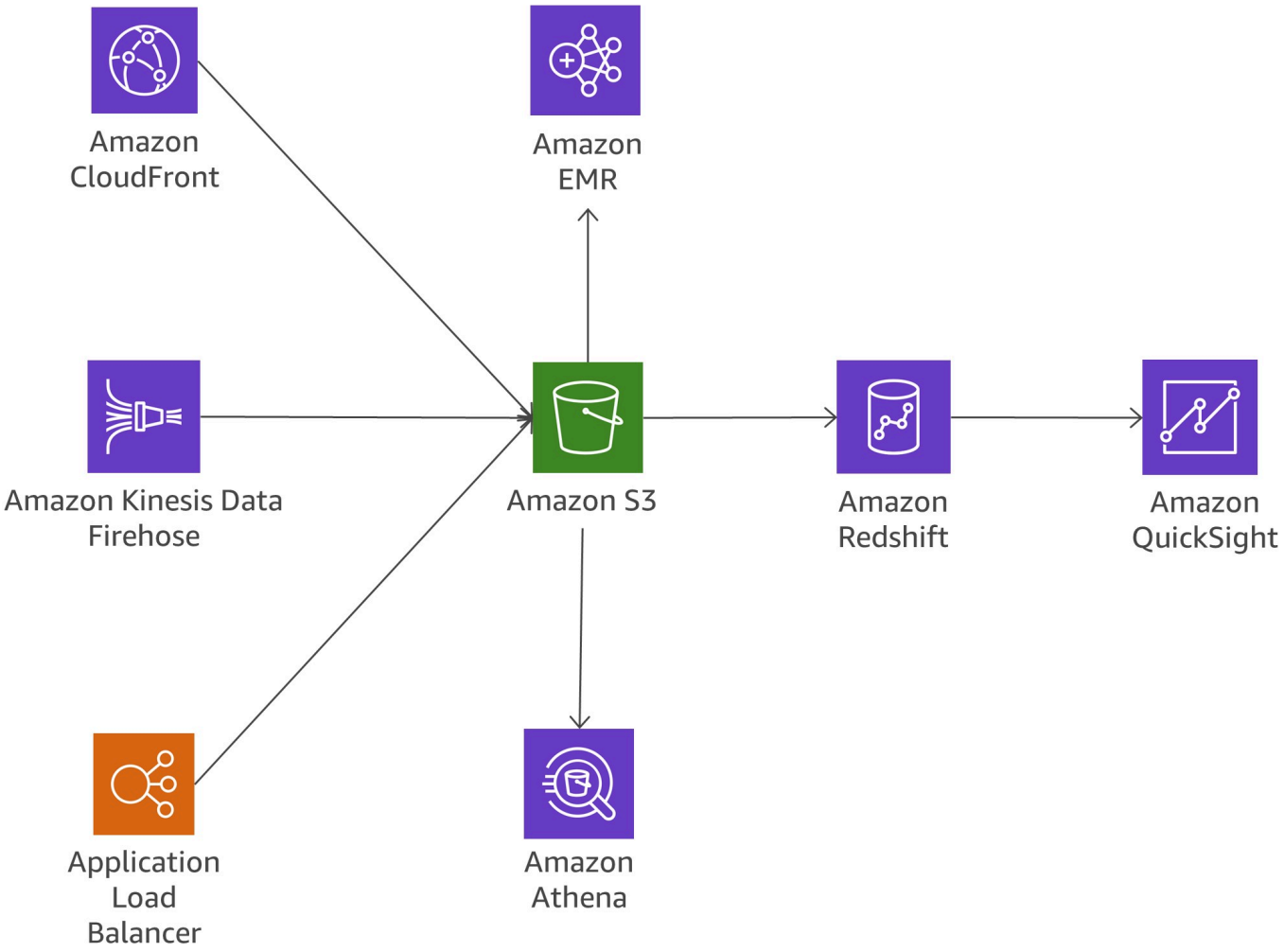
CloudWatch Logs는 로그 데이터의 중앙 스토어 역할을 할 수 있으며, 데이터를 저장하기만 하는 데 그치는 것이 아니라 로그 항목을 Amazon Kinesis Data Firehose로 스트리밍할 수도 있습니다.

다음 그림은 CloudWatch Logs 및 Kinesis Data Firehose를 사용하여 여러 소스에서 Amazon Redshift로 로그 항목을 스트리밍하는 시나리오를 보여줍니다. Amazon QuickSight는 분석, 보고 및 시각화를 위해 Amazon Redshift에 저장된 데이터를 사용합니다.



Amazon Redshift 및 Amazon QuickSight를 사용한 로그 분석

다음 그림은 Amazon S3에 대한 로그 분석 시나리오를 보여줍니다. 로그가 Amazon S3 버킷에 저장되는 경우 로그 데이터를 Amazon Redshift나 Amazon EMR 같은 다른 AWS 데이터 서비스에 로드하여 로그 스트림에 저장된 데이터를 분석하고 이상 징후를 찾을 수 있습니다.



Amazon S3의 로그 분석

통신 수

모놀리식 애플리케이션을 작은 마이크로서비스로 분할하면 마이크로서비스들이 서로 통신해야 하므로 통신 오버헤드가 증가합니다. 많은 구현 사례에서 HTTP를 통한 REST가 사용됩니다. HTTP를 통한 REST는 경량 통신 프로토콜이지만 메시지 볼륨이 클 경우 문제를 일으킬 수 있습니다. 경우에 따라 많은 메시지를 주고 받는 서비스를 통합하는 방법을 고려해야 할 수 있습니다. 통신 수를 줄이기 위해 점점 더 많은 서비스를 통합하는 상황에 직면한 경우 문제 도메인과 도메인 모델을 검토해야 합니다.

프로토콜

본 백서의 앞부분인 the section called “비동기식 통신 및 간단한 메시징” 섹션에서는 여러 가지 가능한 프로토콜에 대해 설명합니다. 마이크로서비스의 경우 HTTP와 같은 간단한 프로토콜을 사용하는 것이

일반적입니다. 서비스에서 교환하는 메시지는 JSON 또는 YAML과 같이 사람이 읽을 수 있는 형식이나 Avro 또는 Protocol Buffers와 같은 효율적인 바이너리 형식과 같이 다양한 방식으로 인코딩할 수 있습니다.

캐싱

캐시는 마이크로서비스 아키텍처의 지연 시간과 통신 수를 줄일 수 있는 좋은 방법입니다. 실제 사용 사례와 병목 현상에 따라 여러 캐싱 계층을 사용할 수 있습니다. AWS에서 실행되는 많은 마이크로서비스 애플리케이션은 ElastiCache를 사용하여 결과를 로컬로 캐싱함으로써 다른 마이크로서비스에 대한 호출량을 줄입니다. API Gateway는 백엔드 서버의 로드를 줄일 수 있도록 내장된 캐싱 계층을 제공합니다. 또한, 캐싱은 데이터 지속성 계층의 로드를 줄이는 데 유용합니다. 우수한 캐시 적중률과 데이터의 적시성/일관성 간에 적절한 균형을 찾는 것이 모든 캐싱 메커니즘의 공통적인 과제입니다.

감사

수백 개의 분산된 서비스로 구성될 수도 있는 마이크로서비스 아키텍처에서 해결해야 할 또 다른 과제는 각 서비스에 대한 사용자 작업의 가시성을 보장하고 조직 차원에서 모든 서비스를 전반적으로 잘 파악할 수 있도록 하는 것입니다. 보안 정책을 시행하려면 리소스 액세스뿐만 아니라 시스템 변경으로 이어지는 활동까지 모두 감사하는 것이 중요합니다.

개별 서비스 수준과 서비스 전반에 걸친 더 넓은 시스템에서 변경 사항을 추적해야 합니다. 일반적으로 변경은 마이크로서비스 아키텍처에서 자주 발생합니다. 이로 인해 변경 감사가 더 중요해집니다. 이 섹션에서는 마이크로서비스 아키텍처를 감사하는데 유용한 AWS의 주요 서비스와 기능을 살펴봅니다.

감사 추적

AWS CloudTrail은 AWS 클라우드에서 실행된 모든 API 호출을 기록하여 실시간으로 CloudWatch Logs에 전달하거나 몇 분 안에 Amazon S3에 전달할 수 있으므로 마이크로서비스의 변경 사항을 추적하는 데 유용한 도구입니다.

모든 사용자 및 자동화된 시스템 작업은 검색할 수 있으며, 예기치 않은 동작, 회사 정책 위반 또는 디버깅을 목적으로 분석할 수 있습니다. 기록되는 정보로는 타임스탬프, 사용자/계정 정보, 호출된 서비스, 요청된 서비스 작업, 호출자의 IP 주소, 요청 파라미터 및 응답 요소 등이 있습니다.

CloudTrail을 사용하면 동일한 계정에 대해 여러 추적을 정의할 수 있으므로 보안 관리자, 소프트웨어 개발자, IT 감사자 등 다양한 이해 관계자가 자체적으로 추적을 생성하고 관리할 수 있습니다. 마이크로서비스 팀에서 여러 AWS 계정을 사용하는 경우 여러 추적을 단일 S3 버킷에 집계할 수 있습니다.

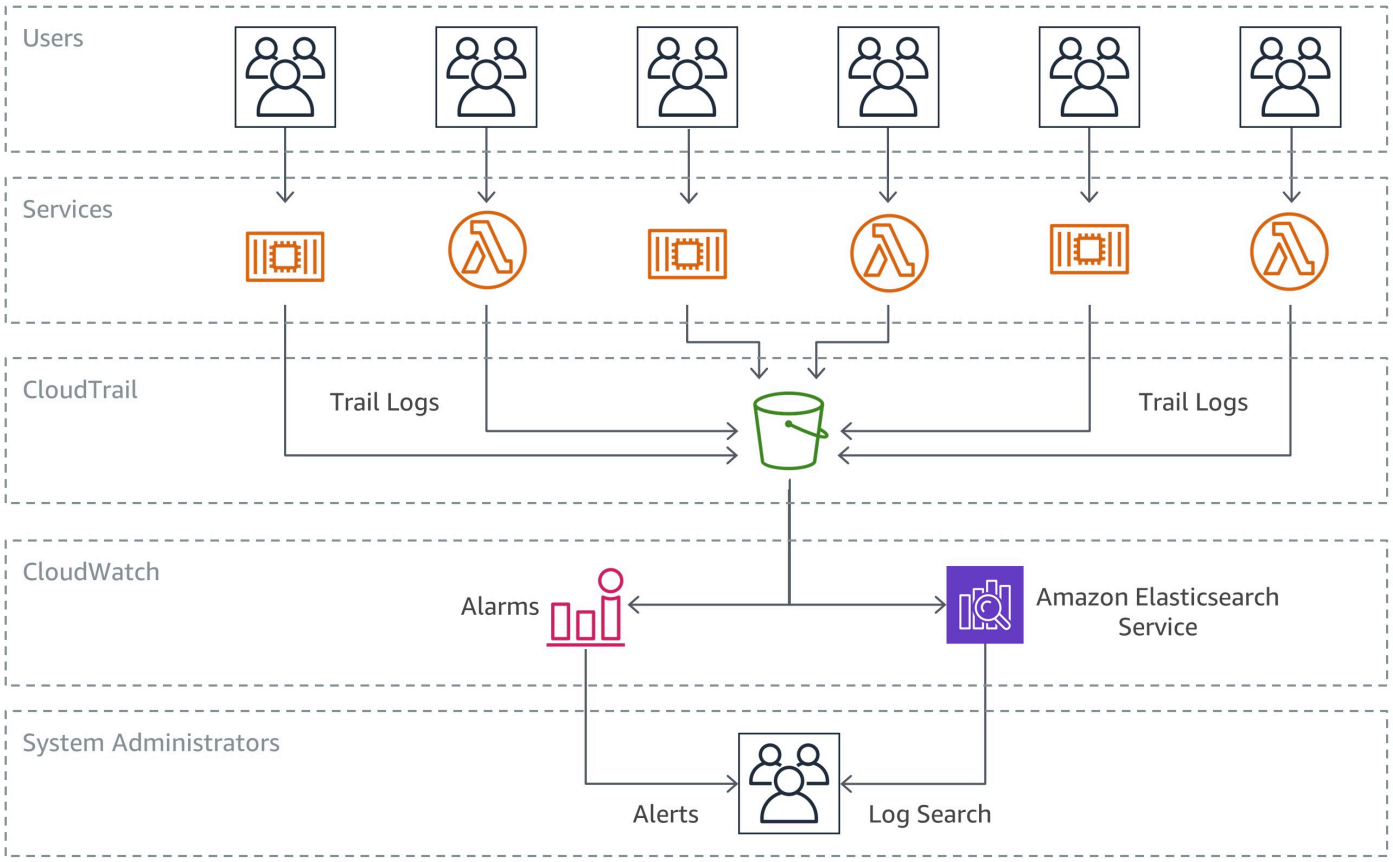
감사 추적을 CloudWatch에 저장하면 감사 추적 데이터가 실시간으로 기록되고 검색 및 시각화를 위해 정보를 Amazon OpenSearch Service로 쉽게 다시 라우팅할 수 있다는 이점이 있습니다. Amazon S3과 CloudWatch Logs 모두에 로그인하도록 CloudTrail을 구성할 수 있습니다.

이벤트 및 실시간 작업

시스템 아키텍처의 특정 변경 사항에는 신속하게 대응해야 하며, 이러한 상황을 해결하기 위한 조치를 취하거나 변경 사항을 승인하기 위한 특정 관리 절차를 시작해야 합니다. Amazon CloudWatch Events와 CloudTrail을 통합하면 모든 AWS 서비스에서 모든 변종 API 호출에 대한 이벤트를 생성할 수 있습니다. 또한 사용자 지정 이벤트를 정의하거나 정해진 일정에 따라 이벤트를 생성할 수도 있습니다.

이벤트가 발생하고 정의된 규칙과 일치하면 조직의 사전 정의된 사용자 그룹에 즉시 알림을 보내 담당자가 적절한 조치를 취할 수 있게 합니다. 필요한 조치를 자동화할 수 있는 경우 규칙을 통해 자동으로 기본 제공된 워크플로를 트리거하거나 Lambda 함수를 호출하여 문제를 해결할 수 있습니다.

다음 그림은 마이크로서비스 아키텍처 내에서 CloudTrail과 CloudWatch Events가 함께 작동하여 감사 및 문제 해결 요구 사항을 해결하는 환경을 보여줍니다. CloudTrail은 모든 마이크로서비스를 추적하며, 감사 추적은 Amazon S3 버킷에 저장됩니다. CloudWatch Events는 운영 변경 사항이 발생할 때 이를 인식합니다. CloudWatch Events는 이러한 운영 변경 사항에 응답하고, 환경에 응답하기 위한 메시지를 전송하고 함수를 활성화하며 변경을 수행하고 상태 정보를 기록하는 등 필요에 따라 수정 조치를 취합니다. CloudWatch Events는 CloudTrail보다 상위에 위치하며 아키텍처가 변경될 때 알림을 트리거합니다.



감사 및 수정

리소스 인벤토리 및 변경 관리

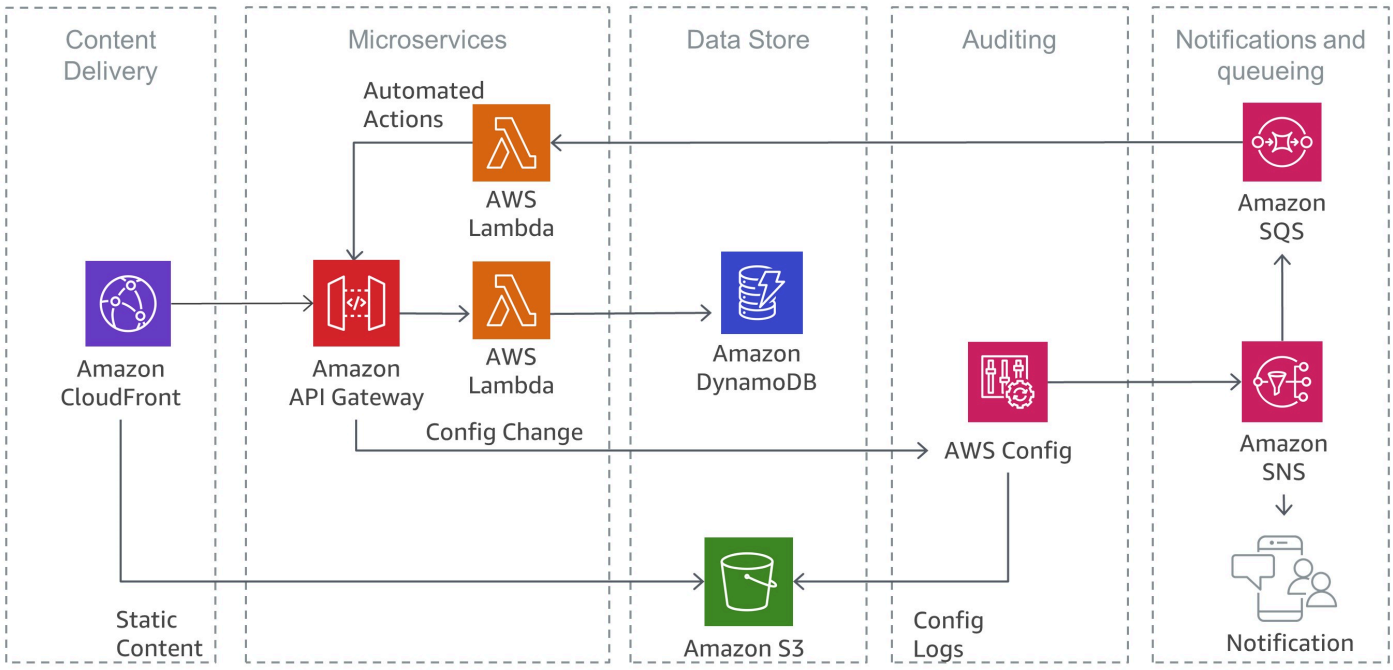
애자일 개발 환경에서 빠르게 변경되는 인프라 구성을 지속적으로 제어하려면 아키텍처를 감사하고 제어할 수 있는 좀 더 자동화된 관리형 접근 방식이 중요합니다.

CloudTrail과 CloudWatch Events가 여러 마이크로서비스에 걸쳐 인프라 변경 사항을 추적하고 그에 대응하는 데 있어서 중요한 빌딩 블록이라면 AWS Config 규칙은 기업이 특정 규칙으로 보안 정책을 정의하여 정책 위반의 감지, 추적, 알림을 자동화할 수 있도록 하는 서비스입니다.

다음 예는 마이크로서비스 아키텍처 내에서 규정을 준수하지 않는 구성 변경 사항을 탐지하고 알려주고 자동으로 대응하는 방법을 보여줍니다. 개발 팀원이 마이크로서비스가 HTTPS 요청만 허용하는 대신에 엔드포인트가 인바운드 HTTP 트래픽을 허용하도록 API Gateway를 변경했습니다.

이러한 상황은 조직에 의해 이전에는 보안 규정 준수 문제로 식별되었기 때문에 AWS Config 규칙은 이 조건을 이미 모니터링하고 있습니다. 이 규칙은 해당 변경 사항을 보안 위반으로 식별하고 두 가지 작업을 수행합니다. 즉, 감사를 위해 Amazon S3 버킷에 감지된 변경 사항의 로그를 생성하고, SNS 알

림을 생성합니다. 이 시나리오에서 Amazon SNS는 두 가지 목적으로 사용됩니다. 첫째는 지정된 그룹에 이메일을 보내 보안 위반을 알리는 것이고, 둘째는 SQS 대기열에 메시지를 추가하는 것입니다. 다음으로, 메시지를 선택하고 API Gateway 구성을 변경하여 규정 준수 상태를 복원합니다.



AWS Config를 사용한 보안 위반 감지

결론

마이크로서비스 아키텍처는 기존 모놀리식 아키텍처의 한계를 극복하도록 개발된 분산 설계 방식입니다. 마이크로서비스는 애플리케이션과 조직을 확장하는 한편 주기를 단축하는 데 도움이 됩니다. 그러나 마이크로서비스는 아키텍처 복잡성과 운영 부담을 가중시킬 수 있는 몇몇 문제도 안고 있습니다.

AWS는 제품 팀이 마이크로서비스 아키텍처를 구축하고 아키텍처 및 운영상의 복잡성을 최소화할 수 있도록 지원하는 대규모 관리형 서비스 포트폴리오를 제공합니다. 본 백서에서는 관련 AWS 서비스를 하나씩 소개하고, AWS 서비스를 기본적으로 활용하여 서비스 검색 또는 이벤트 소싱과 같은 일반적인 패턴을 구현하는 방법을 설명합니다.

리소스

- AWS 아키텍처 센터
- AWS 백서
- AWS Architecture Monthly
- AWS 아키텍처 블로그
- This Is My Architecture 동영상
- AWS Answers
- AWS 설명서

문서 이력 및 기여자

문서 이력

본 백서의 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하세요.

update-history-change	update-history-description	update-history-date
백서 업데이트됨	Amazon EventBridge, AWS OpenTelemetry, AMP, AMG, Container Insights를 통합했고, 텍스트를 약간 변경했습니다.	2021년 11월 9일
마이너 업데이트	페이지 레이아웃을 조정했습니다.	2021년 4월 30일
마이너 업데이트	텍스트를 약간 변경했습니다.	2019년 8월 1일
백서 업데이트됨	Amazon EKS, AWS Fargate, Amazon MQ, AWS PrivateLink, AWS App Mesh, AWS Cloud Map을 통합했습니다.	2019년 6월 1일
백서 업데이트됨	AWS Step Functions, AWS X-Ray 및 ECS 이벤트 스트림을 통합했습니다.	2017년 9월 1일
최초 게시	AWS에서 마이크로서비스 구현을 게시했습니다.	2016년 12월 1일

> **Note**
>
> RSS 업데이트에 가입하려면 사용 중인 브라우저에 대해 RSS 플러그인이 활성화되어 있어야 합니다.

기여자

다음은 본 문서를 작성하는 데 도움을 준 개인 및 조직입니다.

- Sascha Möllering, AWS, 솔루션스 아키텍트
- Christian Müller, AWS, 솔루션스 아키텍트
- Matthias Jung, AWS, 솔루션스 아키텍트
- Peter Dalbhanjan, AWS, 솔루션스 아키텍트
- Peter Chapman, AWS, 솔루션스 아키텍트
- Christoph Kassen, AWS, 솔루션스 아키텍트
- Umair Ishaq, AWS, 솔루션스 아키텍트
- Rajiv Kumar, AWS, 솔루션스 아키텍트

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.