AWS 기반 현대적 애플리케이션 개발

AWS 기반 클라우드 네이티브 모던 애플리케이션 개발 및 설계 패턴

2019년 10월

고지 사항

고객은 본 문서에 포함된 정보를 독립적으로 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.

© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved.

목차

소개 5

- 혁신 플라이휠 가속화 5

모던 애플리케이션 개발 6

- 모던 애플리케이션의 기능 6
- 모던 애플리케이션 개발을 위한 모범 사례 8

모던 애플리케이션 설계 패턴 14

- AWS 서비스를 사용하여 마이크로서비스 아키텍처 구현 14

AWS에서의 지속적 통합 및 지속적 전달 31

- AWS 기반 CI/CD 서비스 31
- 다양한 애플리케이션 유형별 CI/CD 패턴 34

결론 40

기고자 40

추가 자료 41

- AWS 서비스 41
- 백서 42
- 동영상 43

문서 개정 43

참고 44

개요

컨테이너 및 서버리스 기술을 사용한 모던 애플리케이션 개발은 조직이 혁신을 가속화하는 데 도움이 될 수 있습니다. 이 백서에는 AWS 클라우드에 모던 애플리케이션을 구축하는 데 활용할 수 있는 중요한 모범 사례와 설계 패턴에 대한 정보가 수록되어 있습니다.

소개

현대 기업들은 갈수록 글로벌화되고, 그 제품은 갈수록 디지털화되고 있습니다. 클라우드 인프라, 모바일 앱, 빅 데이터 파이프라인, 소셜 미디어 등의 디지털 제품은 애플리케이션 개발에 영향을 미치며, 기업에 전례 없는 변화 속도를 요구하고 있습니다. 비즈니스 리더들은 이러한 속도를 실현하기 위해 디지털 시대의 새로운 환경에 맞춰 문화, 프로세스, 기술을 재정립해야 합니다.

인적 자원을 최대한 활용하고 새로운 기회를 찾고 새로운 아이디어를 육성하여 성장을 도모해야 하는 현대 기업에게는 신속한 혁신이 필수적입니다. 디지털 기술은 이러한 빠른 혁신의 핵심입니다.

혁신 플라이휠 가속화

거의 모든 산업 분야에서 기업들은 전례 없는 변화 속도를 경험하고 있으며, 빠른 혁신은 이 같은 변화 속도를 쫓아가는 능력을 개선하는 데 매우 중요합니다. 작고 알려지지 않은 경쟁업체들이 혁신에 집중하여 몇 달 만에 앞서 나갈 수 있으므로 단순히 혁신하는 것이 아니라 신속하게 혁신하는 것이 필수적입니다.

Amazon은 실험을 통해 더 빠르게 혁신할 수 있음을 경험으로 배웠습니다. 혁신을 가속화하기 위해 실험을 하고, 사용자 피드백을 듣고, 다시 실험합니다. 실패를 두려워하지 않고 향후 혁신을 위한 작업에 각 실험에서 얻은 교훈을 적용합니다. 이를 혁신 플라이휠이라고 합니다. 이 플라이휠을 빠르게 진행하려면 제품을 출시하고, 피드백을 수집하고, 새로운 기능을 추가하고, 다시 릴리스하는 시스템이 필요합니다. 모던 애플리케이션의 기능은 이러한 프로세스를 가능하게 하며, 빠른 혁신을 통해 플라이휠을 진행하고 경쟁에서 앞서 나갈 수 있게 해줍니다.

모던 애플리케이션 개발

성공을 거두고 있는 기업들은 경쟁사와 차별화할 수 있는 핵심 요소가 자사의 기술이라는 사실을 잘 알고 있습니다. 기업이 성장하고 성공을 실현하기 위해서는 신제품을 신속하게 개발해야 합니다. 이를 가능하게 하는 혁신 문화를 증진하기 위해 성공적인 기업들은 애플리케이션 설계, 구축 및 관리 방법을 지속적으로 업데이트합니다. 이를 모던 애플리케이션 개발이라고 합니다.

모던 애플리케이션 개발은 기업이 빠르게 혁신할 수 있도록 지원하여 경쟁 우위를 확보할 수 있게 해줍니다. 혁신 기술을 포용하는 기업은 인프라 관리 및 프로비저닝과 같은 획일적이고 과중한 업무를 수행하던 리소스를 더 가치 있는 활동에 투입함으로써 더 많은 실험을 수행하고 새로운 아이디어를 시장에 더 빠르게 선보일 수 있습니다.

모던 애플리케이션 개발 방식은 기업이 혁신 기술에 신속하게 대응하고 민첩성을 실현하는 데 도움이 될 수 있습니다. 일부 고객은 온프레미스 가상 머신(Virtual Machine, VM)을 Amazon Elastic Compute Cloud(EC2[1])로 이동(리프트 앤 시프트라고도 함)하여 호스팅합니다. 애플리케이션 플랫폼을 클라우드에 최적화된 컨테이너 기반 모델로 변경하는 고객도 있습니다. 그런가 하면 모놀리식 애플리케이션을 리팩터링하고 마이크로서비스 기반 아키텍처로 전환하는 기업도 있습니다. 대부분의 기업은 클라우드 네이티브 애플리케이션을 구축할 때 관리 오버헤드로 인한 업무 시간이 줄어들고 핵심 비즈니스에 더 집중할 수 있게 됩니다.

모던 애플리케이션의 기능

모던 애플리케이션은 다음과 같은 기능을 제공해야 합니다.

- **보안** – 모든 애플리케이션에는 보안이 무엇보다 중요합니다. 보안 조치는 애플리케이션의 특정 부분에만 적용하는 것이 아니라 모든 계층과 수명 주기의 모든 단계에서 구현되어야 합니다.

- **복원력 –** 모던 애플리케이션은 복원력이 뛰어납니다. 예를 들어 애플리케이션에서 외부 데이터 원본을 호출할 때 오류가 발생할 경우, 응답하지 않게 되는 것이 아니라 재시도하거나 예외를 처리해야 하며 기능이 정상적으로 성능 저하된 상태로 계속 작동해야 합니다. 이 패턴은 마이크로서비스 아키텍처[2] 및 다른 서비스와의 상호 작용에도 적용됩니다.
- **탄력성 –** 모던 애플리케이션은 요청 속도 또는 기타 지표에 따라 유연하게 확장하고 축소함으로써 비즈니스 기회를 놓치지 않고 비용을 최적화할 수 있습니다. 확장 및 축소 프로세스를 자동화하거나 Auto Scaling 기능을 비롯한 관리형 서비스를 사용하면 일상적인 관리 작업을 줄이고 중대한 가동 중단 문제를 방지할 수 있습니다.
- **모듈식 –** 모던 애플리케이션은 응집도가 높고 느슨하게 결합된 모듈로 구성됩니다. 대규모 시스템은 단일 모놀리식으로 구축되어서는 안 되며, 도메인 경계를 따라 서로 다른 역할을 하는 여러 구성 요소로 분리되어야 합니다. 이렇게 분리할 경우 가용성과 확장성이 향상될 뿐만 아니라 서로 다른 구성 요소를 독립적으로 배포할 수 있으므로 릴리스가 더 쉬워집니다.
- **자동화 –** 잦은 고품질 릴리스를 지원하려면 모던 애플리케이션의 통합 및 배포를 자동화해야 합니다. 수동 프로세스는 오류가 발생하기 쉬울 뿐만 아니라, 단일 관리자가 모든 배포 작업을 처리하게 되는 등 개별 사용자에 의존하게 될 수 있습니다. 민첩한 개발과 잦은 릴리스를 지원하려면 지속적 통합 및 지속적 전달(CI/CD) 파이프라인을 통해 모던 애플리케이션을 배포해야 합니다. CI/CD 모델에서는 코드를 버전 관리 시스템으로 푸시하고, 안전한 CI 환경에서 테스트를 실행하며, 모든 테스트를 통과하면 자동으로 배포를 실행합니다.
- **상호 운용성 –** 모던 애플리케이션에서 각 서비스는 다른 서비스와 상호 작용하고, 요청된 리소스를 제공하고, 예상되는 작업을 수행해야 합니다. 다양한 서비스에 독립적으로 기능을 추가하고 다른 서비스에 영향을 주지 않으면서 자주 릴리스할 수 있어야 합니다. 즉, 서비스의 구현 세부 정보를 비공개로 유지하면서도, 강력한 공개 API를 통해 필요한 모든 기능을 노출해야 합니다. 또한 독립적인 릴리스를 허용하려면 이러한 공개 API가 안정적이고 이전 버전과 호환되어야 합니다.

모던 애플리케이션은 다양한 방식으로 구현할 수 있습니다. 이 백서에는 컨테이너와 서버리스 기술을 활용하여 클라우드에서 애플리케이션을 배포하는 방법에 대한 정보가 수록되어 있습니다.

모던 애플리케이션 개발을 위한 모범 사례

고객, 그리고 사내 개발 팀과의 대화를 통해 AWS는 혁신적인 아이디어를 시장에 빠르게 선보이고 있는 조직에서 공통으로 나타나는 몇 가지 최신 애플리케이션 개발 모범 사례를 찾아냈습니다.

보안 및 규정 준수

AWS 클라우드에 시스템을 구축할 때는 항상 보안 및 규정 준수부터 시작하는 것이 좋습니다. 전체 애플리케이션 수명 주기에 보안을 적용하면 조직은 혁신 속도를 그대로 유지하면서 보안 위협을 해결할 수 있습니다.
예:

- **인증 –** 악의적인 액세스를 방지하는 권한 설정으로 시스템에 대한 액세스를 제어합니다. AWS 관리자는 AWS Identity and Access Management(IAM) 자격 증명을 사용하거나 Microsoft Active Directory 또는 SAML Identity Provider와의 통합을 통해 AWS 콘솔에 로그인할 수 있습니다. AWS에 구축된 애플리케이션은 Amazon Cognito를 활용하여 최종 사용자가 리소스에 대해 인증을 받고 리소스에 액세스하도록 허용할 수 있습니다.
- **권한 부여 –** 지나치게 복잡한 관리 작업의 부담 없이 리소스 사용을 제한하는 유연한 정책을 통해 역할 기반 액세스 제어를 구현합니다. IAM은 모든 AWS 리소스에 대해 세분화된 권한 부여 정책을 제공합니다.
- **감사 및 거버넌스 –** 워크로드의 동작을 평가하고 워크로드가 규정 요건과 조직의 표준을 준수하는지 확인합니다. AWS CloudTrail을 통해 AWS API와의 상호 작용을 감사할 수 있으며, Amazon CloudWatch를 사용한 로그 집계를 통해

애플리케이션을 감사할 수 있습니다. AWS Config는 AWS 리소스가 조직의 표준에 맞게 구성되도록 합니다.

- **검증** – 애플리케이션 기능의 모든 측면을 테스트하고 의도 대로 작동하는지 확인합니다. 지속적 통합 및 지속적 전달(CI/CD)을 통해 최대한 검증을 자동화합니다.

모던 애플리케이션은 자주 철저히 테스트해야 하지만, 그로 인해 개발 속도가 저하되어서는 안 됩니다. 마찬가지로 개발자 권한을 제한해야 하지만 필요한 액세스 권한을 박탈해서는 안 됩니다. 전체 애플리케이션 수명 주기에 보안을 구축하고, 보안 프로세스 및 표준을 자동화한 후 지속적으로 재평가합니다.

마이크로서비스 아키텍처

모놀리식 애플리케이션이 커질수록 애플리케이션의 수정이나 기능 추가가 어렵고, 특정 변경 사항과 관련된 코드베이스의 해당 부분을 추적하기가 어려워집니다. 따라서 사소한 변경으로 인해 회귀 테스트에 시간이 많이 소요될 수 있으며, 새로운 기능의 개발 속도가 느려질 수 있습니다. 마이크로서비스 아키텍처와 약결합 구성 요소로 구축된 애플리케이션에서는 많은 새로운 기능과 버그 수정을 단일 서비스 수준에서 구현하고 훨씬 더 빠르게 릴리스할 수 있습니다.

모놀리식 레거시 애플리케이션을 사용하는 조직은 애플리케이션을 마이크로서비스로 재설계함으로써 조직의 민첩성과 유연성을 높일 수 있습니다. 각 서비스는 개별적으로 배포되며 모든 서비스가 함께 작동하여 모놀리식 시스템과 동일한 기능을 제공합니다. 마이크로서비스를 신속하게 구축, 수정 및 릴리스할 수 있으므로 실험과 혁신을 더 빠르게 진행할 수 있습니다. 또한 마이크로서비스를 구축하는 각 팀이 자체적인 설계, 개발, 배포 및 운영에 대한 명확한 소유권을 가질 수 있습니다.

이러한 약결합을 실현하려면 시스템의 마이크로서비스가 서로 통신해야 합니다. 서비스 간에 공유되는 데이터 스토어는 강한 결합, 숨은 종속성, 타이밍 문제, 확장 및 가용성

문제를 야기합니다. 게시된 API 또는 비동기식 메시지 대기열을 사용하여 별도의 서비스 간에 통신하는 것이 좋습니다. 프로세스를 대기열의 메시지로 연결된 여러 부분으로 분리하면 트랜잭션의 경계가 명확해지고 서비스를 좀 더 독립적으로 운영할 수 있습니다.

메시징 시스템의 다음과 같은 특성은 확장성, 복원력, 가용성, 일관성 및 분산 트랜잭션을 제공합니다.

- 신뢰성과 복원력이 뛰어난 메시지 전송 시스템
- 비차단, 단방향 작업
- 약결합 서비스
- 시스템의 다양한 논리적 구성 요소에 초점을 맞추고 각 구성 요소가 독립적으로 작동할 수 있도록 허용

이러한 요소를 활용하는 아키텍처는 강력한 API 및 비동기식 통신 채널을 손쉽게 노출할 수 있으므로, 각 서비스를 독립적으로 운영하고 자동화할 수 있으며, 결과적으로 안정성이 향상됩니다.

프로세스를 수행하기 위해 여러 마이크로서비스가 연결되어 있는 경우 포괄적인 단일 작업의 상태를 모니터링하는 방법이 있어야 합니다. 또한 필요한 모든 단계가 올바른 순서로 올바른 시간에 이루어지는지 확인해야 합니다. 상태 시스템을 사용하여 작업 상태를 모니터링하고 작업이 올바른 순서로 실행되는지 확인할 수 있습니다.

또한 서비스 간 전체 워크플로를 관리하고, 다양한 시간 제한, 취소, 장기 실행 작업의 하트비트, 세분화된 모니터링 및 감사를 구성하는 도구도 필요합니다. 이러한 유형의 도구를 사용하여 서비스를 관리하면 속도, 생산성 및 유연성이 향상됩니다. 모던 애플리케이션은 마이크로서비스가 적절한 타이밍에 따라 올바른 순서로 실행되도록 하기 위해 오케스트레이션 및 메시징 도구를 활용합니다. 오케스트레이션 도구를 사용하면 반복 가능한 방식으로 강력한 서비스를 손쉽게 구축할 수 있습니다. AWS Step Functions는 여러 서비스에 걸쳐 임의의 워크플로를 구성할 수 있는 완전관리형 도구입니다. 메시징 도구를

사용하면 서비스 간의 직접적인 종속성을 없애 안정성과 확장성을 개선할 수 있습니다. 워크로드에 따라 Amazon Simple Queue Service(Amazon SQS), Amazon CloudWatch Events, Amazon Kinesis 등 다양한 도구를 사용할 수 있습니다. 오케스트레이션 및 메시징 도구를 함께 사용하면 개발자가 워크플로 실행, 상태 관리 및 서비스 간 통신에 시간을 허비하지 않고 핵심 비즈니스 로직에 귀중한 업무 시간을 집중할 수 있습니다.

서버리스 기술 사용

조직의 애플리케이션을 실행하는 서버 및 운영 체제(Operating System, OS)를 운영하고 유지 관리하려면, 시스템 관리자가 OS 보안 패치를 적용하는 등의 단순하고 반복적인 작업을 수행하는 데 시간을 할애해야 합니다.
요청 볼륨을 기준으로 확장하는 것이 아니라, 가용성 및 내구성 요구 사항을 신중하게 고려하면서 최대 볼륨에 대비하여 서버를 사전 프로비저닝해야 합니다. 또한 사용량에 따라 비용을 지불하는 것이 아니라 이렇게 초과 프로비저닝된 인프라에 대해 사전에 비용을 지불해야 할 수 있습니다.

AWS Auto Scaling 및 AWS Systems Manager와 같은 서비스로 기존 VM 기반 인프라에서 이러한 부담을 줄일 수 있지만, 애초에 서버리스 기술을 기반으로 시스템을 구축하면 서버를 프로비저닝하고 관리할 필요가 없습니다. 관리자가 OS 패치에 시간을 허비하거나 드물게 발생하는 최대 사용량에 대비하여 사용하지 않는 리소스를 유지할 필요가 없습니다. 서버리스 애플리케이션은 각 구성 요소의 정확한 수요에 맞춰 확장됩니다. 안정성 및 내결함성 또한 대부분 기본적으로 내장되어 있어 시스템의 이러한 측면에 요구되는 설계 및 운영 시간을 크게 절약할 수 있습니다. 처음부터 서버리스 기술을 기반으로 모던 애플리케이션을 구축하면 애플리케이션 구축, 배포 및 실행의 전체 수명 주기를 안전하게 유지할 수 있습니다. 운영의 복잡성을 없애면 개발자가 고객을 만족시킬 제품을 개발하는 데 시간과 노력을 집중할 수 있습니다.

AWS는 AWS Lambda[3], AWS Fargate[4] 등의 서버리스 컴퓨팅 서비스를 제공합니다. 객체 스토리지용 Amazon Simple Storage Service(Amazon S3)[5]뿐만 아니라, 이제 빠르고

유연한 NoSQL 데이터베이스인 Amazon DynamoDB[6]과 Amazon Aurora용 온디맨드 및 Auto Scaling 구성인 Amazon Aurora Serverless[7]라는 두 가지 서버리스 데이터베이스 옵션까지 제공합니다. 포괄적인 서버리스 애플리케이션을 구축하려는 경우 컴퓨팅, 데이터베이스 및 스토리지 서비스로는 충분하지 않을 수 있습니다. 이 경우 API 관리, 메시징 및 오케스트레이션부터 문제 해결 및 모니터링에 이르기까지, 전체 워크로드에 걸쳐 다른 서버리스 AWS 제공 서비스[8]를 활용할 수 있습니다.

CI/CD로 배포 자동화

기업은 신속하게 혁신을 실현하여 가능한 한 빨리 고객에게 가능한 최고의 가치를 제공하기 위해 노력합니다. 이를 위해 모던 애플리케이션에서는 지속적 통합 및 지속적 전달(CI/CD) 기술을 사용하여 전체 릴리스 프로세스(테스트 구축 및 실행, 아티팩트를 스테이징으로 승격, 프로덕션에 최종 배포)를 자동화합니다.
또한 CI/CD를 통해 알려진 취약성 검사, 정적 분석 수행 등의 특정 보안 제어 기능을 자동화할 수도 있습니다. 전체 CI/CD 파이프라인은 여러 번의 품질 검사 및 제어 수단으로 구성할 수 있으며, 새로운 코드를 프로덕션에 적용하려면 이 같은 게이트와 제어 수단을 모두 성공적으로 통과해야 합니다.

전체 구축/테스트/배포 프로세스를 자동화하면 재현 가능성이 높아질 뿐만 아니라 속도도 더 빨라집니다. 또한 훨씬 더 빈번하게(하루에 여러 번) 수행할 수 있게 됩니다. 즉, 개별 배포 건에 포함되는 변경 사항이 더 적고 위험도 줄어듭니다. CI/CD는 모든 요소가 관여하는 위험성이 높은 방식으로 배포가 실행되는 것이 아니라 일상적인 작업으로서 프로덕션에 배포할 수 있게 해줍니다. 마지막으로, 코드가 커밋된 시점부터 배포 시점까지의 시간이 수동 프로세스보다 훨씬 짧기 때문에 우선 순위가 높은 보안 수정 사항이나 구성 변경을 더 이상 특별한 핫 패치를 통해 적용할 필요 없이 표준 파이프라인을 통해 적용할 수 있습니다.

AWS 고객은 오픈 소스 옵션 및 타사 Marketplace 제공 서비스는 물론, AWS CodeBuild, AWS CodePipeline 및 AWS CodeDeploy와 같은 완전관리형 CI/CD 서비스도 활용할 수 있습니다.

코드형 인프라(Infrastructure as Code, IaC)

CI/CD의 이점을 최대한 활용하려면 전체 애플리케이션과 코드형 인프라(IaC)에 대한 모델을 만들어야 합니다. 코드형 인프라를 모델링하면, 해당 모델을 표준 애플리케이션 개발 수명 주기에 통합하고, CI/CD 파이프라인에서 인프라 변경 작업을 실행하고, 구성 오류 감소와 프로비저닝 속도 향상과 같은 추가적인 이점을 얻을 수 있습니다. AWS는 다양한 IaC 도구를 제공합니다. 그 중 하나인 AWS CloudFormation[9]은 사용자가 간단한 템플릿 파일에 필요한 모든 클라우드 인프라를 지정하면 인프라를 자동으로 프로비저닝해 주는 서비스입니다. 다음으로, AWS 서버리스 애플리케이션 모델(SAM)[10]은 AWS CloudFormation을 기반으로 서버리스 애플리케이션 구축을 위한 추가 도구와 편리한 기능을 제공합니다. AWS Cloud Development Kit(CDK)[11]는 선택한 언어를 사용하여 코드로 클라우드 인프라를 설계한 다음 CloudFormation을 사용하여 프로비저닝하는 프레임워크를 제공하는 도구입니다.

모니터링 및 로깅

모던 애플리케이션 개발자는 모니터링 및 로깅 도구를 사용하여 런타임에 애플리케이션의 동작을 모니터링하고 해당 데이터를 바탕으로 고객 경험을 유지 관리하거나 개선해야 합니다. 모던 디지털 제품에서는 애플리케이션 로그, 모바일 디바이스의 데이터, 웹 클릭 스트림, IoT 센서 데이터 또는 기타 사용량 데이터를 비롯한 다양한 데이터 유형을 모니터링해야 할 수 있습니다. 모던 애플리케이션 개발자는 제품을 계속 확장하고 강화하는 데 이 같은 데이터를 십분 활용해야 합니다.

AWS에서는 Amazon CloudWatch를 사용하여 모든 애플리케이션 구성 요소에 대한 모니터링, 로깅 및 경보를 설정할 수 있습니다. 로깅에 대한 자세한 내용은 로그 집계를 참조하십시오.

모던 애플리케이션 체크리스트

다음 정보를 사용하여 애플리케이션의 현대화 수준을 확인합니다.

- 전체 애플리케이션 수명 주기에 걸쳐 보안 및 규정 준수 기본 지원
- 애플리케이션을 일련의 마이크로서비스로 구성 가능
- 서버리스 기술 최대한 활용 가능
- 고품질 기능을 신속하게 제공하기 위해 CI/CD 사용 가능
- 인프라를 코드로 개발 및 배포 가능
- 애플리케이션의 동작을 파악하는 데 모니터링 도구 사용

모던 애플리케이션 설계 패턴

모던 애플리케이션 개발에 있어서는 패턴을 사용하여 애플리케이션을 설계하고 구현하는 것이 모범 사례입니다. AWS 서비스를 이러한 애플리케이션의 빌딩 블록으로 사용하면 구현 작업 부담을 크게 줄이고 안정성과 가용성을 실현할 수 있습니다. 결과적으로 개발자는 애플리케이션에 가치를 더하는 비즈니스 로직에 집중할 수 있습니다.

AWS 서비스를 사용하여 마이크로서비스 아키텍처 구현

마이크로서비스에 공통적으로 적용되는 패턴을 사용하고, 모범 사례를 따르고, AWS 서비스를 사용하여 구현할 수 있습니다.

API 게이트웨이

API 게이트웨이 패턴은 백엔드 서비스에 대한 호출이 많고, 클라이언트 인터페이스 또는 디바이스 유형에 따라 제공되는 콘텐츠가 달라지는 경우에 사용할 수 있습니다. API 게이트웨이는 통합 API를 통해 여러 백엔드 서비스를 통합하고 각 디바이스에 필요한 콘텐츠를 제공할 수 있습니다.

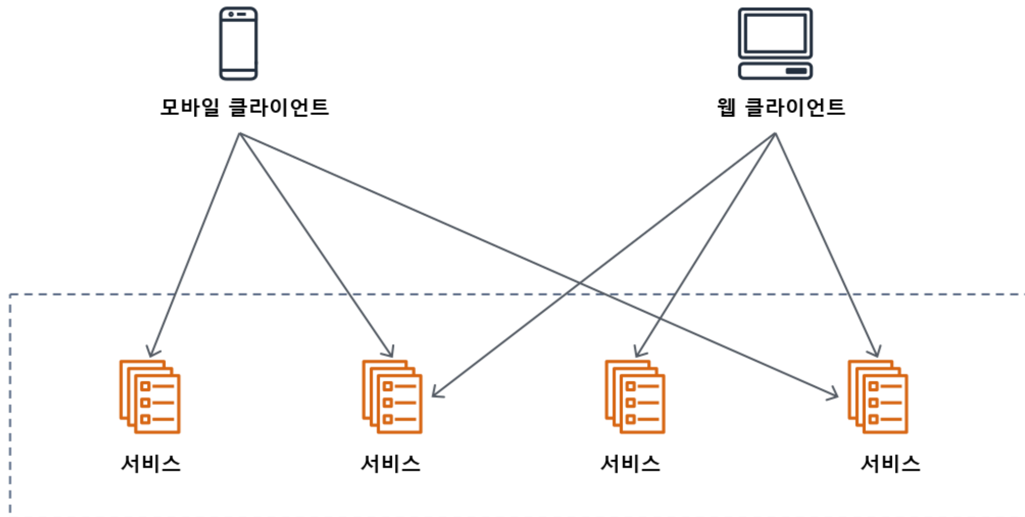

그림 1 – API 게이트웨이를 사용하지 않는 모바일 디바이스 및 컴퓨터 브라우저와 서비스 간 통신의 예

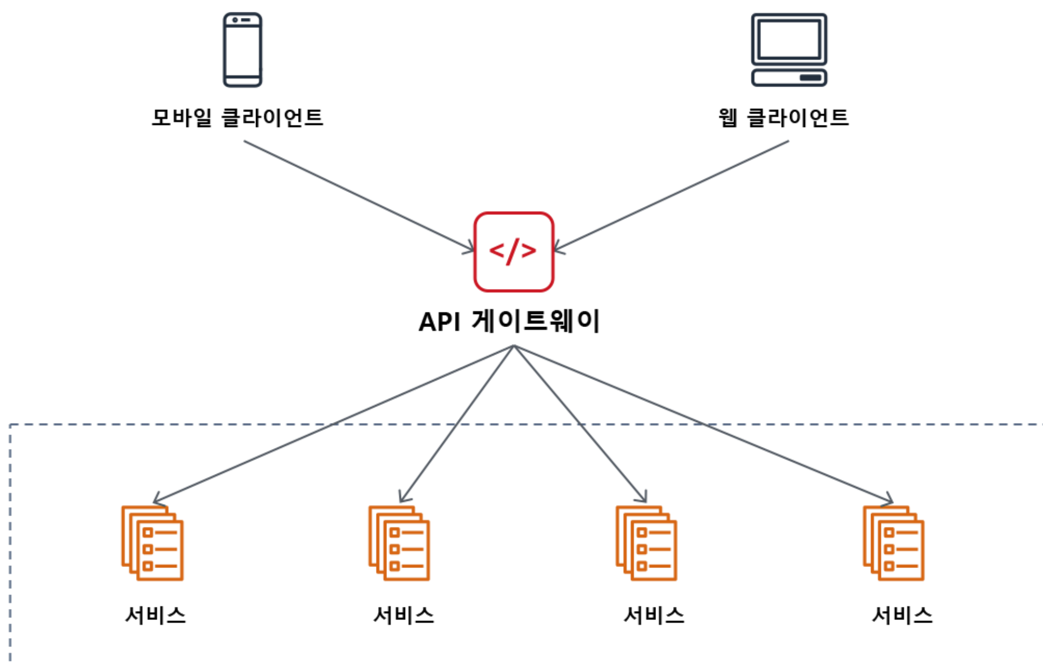

그림 2 – API 게이트웨이를 사용한 모바일 디바이스 및 컴퓨터 브라우저와 서비스 간 통신의 예

AWS 클라우드에서 API 게이트웨이 패턴을 사용하려는 경우, Amazon API Gateway[12]를 사용하여 백엔드 엔드포인트와 통합할 수 있습니다. 또한 Amazon API Gateway를 사용하면 규모와 관계없이 REST 또는 WebSocket API를 생성, 게시, 유지 관리, 모니터링 및 보호할 수 있습니다.

Amazon API Gateway는 조절, 캐싱, 로깅, API 토큰, Amazon Cognito와 통합된 인증 또는 권한 부여, 사용자 지정 권한 부여자, 다른 AWS 서비스에 대한 요청 프록시 등 프로덕션 수준의 API에 필요한 다른 기능을 많이 제공합니다. Amazon API Gateway에서 프록시 요청을 전송하는 필수 AWS 서비스 중 하나는 AWS Lambda입니다. AWS Lambda는 서버 인프라를 관리하지 않으면서 임의의 웹 서비스를 생성하는 데 기반이 됩니다.

Amazon API Gateway는 AWS에서 관리하므로 운영 및 유지 관리에 대한 부담이 없습니다. Amazon API Gateway를 사용하면 보안, 안정성 및 가용성이 향상되므로 개발자는 핵심 애플리케이션 기능에 더 많은 시간을 할애할 수 있습니다.

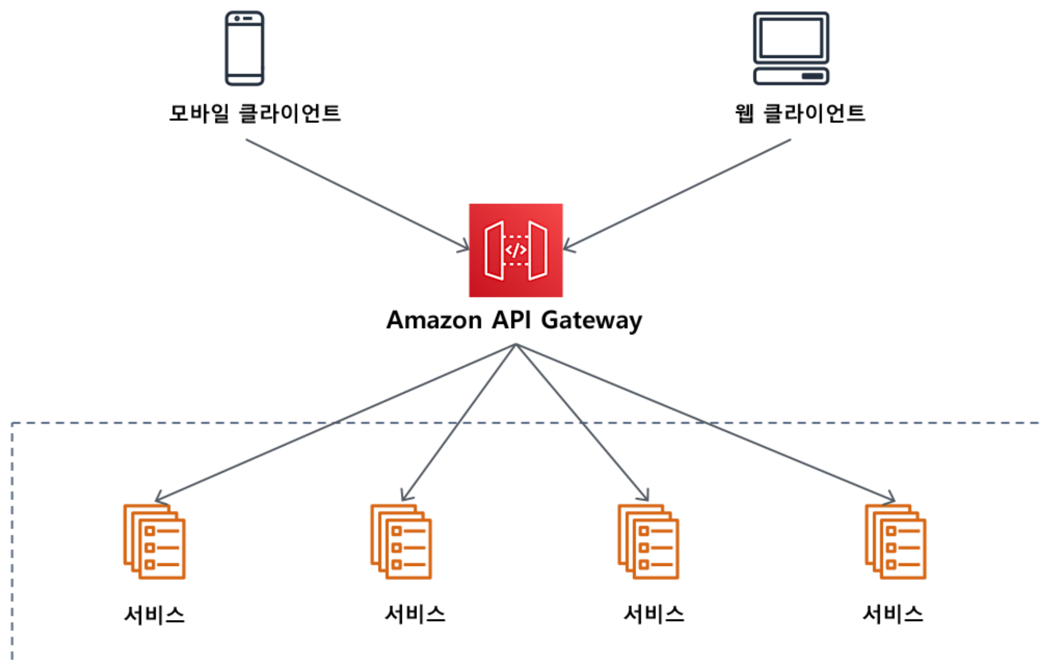

그림 3 – Amazon API Gateway를 사용한 모바일 디바이스 및 컴퓨터 브라우저와 서비스 간 통신의 예

서비스 검색 및 서비스 레지스트리

시스템에 여러 마이크로서비스가 포함된 경우, 각 서비스는 해당 서비스와 종속 관계가 있는 다른 서비스의 위치를 찾을 수 있어야 합니다. 마이크로서비스는 확장 가능하고 탄력적이어야 하며, 구성 요소에서 장애가 발생할 경우 지속적인 가용성을 보장하기 위해 새로운 인스턴스 또는 컨테이너를 온라인 상태로 전환해야 합니다. 즉, 마이크로서비스에

있는 인스턴스 또는 컨테이너의 IP 주소는 계속 변경될 수 있습니다. 서비스를 구성하는 각 인스턴스의 가용성도 지속적으로 모니터링해야 합니다. 로드 밸런서를 사용하여 안정적이고 사용 가능한 엔드포인트를 제공할 수 있으며, 이는 일반적으로 퍼블릭 웹 엔드포인트에 적합한 옵션입니다. 하지만 로드 밸런서에는 추가 컴퓨팅 리소스가 필요하며 지연을 유발합니다. 마이크로서비스 간 호출과 마찬가지로 클라이언트를 제어할 수 있는 경우, 서비스 검색 패턴을 사용하는 것이 더 효율적일 수 있습니다. 이 패턴은 클라이언트 측 로드 밸런싱이라고 할 수 있습니다.

서비스 검색 패턴에서는 검색할 서비스에 대한 정보를 어딘가에 등록해야 합니다. 서비스 레지스트리는 호출 대상 서비스가 각 개별 컨테이너 또는 인스턴스 시작 시에 자신에 대한 정보를 저장할 수 있는 중앙 위치입니다.

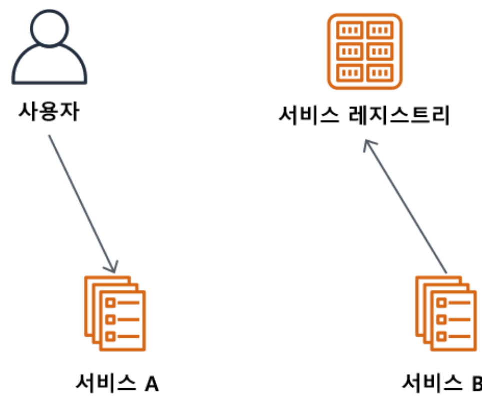

그림 4 – 서비스 레지스트리 패턴의 예

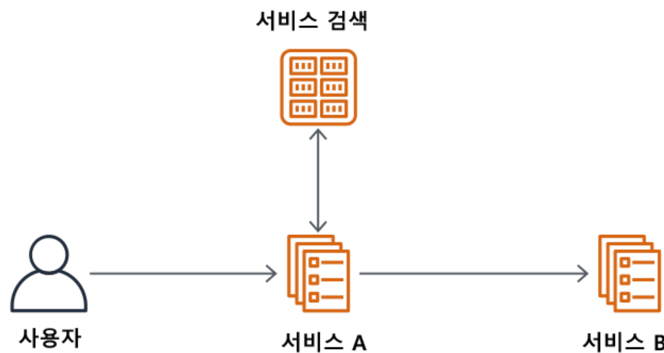

그림 5 – 서비스 검색 패턴의 예

AWS Cloud Map을 사용하여 AWS 클라우드에 서비스 레지스트리 및 서비스 검색 패턴을 구현할 수 있습니다. AWS Cloud Map은 클라이언트가 DNS를 사용하여 서비스 인스턴스의 IP 주소 및 포트 조합을 조회하고 HTTP 기반 서비스 검색 API를 통해 URL 또는 Amazon 리소스 이름(Amazon Resource Name, ARN)과 같은 추상 엔드포인트를 동적으로 검색할 수 있게 해주는 완전관리형 서비스입니다.

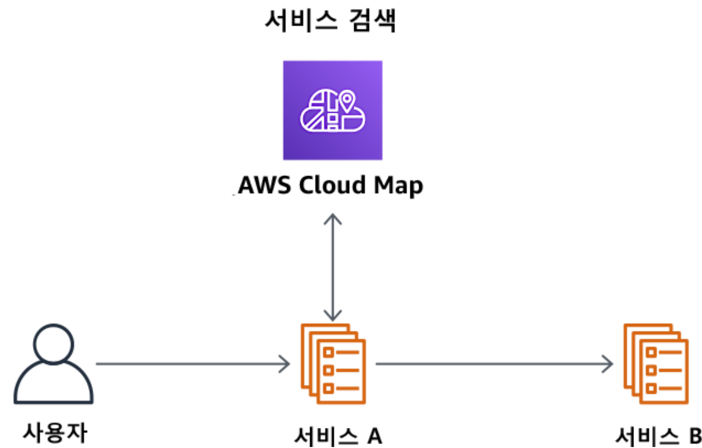

그림 6 – AWS Cloud Map을 사용한 서비스 레지스트리 및 서비스 검색 패턴의 예

회로 차단기(Circuit Breaker)

회로 차단기 패턴은 애플리케이션의 마이크로서비스 간 호출을 규제합니다. 애플리케이션의 마이크로서비스는 사용자 요청에 응답하기 위해 서로 호출합니다. 서비스 A가 서비스 B로 호출을 전송하는데 서비스 B의 회신 호출이 지연되거나 오류가 발생할 경우 서비스 A가 사용자에게 오류를 반환합니다. 이때 서비스 A가 오류를 반환하는 대신 호출을 재시도하면 더 나은 사용자 경험을 제공할 수 있지만, 재시도는 추가 로드 및 긴 지연을 발생시킬 수 있으며 결국 사용자에게 오류를 반환하는 결과로 끝날 수 있습니다. 대신 서비스 A는 서비스 B가 작동하지 않는 상태임을 인식하여 가능한 경우 정상적으로 성능 저하된 상태로 작동해야 합니다.

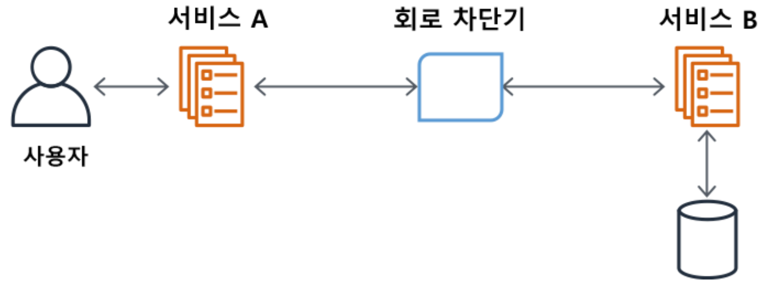

그림 7 – 마이크로서비스 간의 호출이 반환된 회로 차단기 패턴의 예

회로 차단기 패턴에서 다른 서비스에 대한 호출이 예상보다 오래 걸리거나 오류가 반환될 경우, 회로 차단기는 발생 횟수를 계속 계산하여 사용자가 구성한 한도를 초과할 경우 개방 상태로 변경합니다. 개방 상태일 때 회로 차단기는 다운스트림 서비스를 호출하지 않고 호출자에게 즉시 오류를 반환합니다.

고정된 시간이 경과하면 회로 차단기가 폐쇄 상태로 돌아가 다운스트림 서비스에 대한 호출이 정상적으로 실행됩니다.

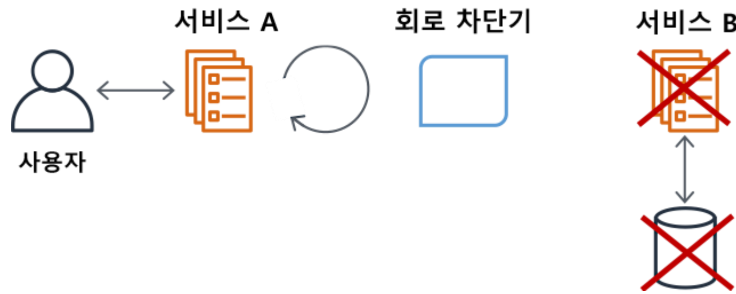

그림 8 – 사용자에게 오류를 즉시 반환하는 회로 차단기 패턴의 예

이전에는 서비스 코드에서 라이브러리 또는 프레임워크를 사용하여 회로 차단기를 구현하는 것이 모범 사례였지만, 이제는 사이드카(Sidecar)를 사용하여 컨테이너식 마이크로서비스에서 처리하는 경우가 많습니다. 사이드카는 코어 서비스를 노출하는 기본 컨테이너와 함께 실행되는 별도의 헬퍼 컨테이너입니다. Envoy Proxy[13]는 사이드카의 대표적인 예입니다. Envoy Proxy는 독립적으로 배포할 수 있지만 서비스 메시의 일부로 배포되는 경우가 많습니다. 이 배포 유형에서 Envoy Proxy는 데이터 영역에 해당하고 AWS App Mesh 또는 Istio 같은 도구는 제어 영역에 해당합니다.

Command-Query Responsibility Segregation

CQRS(Command Query Responsibility Segregation)는 시스템의 데이터 변형 또는 명령 부분을 쿼리 부분과 분리하는 것을 말합니다. 업데이트 및 쿼리는 일반적으로 단일 데이터 스토어를 사용하여 수행됩니다. 처리량, 지연 시간 또는 일관성 요구 사항이 서로 다른 경우 CQRS를 사용하여 이 두 워크로드를 분리할 수 있습니다. 명령 기능과 쿼리 기능을 분리하면 두 기능을 독립적으로 확장할 수 있습니다. 예를 들어 수평적으로 확장 가능한 읽기 전용 복제본에 쿼리를 전송할 수 있습니다. 업데이트 및 쿼리에 서로 다른 데이터 모델과 데이터 스토어를 사용하여 명령 기능과 쿼리 기능의 분리를 강화할 수 있습니다. ORM(객체 관계형 매핑)을 통해 관계형 데이터베이스의 정규화된 모델에 대한 쓰기 작업을 수행하고, API에 요구되는 것과 동일한 형식(예: 데이터 전송 객체 또는 DTO)으로 데이터를 저장하는 비정규화된 데이터베이스에 대해 쿼리를 수행하여 처리 오버헤드를 줄일 수 있습니다.

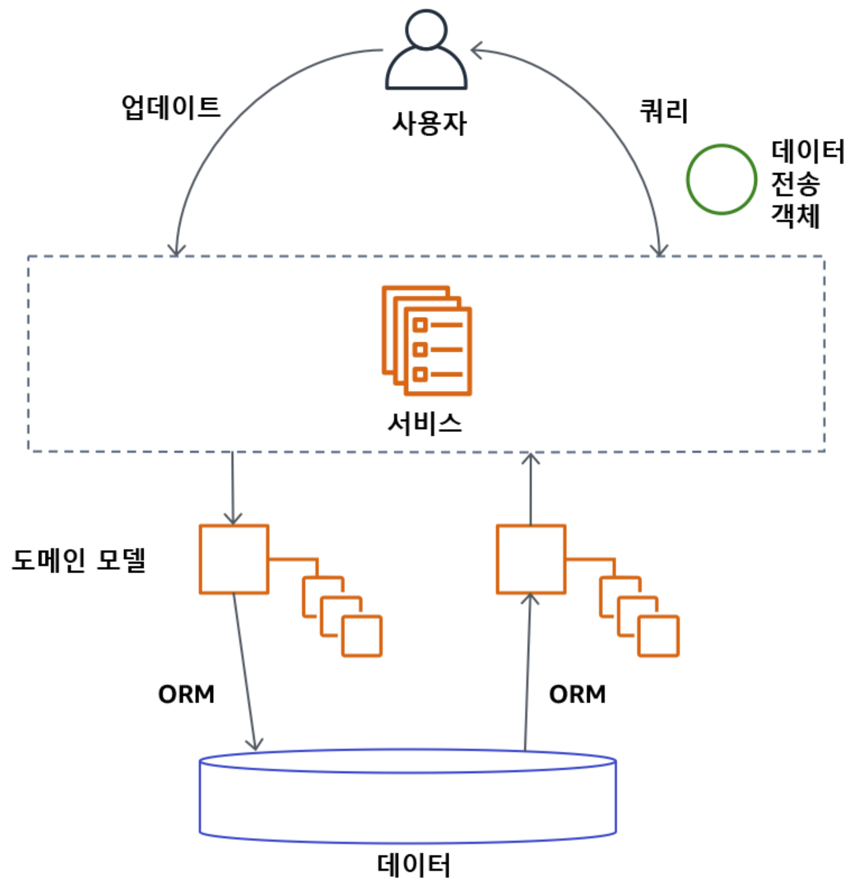

그림 9 – 단일 데이터 스토어 및 ORM을 사용하는 업데이트 및 쿼리가 포함된 아키텍처의 예

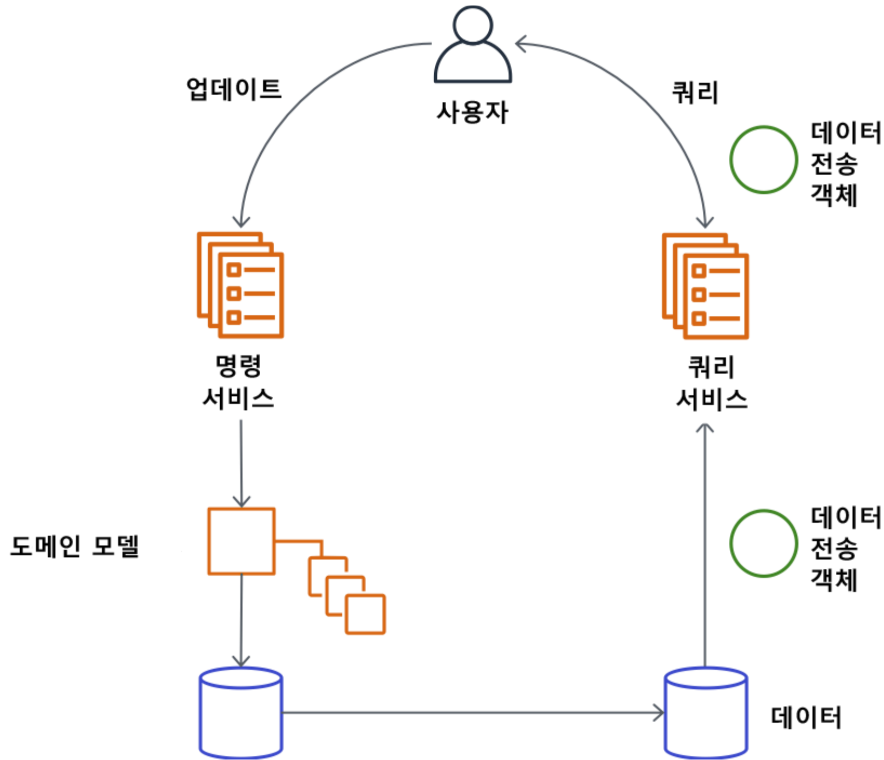

그림 10 – 명령 및 쿼리 워크로드가 분리되어 있고 2개의 데이터 스토어가 있는 CQRS 아키텍처의 예

이 예제에서는 아키텍처를 최적화하여 관계형 데이터베이스에서 일관된 쓰기와 지연 시간이 매우 짧은 읽기를 구현하고 있지만, 매우 높은 쓰기 처리량과 유연한 쿼리 기능을 제공하도록 최적화할 수도 있습니다. 이 경우 Amazon DynamoDB와 같은 NoSQL 데이터 스토어를 사용하여, 잘 정의된 특정 액세스 패턴을 통해 워크로드에서 데이터를 추가할 때 높은 쓰기 확장성을 확보할 수 있습니다. 그런 다음 Amazon Aurora와 같은 관계형 데이터베이스를 사용하여 복잡한 일회성 쿼리 기능을 제공할 수 있습니다.

이 옵션에서는 데이터를 AWS Lambda 함수로 전송하는 Amazon DynamoDB 스트림을 사용하여 적절한 업데이트를 적용함으로써 Amazon Aurora의 데이터를 최신 상태로 유지할 수 있습니다.

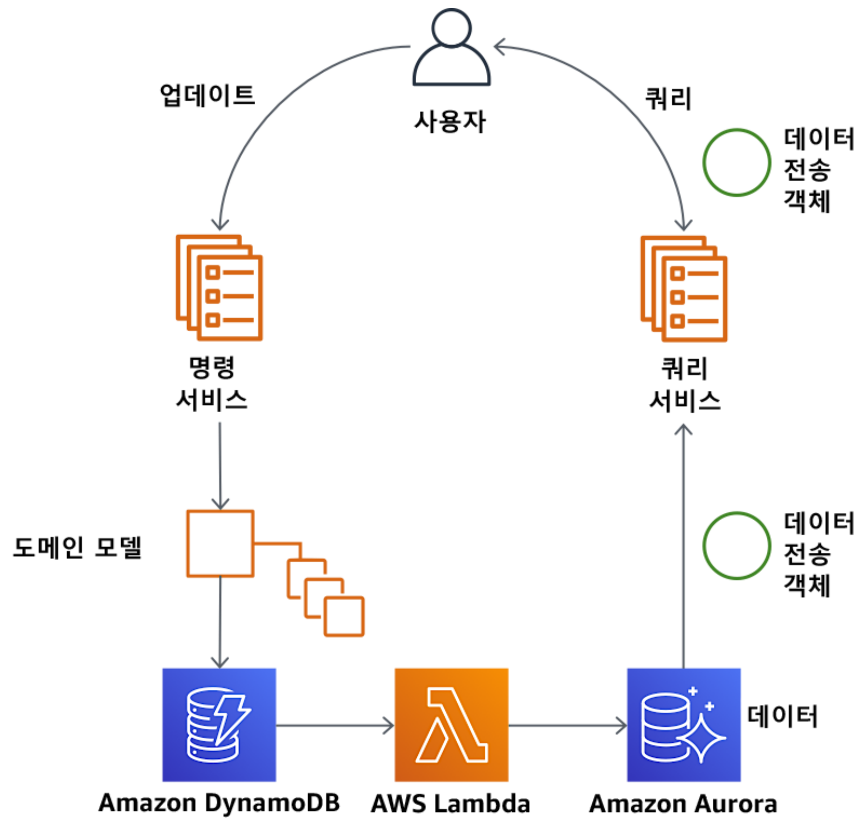

그림 11 – DynamoDB, Lambda 및 Aurora를 사용하는 AWS의 CQRS 아키텍처 예

CQRS 아키텍처의 명령 부분을 이벤트 소싱 패턴과 결합할 수도 있습니다(다음 섹션 참조). 이러한 패턴을 결합하면 업데이트 이벤트를 재생하여 서비스 쿼리 데이터 모델을 최신 애플리케이션 상태로 다시 빌드할 수 있습니다. 일반적으로 CQRS 패턴을 사용할 경우 쿼리한 데이터 스토어와 데이터가 쓰여진 데이터 스토어 간에 최종적 일관성이 발생된다는 점을 기억하는 것이 중요합니다.

이벤트 소싱

이벤트 소싱 패턴을 사용하면 데이터 스토어를 직접 업데이트하는 것이 아니라 비즈니스 로직에 중요한 이벤트(예: 처리 중인 주문, 처리 중인 신용 조회, 처리 또는 배송 중인 주문)가 내구성 있는 이벤트 로그에 추가됩니다. 각 이벤트 레코드가 개별적으로 저장되므로 모든 업데이트는 원자성(분할 불가, 축소 불가)이 보장됩니다.

저장된 이벤트를 간단히 재처리하여 언제든지 애플리케이션 상태를 다시 빌드할 수 있다는 것이 이 패턴의 주요 특징입니다. 데이터가 데이터 스토어에 직접 업데이트하는 것이 아니라 일련의 이벤트 형태로 저장되므로 다양한 서비스가 이벤트 스토어에서 이벤트를 재생하여 해당 데이터 스토어의 적절한 상태를 계산할 수 있습니다. 이 패턴은 앞서 설명한 CQRS 패턴과 함께 사용할 때 효과적으로 작동합니다. 특히 명령 데이터 스토어와 쿼리 데이터 스토어의 스키마가 서로 다른지 여부에 관계없이 이벤트의 데이터를 재현할 수 있기 때문입니다.

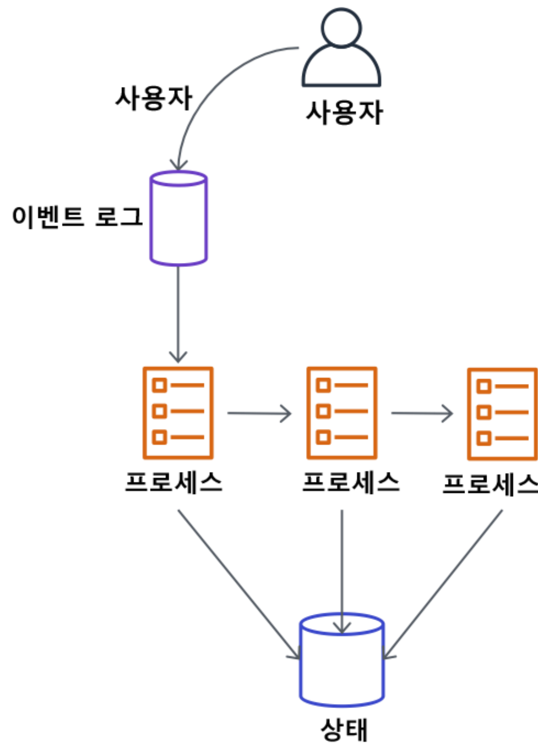

그림 12 – 이벤트 소싱 패턴의 예

이벤트 소싱 패턴에서는 이벤트 메시지를 저장한 후 나중에 재생하게 되므로 메시지 저장 및 검색을 위한 몇 가지 메커니즘이 필요합니다. AWS 클라우드에서 이 패턴을 사용하려는 경우 사용 사례에 따라 Amazon Kinesis Data Streams[14], Amazon Simple Queue Service(SQS)[15], Amazon MQ[16] 또는 Amazon Managed Streaming for Kafka(Amazon MSK)[17]를 이용할 수 있습니다. 이벤트 소싱 패턴에서는 시스템을 변경하는 각 이벤트가 먼저 메시지 대기열에 저장되고, 해당 이벤트에 따라 애플리케이션 상태가 업데이트됩니다. 예를 들어 이벤트를

Amazon Kinesis 스트림에 레코드로 기록한 다음 AWS Lambda를 기반으로 구축된 서비스를 통해 레코드를 검색하고 자체 데이터 스토어에서 업데이트를 수행할 수 있습니다.

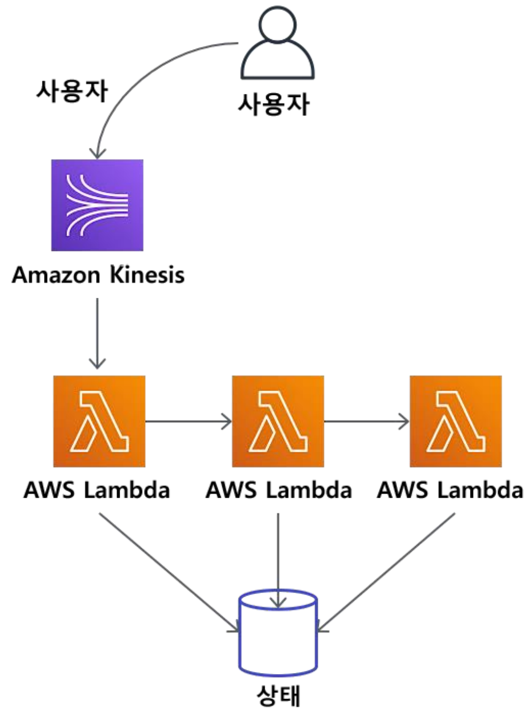

그림 13 – Amazon Kinesis 및 AWS Lambda를 사용한 이벤트 소싱 패턴의 예

이벤트 소스 하나에서 여러 대상으로 확장하는 방식이 유용한 경우도 있습니다. 여러 소비자가 스트림에서 데이터를 검색할 수 있도록 허용하는 Amazon Kinesis Data Streams를 사용하여 이를 직접 수행할 수 있습니다. 또한 Amazon Simple Notification Service(Amazon SNS)를 사용하여 여러 Lambda 함수로 확장할 수 있습니다. 이 경우 모든 함수가 동일한 주제를 수신하고 Kinesis에서 다른 상태 저장 구성 요소로 이벤트 데이터를 전파할 수 있습니다. 이 구성에서는 Amazon SNS와 지정된 Lambda 함수 사이에 Amazon Simple Queue Service(Amazon SQS) 대기열을 추가하여 Lambda 함수의 실행 이유를 지정할 수 있습니다.

코레오그래피

기업의 웹 사이트에 계정을 생성하는 새 고객은 프로필 정보를 저장하고, 환영 이메일을 수신하고, 사이트에서 사용할 초기 포인트를 지급 받아야 할 수 있습니다. 이러한 모든 활동은 서로 다른 서비스에 의해 구현됩니다.

마이크로서비스 간에 이러한 작업을 실행하기 위한 구현 방법으로는 오케스트레이션 패턴과 코레오그래피 패턴이 있습니다. 교향곡에서 지휘자와 오케스트라 간의 관계와 유사한 오케스트레이션 패턴에서는 중앙의 서비스가 다른 서비스에 명령을 내리고 전체 프로세스가 완료되도록 합니다. 코레오그래피 패턴에서는 무용수들이 무용의 안무를 배운 후 독립적으로 움직이는 것처럼, 각 서비스가 특정 이벤트에 대응하여 독립적으로 실행됩니다.

코레오그래피 패턴을 사용하면 필요한 모든 정보가 포함된 초기 이벤트를 단일 메시지로 저장하고 초기 트랜잭션을 마칩니다.
그런 다음 다른 서비스가 해당 메시지를 비동기식으로 검색하고 맡은 작업을 완료할 수 있습니다. 이 아키텍처를 사용하면 서비스가 약결합되어 서로에게 직접적인 영향을 미치지 않습니다. 메시지 저장과 검색 간의 비동기식 관계는 확장성과 안정성의 이점도 제공합니다.

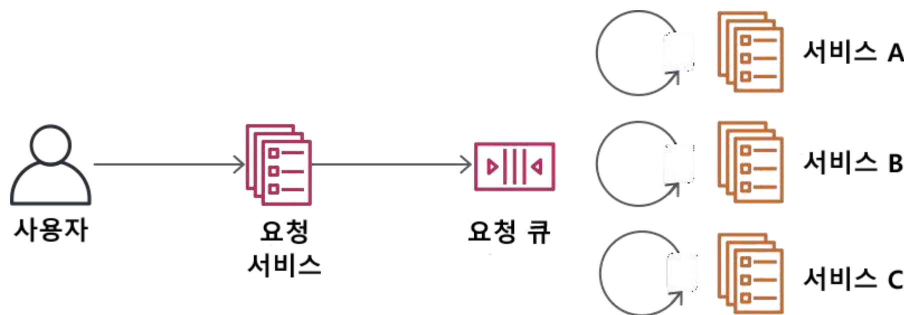

그림 14 – 코레오그래피 패턴의 예

AWS 클라우드에서 코레오그래피 패턴을 구현할 때는 Amazon Kinesis와 AWS Lambda를 사용하거나, 요구 사항에 따라 Amazon Simple Notification Service(Amazon SNS), Amazon Simple Queue Service(Amazon SQS) 및 AWS Lambda를 조합하여 사용할 수 있습니다.

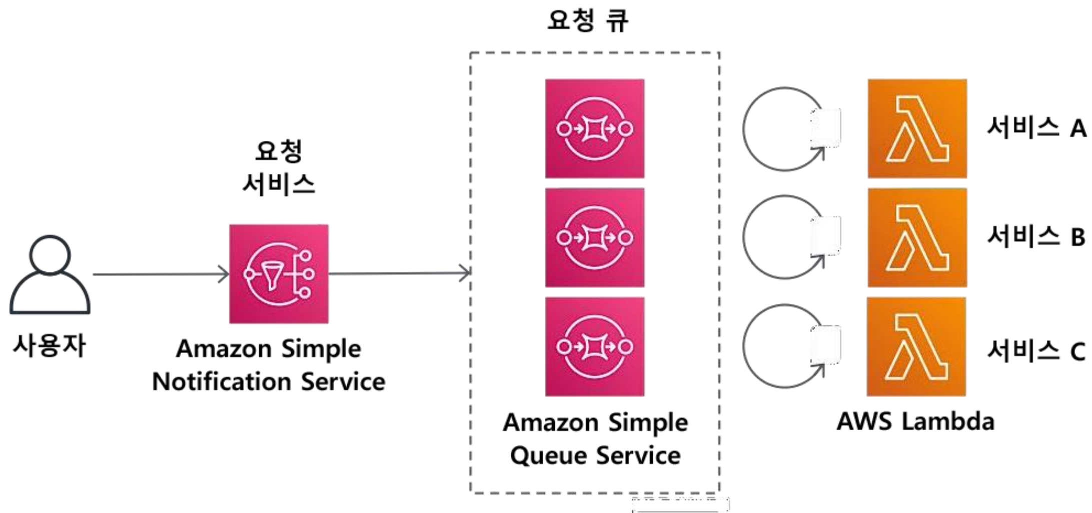

그림 15 – Amazon SNS, Amazon SQS 및 AWS Lambda를 사용한 코레오그래피 패턴의 예

로그 집계

시스템이 복잡할수록 유용한 로그의 중요성이 커집니다. 문제는 로그가 여러 서비스에 분산되어 있는 경우 전체 시스템을 보여 주는 통합 보기를 확보하기가 어렵다는 것입니다. 따라서 로그를 통합 관리하고 검색할 수 있는 중앙 집중식 기능을 갖추는 것이 필수적입니다. 지표 수집도 중요합니다. 마이크로서비스 아키텍처에서는 주어진 요청을 처리하기 위해 다양한 서비스를 호출해야 할 수 있으므로 모놀리식과 비교하여 성능 저하 또는 오류의 원인을 찾기가 더 어려울 수 있습니다. 따라서 로그 및 실행 시간 지표를 중앙에서 집계하는 것이 중요합니다.

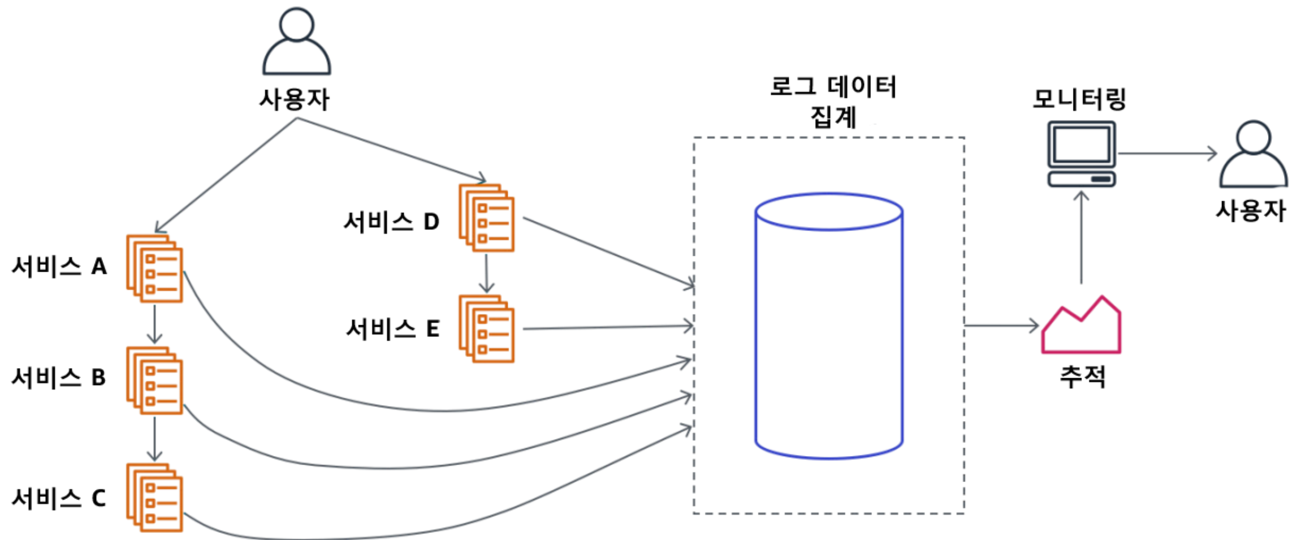

그림 16 - 로그 집계를 사용하는 아키텍처의 예

AWS 클라우드에서 집계 로깅에는 Amazon CloudWatch Logs[18]를 사용할 수 있습니다. AWS Lambda를 사용하여 마이크로서비스를 구현하면 stdout에 작성하는 모든 내용이 CloudWatch Logs로 전송됩니다. 또한 Amazon Elastic Container Service(ECS)와 AWS Fargate도 awslogs 로그 드라이버를 통해, stdout에 작성된 모든 내용을 Amazon CloudWatch Logs로 전송할 수 있습니다. Amazon Elastic Kubernetes Service(Amazon EKS)를 사용하는 경우 Fluentd[19] 또는 Fluent Bit[20]와 함께 사이드카 패턴을 사용하여 Amazon CloudWatch Logs로 로그를 전송할 수 있습니다. CloudWatch Container Insights[21]는 Amazon ECS 및 AWS Fargate 또는 Amazon EKS에서 실행되는 컨테이너식 애플리케이션의 로그와 지표를 CloudWatch로 전송하는 데에도 사용할 수 있습니다.

실행 시간을 추적하거나 서비스 간 호출 오류를 추적하는 데 AWS X- Ray[22]를 사용할 수 있습니다. X-Ray를 통해 애플리케이션과 그 기본 서비스가 어떻게 작동하는지 이해함으로써 작동 관련 문제 및 오류의 근본 원인을 알아내 해결할 수 있습니다. X-Ray는 요청이 애플리케이션을 통과함에 따라 요청에 대한 엔드 투 엔드 뷰를 제공하고 애플리케이션의 기본 구성 요소를 맵으로 보여줍니다.

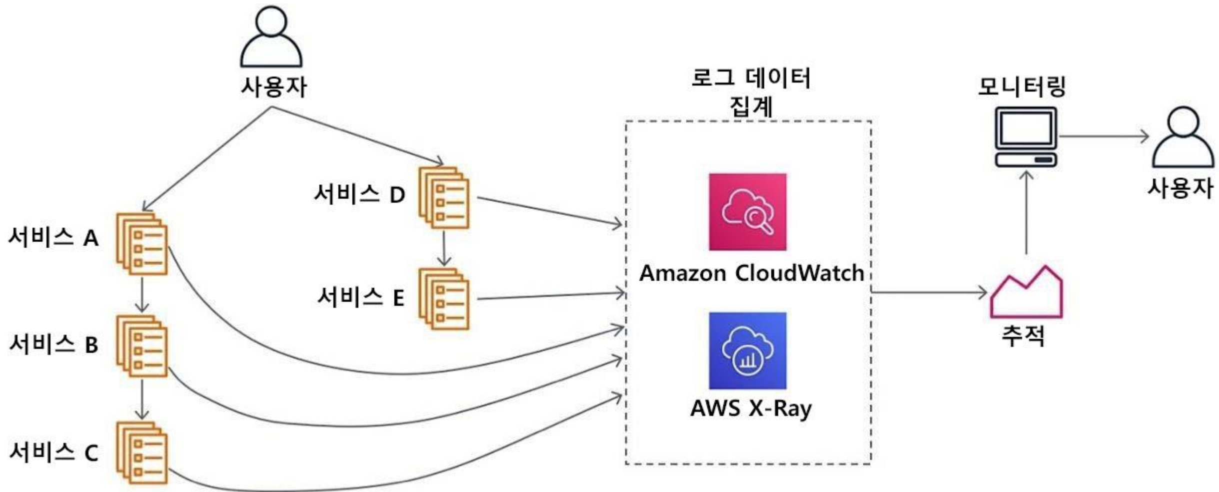

그림 17 – Amazon CloudWatch 및 AWS X-Ray를 사용한 로그 집계 아키텍처의 예

폴리글랏 지속성(Persistence)

마이크로서비스 아키텍처에서 각 서비스는 공개 API를 노출하되, 구현 세부 정보는 다른 서비스가 알지 못하도록 숨겨야 합니다. 이 아키텍처를 사용하면 특정 서비스를 구축하는 팀이 API 계약을 유지하는 한 다른 서비스가 수정된 코드에 의존하는지 여부를 신경 쓰지 않고 서비스의 내부 코드를 자유롭게 수정할 수 있습니다.
또한 이러한 팀은 필요할 때 자체 서비스에 대해 배포 작업을 수행할 수 있으며 선호하는 프로그래밍 언어 및 데이터베이스로 서비스를 구현할 수 있습니다. 폴리글랏 지속성을 사용할 경우 서비스의 데이터 액세스 패턴과 기타 요구 사항에 따라 적합한 데이터 스토리지 기술을 선택할 수 있습니다.

모든 서비스 팀이 어쩔 수 없이 동일한 데이터 스토리지 기술을 사용할 경우, 데이터 스토어가 특정 상황에 적합하지 않을 경우 구현 문제 또는 성능 저하가 발생할 수 있습니다. 팀에서 요구 사항에 가장 적합한 데이터 스토어를 선택할 수 있으면, 서비스를 좀 더 쉽게 구현하고 더 나은 성능과 확장성을 실현할 수 있습니다.

AWS는 다음 표에 요약된 바와 같이 폴리글랏 지속성을 지원하는 다양한 데이터 스토리지 서비스를 제공합니다.

표 1 – 폴리글랏 지속성을 지원하는 AWS 데이터 스토리지 서비ㅍ

데이터 스토어	기능
Amazon DynamoDB	규모에 관계없이 10밀리초 미만의 성능을 제공하는 키–값 및 문서 데이터베이스입니다. 인터넷 규모의 애플리케이션을 위한 보안, 백업 및 복원, 인메모리 캐싱이 내장된 탁월한 내구력의 완전관리형 다중 리전 다중 마스터 데이터베이스입니다. DynamoDB는 하루에 10조 개가 넘는 요청을 처리할 수 있으며 초당 2천만 개 이상의 요청까지 지원할 수 있습니다.
Amazon Aurora 및 Amazon Relational Database Service(RDS)	Amazon Aurora는 MySQL 및 PostgreSQL과 호환되는 클라우드용 관계형 데이터베이스 로서 기존 엔터프라이즈 데이터베이스의 성능 및 가용성과 오픈 소스 데이터베이스의 단순성 및 비용 효율성을 결합한 것입니다. Amazon Relational Database Service(Amazon RDS)[23]를 사용하면 클라우드에서 관계형 데이터베이스를 손쉽게 설치, 운영 및 확장할 수 있습니다. 하드웨어 프로비저닝, 데이터베이스 설정, 패치 적용, 백업 등 시간이 많이 소요되는 관리 작업을 자동화하는 동시에 비용 효율적이고 크기 조정이 가능한 용량을 제공합니다. 덕분에 애플리케이션에 집중하면서 애플리케이션에 필요한 빠른 성능, 고가용성, 보안 및 호환성을 제공할 수 있습니다.
Amazon ElastiCache	Amazon ElastiCache는 완전관리형 Redis 및 Memcached를 제공합니다. 인기 있는 오픈 소스 호환 인메모리 데이터 스토어를 원활하게 배포, 실행 및 확장합니다. 높은 처리량과 짧은 지연 시간을 제공하는 인메모리 데이터 스토어에서 데이터를 검색함으로써 데이터 집약적인 앱을 구축하거나 기존 앱의 성능을 개선합니다.

데이터 스토어	기능
Amazon EBS	Amazon Elastic Block Store(EBS)는 Amazon Elastic Compute Cloud(EC2)와 함께 사용하여 규모에 관계없이 높은 처리량이 요구되고 트랜잭션 집약적인 워크로드를 지원할 수 있도록 설계된 간편한 고성능 블록 스토리지 서비스입니다. Amazon EBS 볼륨 데이터는 단일 구성 요소의 장애로 인한 데이터 손실을 방지하기 위해 가용 영역의 여러 서버에 복제됩니다. Amazon EBS 볼륨은 워크로드 실행에 필요한 지연 시간이 짧고 일관된 성능을 제공합니다. Amazon EBS를 사용하면 프로비저닝하는 항목에 대해서만 적은 요금을 지불하면서 몇 분 내에 사용량을 늘리거나 줄일 수 있습니다.
Amazon EFS	Amazon Elastic File System(Amazon EFS)은 AWS 클라우드 서비스 및 온프레미스 리소스에 사용할 수 있는 간편하고 확장 가능하며 탄력적인 Linux 기반 워크로드용 파일 시스템을 제공합니다. 애플리케이션을 중단하지 않고 필요에 따라 페타바이트까지 확장할 수 있도록 구축되었으며, 파일을 추가하고 제거할 때마다 자동 확장 및 축소되므로, 애플리케이션에서 스토리지가 필요할 때 필요한 만큼 확보할 수 있습니다. 또한 수천 개의 Amazon EC2 인스턴스에 대한 대량 병렬 공유 액세스를 제공하도록 설계되었습니다. 이를 통해 애플리케이션에서 일관되게 짧은 지연 시간으로 높은 수준의 총 처리량과 IOPS를 실현할 수 있습니다.
Amazon S3	Amazon Simple Storage Service(Amazon S3)는 업계 최고의 확장성, 데이터 가용성, 보안 및 성능을 제공하는 객체 스토리지 서비스입니다. 즉, 규모와 산업 분야에 관계없이 모든 고객이 웹 사이트, 모바일 애플리케이션, 백업 및 복원, 아카이브, 엔터프라이즈 애플리케이션, IoT 디바이스, 빅 데이터 분석 등 다양한 사용 사례에 맞춰 원하는 양의 데이터를 저장하고 보호하는 데 활용할 수 있습니다.

AWS에서의 지속적 통합 및 지속적 전달

지속적 통합(CI)과 지속적 전달(CD)은 모던 애플리케이션 개발의 가치를 실현하는 데 매우 중요한 요소이므로, AWS를 기반으로 이러한 모범 사례를 구현하는 방법을 신중하게 고려하는 것이 중요합니다. CI/CD로 배포 자동화 섹션에서 설명한 바와 같이 AWS는 모던 애플리케이션을 신속하게 제공하도록 지원하는 몇 가지 서비스를 제공합니다. 이러한 서비스는 완전관리형으로 제공되므로, 개발 팀이 CI 서버 유지 관리 및 보안과 관련한 획일적이고 과중한 작업이 아니라 배포를 자동화하고 새로운 기능을 신속하게 제공하는 데 집중할 수 있습니다.

AWS 기반 CI/CD 서비스

AWS 클라우드에서 CI/CD 배포를 위해 다음 AWS 서비스를 이용할 수 있습니다.

AWS Cloud9

AWS Cloud9[24]은 브라우저만으로 코드를 작성, 실행 및 디버깅하는 데 사용할 수 있는 클라우드 기반 IDE(통합 개발 환경)으로, 코드 편집기, 디버거 및 터미널이 포함되어 있습니다. AWS Cloud9에는 JavaScript, Python, PHP 등 많이 사용되는 프로그래밍 언어를 지원하는 필수 도구가 포함되어 있으므로 새 프로젝트를 시작하기 위해 파일을 설치하거나 개발 머신을 구성할 필요가 없습니다.

AWS Cloud9 IDE는 클라우드 기반이므로 사무실이든, 집이든, 인터넷에 연결된 시스템이 있는 곳이라면 어디서나 프로젝트 작업을 수행할 수 있습니다. 또한 AWS Cloud9은 서버리스 애플리케이션을 개발할 수 있는 원활한 환경을 제공하므로, 손쉽게 리소스를 정의하고, 디버깅하고, 서버리스 애플리케이션의 로컬 실행과 원격 실행 간을 전환할 수 있습니다. AWS Cloud9을 사용하면 개발 환경을 다른 팀원과 빠르게 공유할 수 있으므로 프로그램을 페어링하고 서로 입력한 내용을 실시간으로 추적할 수 있습니다.

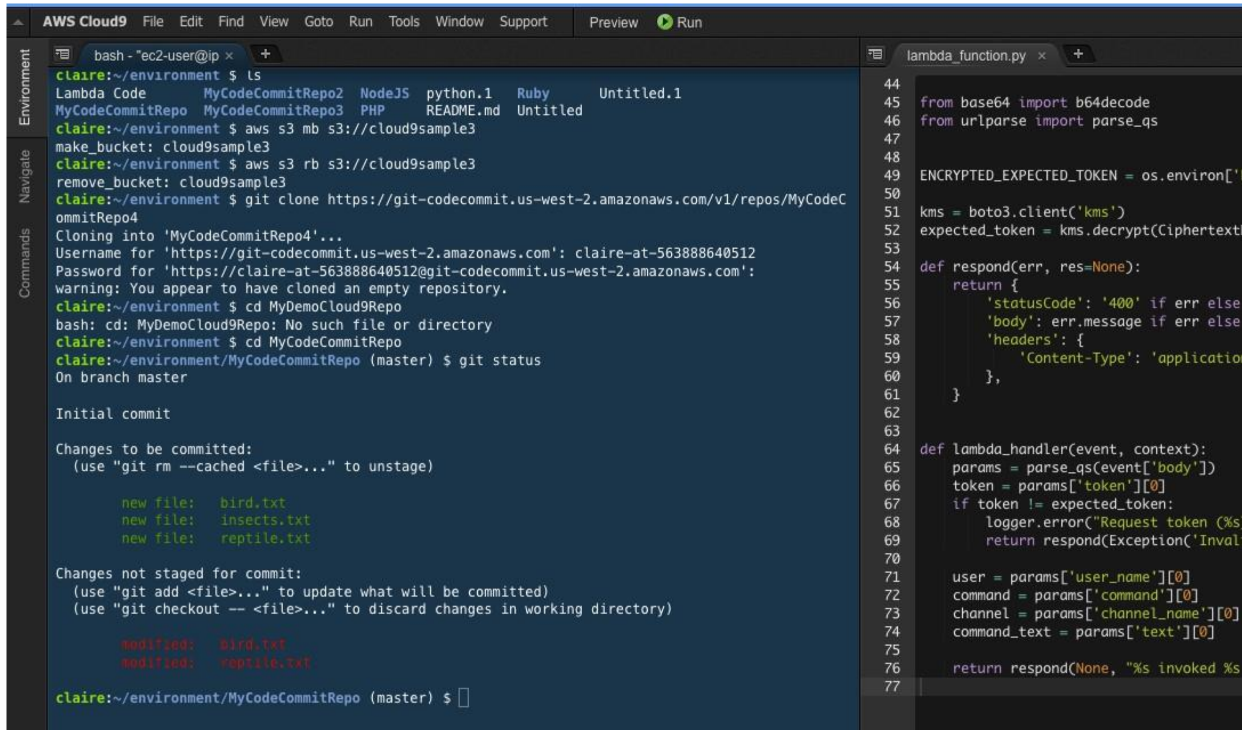

그림 18 – AWS Cloud9의 코드 예

AWS CodeStar

AWS CodeStar[25]를 사용하면 AWS 클라우드에서 애플리케이션을 신속하게 개발, 구축 및 배포할 수 있습니다. AWS CodeStar는 통합 사용자 인터페이스를 제공하므로 소프트웨어 개발 활동을 한 곳에서 손쉽게 관리할 수 있습니다. AWS CodeStar에서는 지속적 전달 도구 체인을 몇 분 만에 설정할 수 있으므로 코드 릴리스를 더 빠르게 시작할 수 있습니다. AWS CodeStar는 팀 전체가 안전하게 협업할 수 있게 해줍니다. 손쉽게 액세스를 관리하고 프로젝트에 소유자, 기여자 및 최종 사용자를 추가할 수 있습니다. 각 AWS CodeStar 프로젝트에는 작업 항목의 백로그부터 팀의 최근 코드 배포에 이르기까지 전체 소프트웨어 개발 프로세스의 진행 상황을 손쉽게 추적할 수 있는 프로젝트 관리 대시보드가 포함됩니다.

AWS CodePipeline

AWS CodePipeline[26]은 릴리스 파이프라인을 자동화하여 빠르고 안정적인 애플리케이션 및 인프라 업데이트를 지원하는 완전관리형 지속적 전달 서비스입니다. CodePipeline은

사용자가 정의한 릴리스 모델에 따라 코드가 변경될 때마다 릴리스 프로세스의 구축, 테스트 및 배포 단계를 자동화합니다. 따라서 기능과 업데이트를 안정적으로 신속하게 제공할 수 있습니다.

AWS CodeCommit

AWS CodeCommit[27]은 안전한 Git 기반 리포지토리를 호스팅하는 완전관리형 소스 제어 서비스로[28] 팀원들이 안전하고 확장성이 뛰어난 에코시스템에서 손쉽게 코드와 관련한 협업을 진행할 수 있습니다. CodeCommit을 사용하면 자체 소스 제어 시스템을 운영할 필요가 없으며 인프라 확장을 걱정하지 않아도 됩니다. CodeCommit을 사용하면 소스 코드에서 이진 코드에 이르기까지 어떤 코드이든 안전하게 저장할 수 있고 기존 Git 도구를 사용해 원활하게 작업할 수 있습니다.

AWS CodeBuild

AWS CodeBuild[29]는 소스 코드를 컴파일하고 테스트를 실행하며 배포 준비가 완료된 소프트웨어 패키지를 생성하는 완전관리형 지속적 통합 서비스입니다. CodeBuild를 사용하면 자체 빌드 서버를 프로비저닝, 관리 및 확장할 필요가 없습니다. CodeBuild는 지속적으로 확장되며 여러 빌드를 동시에 처리하기 때문에 빌드가 대기열에서 대기하지 않고 바로 처리됩니다. 사전 패키징된 빌드 환경을 사용하면 신속하게 시작할 수 있으며 혹은 자체 빌드 도구를 사용하는 사용자 지정 빌드 환경을 만들 수 있습니다.

AWS CodeDeploy

AWS CodeDeploy[30]는 Amazon Elastic Compute Cloud(Amazon EC2), AWS Fargate, AWS Lambda 및 온프레미스 서버와 같은 다양한 컴퓨팅 서비스로의 소프트웨어 배포를 자동화하는 완전관리형 배포 서비스입니다. AWS CodeDeploy를 사용하면 애플리케이션 배포 시에 새로운 기능을 신속하게 릴리스하고 가동 중지를 방지할 수 있습니다. 또한 AWS CodeDeploy는 애플리케이션 업데이트의 복잡성을 해소합니다. AWS CodeDeploy를 사용하여 소프트웨어 배포를 자동화하면 오류가 발생하기 쉬운 수작업을 없앨 수 있습니다. 이 서비스는 배포 요구 사항에 따라 확장됩니다.

AWS Amplify Console

AWS Amplify Console[31]은 전체 스택 서버리스 웹 애플리케이션을 배포하고 호스팅하는 Git 기반 워크플로를 제공합니다. 전체 스택 서버리스 애플리케이션은 클라우드 리소스(GraphQL[32] 또는 REST API)와 파일 및 데이터 스토리지로 구축된 백엔드와 단일 페이지 애플리케이션 프레임워크(React[33], Angular[34], Vue[35] 또는 Gatsby[36])로 구축된 프런트엔드로 구성됩니다. 전체 스택 서버리스 웹 애플리케이션 기능은 브라우저에서 실행되는 프런트엔드 코드와 클라우드에서 실행되는 백엔드 비즈니스 로직으로 분산되는 경우가 많습니다. 따라서 프런트엔드와 백엔드가 호환되고 새로운 기능이 프로덕션 고객에게 지장을 주지 않도록 릴리스 주기를 신중하게 조정해야 하므로 애플리케이션 배포가 복잡해지고 시간이 많이 소요됩니다. Amplify 콘솔은 전체 스택 서버리스 애플리케이션을 배포하는 간단한 워크플로를 제공하여 애플리케이션 릴리스 주기를 가속화합니다. 애플리케이션의 코드 리포지토리를 Amplify 콘솔에 연결하면 프런트엔드와 백엔드에 대한 변경 사항이 단일 워크플로의 모든 코드 커밋에서 배포됩니다.

다양한 애플리케이션 유형별 CI/CD 패턴

AWS 클라우드에 배포할 수 있는 주요 모던 애플리케이션 유형별로 CI/CD 패턴을 사용할 수 있습니다. 복잡한 CI 환경을 설정하고 유지 관리하는 번거로운 작업에 대한 부담 없이 AWS 네이티브 개발 도구를 사용하여 CI/CD를 신속하게 구현할 수 있습니다. 다음은 AWS 클라우드에서 CI/CD 패턴을 사용하는 방법을 보여주는 몇 가지 예입니다.

단일 페이지 애플리케이션 배포

단일 페이지 애플리케이션(SPA)은 브라우저에 다운로드되는 정적 콘텐츠(HTML, CSS, JavaScript 및 미디어)로 구성된 애플리케이션으로, 백엔드 API를 호출합니다. AWS Amplify Console을 사용하여 이러한 SPA를 신속하게 구축하고 릴리스할 수 있습니다. AWS Amplify Console은 새 코드가 GitHub[37] 또는 AWS CodeCommit 같은 리포지토리로 푸시될 때 이를 자동으로 감지하고, 정적 프런트엔드 콘텐츠를 Amazon Simple Storage Service(Amazon S3)에 배포한 다음, 콘텐츠 전송 네트워크인

Amazon CloudFront[38]를 통해 사용자에게 콘텐츠를 전송할 수 있습니다. Amplify 콘솔은 GraphQL 및 REST API, 인증, 분석 및 Amplify CLI에서 생성한 스토리지를 사용하여 서버리스 백엔드에 변경 사항을 배포할 수도 있습니다.

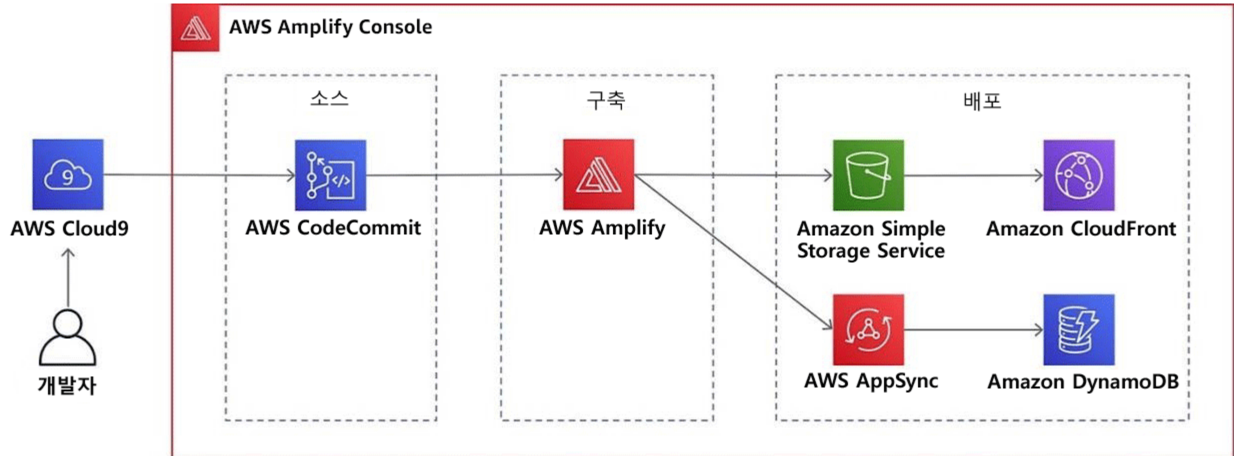

그림 19 – 단일 페이지 애플리케이션의 배포 아키텍처 예

컨테이너에 배포

AWS CodePipeline을 사용하면 최소한의 구성으로 Amazon Elastic Container Service(Amazon ECS) 컨테이너 오케스트레이션 서비스에 지속적으로 배포할 수 있습니다. 소스 단계에서 AWS CodePipeline은 소스 코드 리포지토리의 변경 사항을 자동으로 감지합니다. 빌드 단계에서는 AWS CodeBuild를 사용하여 Docker 이미지를 빌드하고 Amazon Elastic Container Registry(ECR)[39]와 같은 Docker 리포지토리에 푸시합니다. 마지막으로 AWS CodePipleline은 Amazon ECS에 배포합니다.

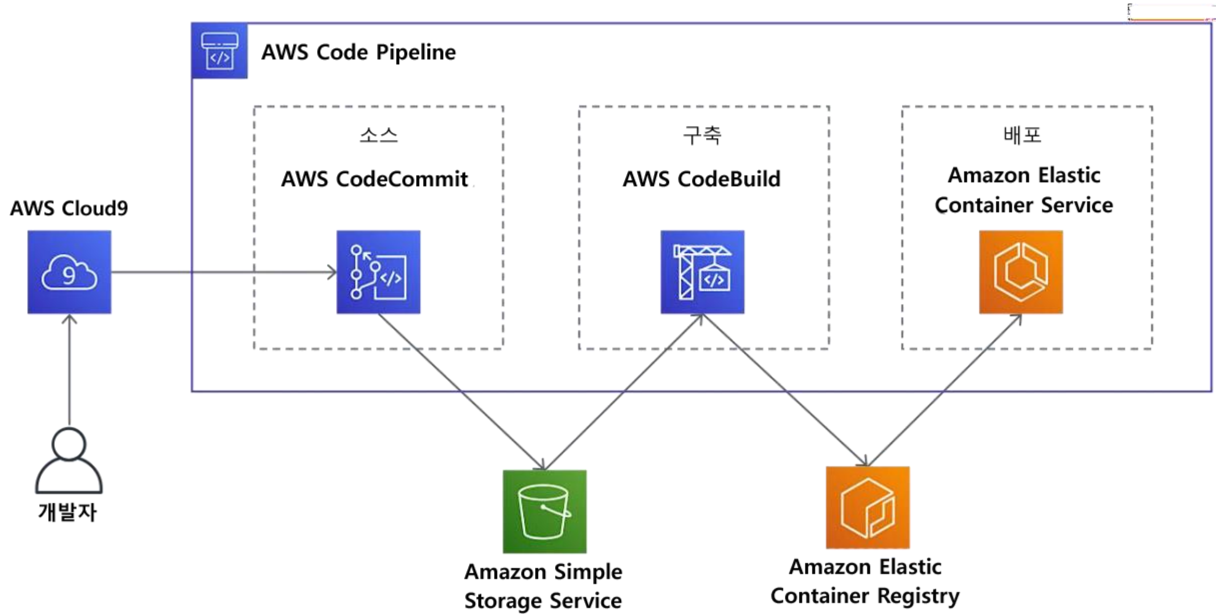

그림 20 – 컨테이너에 배포 아키텍처 예

컨테이너에 배포(블루/그린 배포)

Amazon ECS 및 AWS CodeDeploy는 컨테이너에 블루/그린 배포도 지원합니다.

AWS CodeDeploy는 Amazon Elastic Load Balancing[40]의 유형 중 하나인 Application Load Balancer(ALB)를 사용하여 2개의 병렬 대상 그룹 간에 트래픽을 원활하게 전환하면서 블루/그린 배포를 자동화합니다.

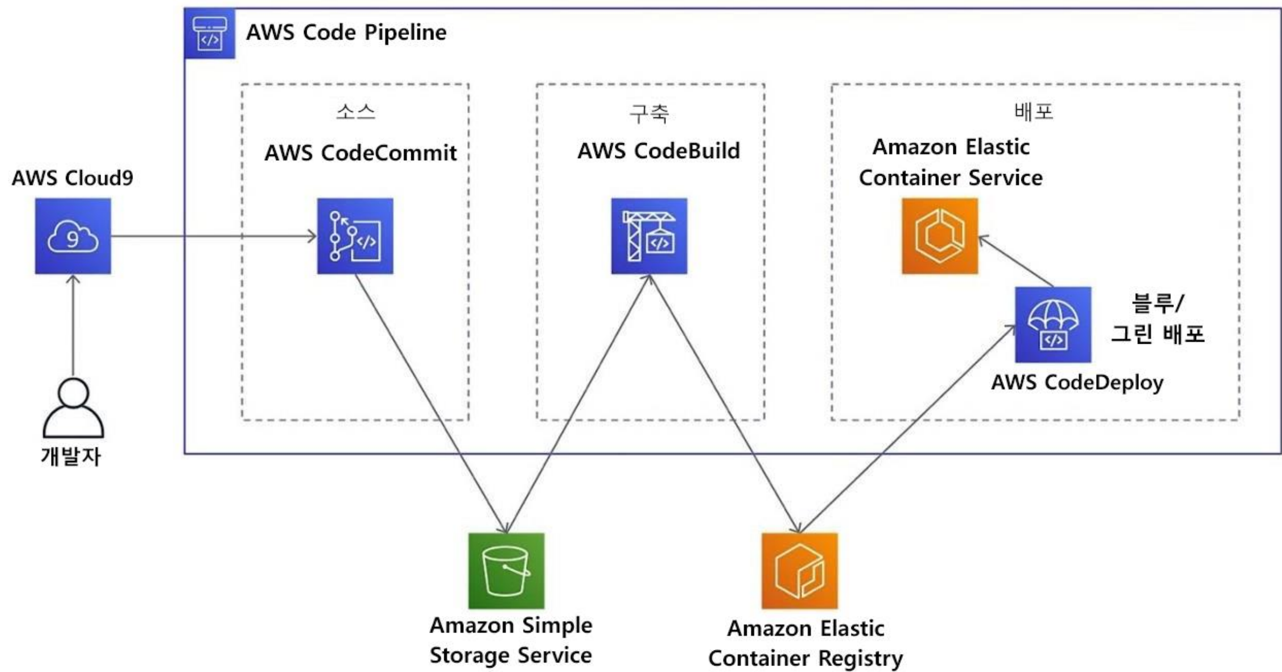

그림 21 – 컨테이너에 블루/그린 배포 아키텍처 예

AWS Lambda로의 Canary 배포

AWS CodeDeploy는 AWS Lambda로의 Canary 배포도 지원합니다. AWS CodeDeploy는 Lambda의 트래픽 전환 기능을 사용하여 새로운 함수 버전의 점진적 롤아웃을 자동화합니다. 따라서 두 버전 간에 트래픽을 점진적으로 전환할 수 있으며, 위험을 줄이고 새 Lambda 배포의 영향을 제한하는 데 도움이 됩니다.

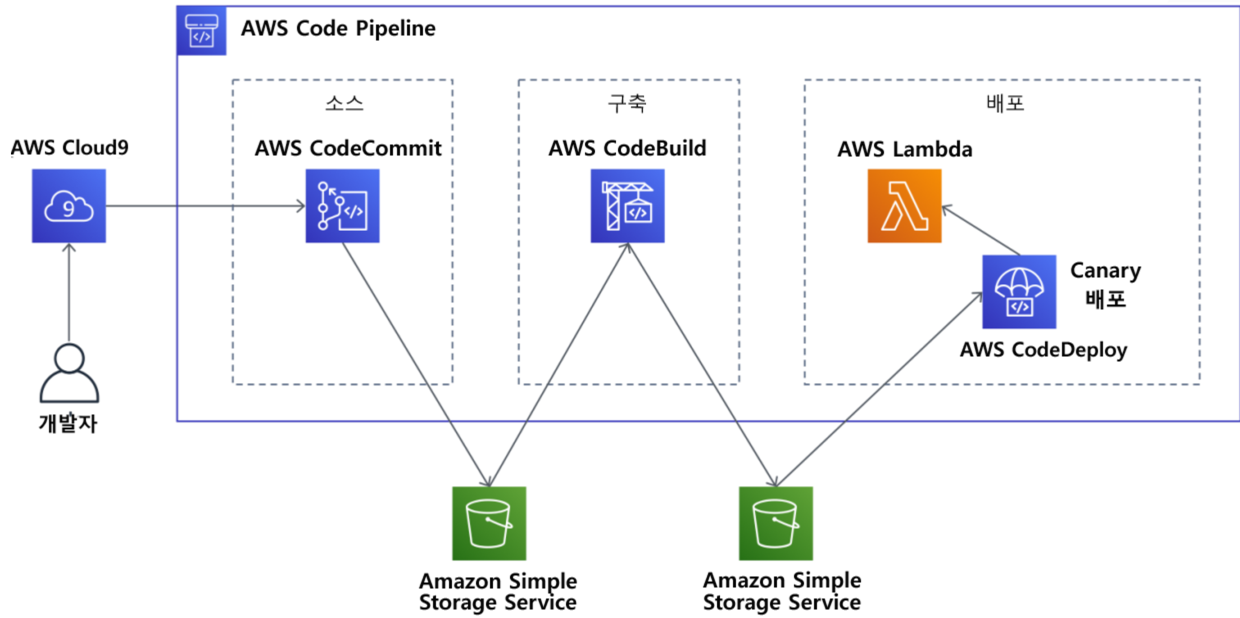

그림 22 – AWS 클라우드에서의 Canary 배포 아키텍처 예

AWS CodeDeploy를 사용하여 AWS Lambda 배포를 수행할 때는 다음과 같은 사전 정의된 배포 구성 옵션 중 하나를 사용하거나 자체적으로 맞춤형 구성을 생성할 수 있습니다. 이러한 모든 옵션은 서버리스 애플리케이션 모델(SAM)을 기반으로 애플리케이션을 배포하는 데에도 사용할 수 있습니다.

표 2 – AWS Lambda 및 AWS CodeDeploy를 사용한 Canary 배포용으로 사전 정의된 배포 구성 옵션

배포 구성	설명
CodeDeployDefault.LambdaCanary10Percent5Minutes	첫 번째 증분에서 트래픽의 10%를 전환합니다. 나머지 90%는 15분 후에 배포됩니다.
CodeDeployDefault.LambdaCanary10Percent10Minutes	첫 번째 증분에서 트래픽의 10%를 전환합니다. 나머지 90%는 10분 후에 배포됩니다.
CodeDeployDefault.LambdaCanary10Percent15Minutes	첫 번째 증분에서 트래픽의 10%를 전환합니다. 나머지 90%는 15분 후에 배포됩니다.
CodeDeployDefault.LambdaCanary10Percent30Minutes	첫 번째 증분에서 트래픽의 10%를 전환합니다. 나머지 90%는 30분 후에 배포됩니다.
CodeDeployDefault.LambdaLinear10PercentEvery1Minute	모든 트래픽이 전환될 때까지 트래픽의 10%를 1분마다 전환합니다.
CodeDeployDefault.LambdaLinear10PercentEvery2Minutes	모든 트래픽이 전환될 때까지 트래픽의 10%를 2분마다 전환합니다.
CodeDeployDefault.LambdaLinear10PercentEvery3Minutes	모든 트래픽이 전환될 때까지 트래픽의 10%를 3분마다 전환합니다.
CodeDeployDefault.LambdaLinear10PercentEvery10Minutes	모든 트래픽이 전환될 때까지 트래픽의 10%를 10분마다 전환합니다.
CodeDeployDefault.LambdaAllAtOnce	모든 트래픽을 업데이트된 Lambda 함수로 한 번에 전환합니다.

결론

현대 기업들은 치열한 경쟁에서 우위를 차지하기 위해 전 세계에 걸쳐 접근성을 확대하고 디지털 이니셔티브에 투자해야 합니다. 디지털 제품과의 사용자 상호 작용이 진화함에 따라, 더 향상된 고객 경험을 제공하고 갈수록 다양한 사용자 풀을 만족시켜야 합니다. 기업이 사용자의 높은 기대치를 충족하려면 실패를 두려워해서는 안 되며, 사용자 피드백을 지속적으로 실험하고 제품에 반영해야 합니다.

모던 애플리케이션 개발은 신속한 업데이트 및 릴리스를 지원하는 사고 방식이자 방법론입니다. 이러한 현대적 방식을 수용하는 개발 팀은 반복적인 작업을 자동화하고, 관리형 서비스를 최대한 활용하고, 고객을 만족시키는 제품을 만드는 데 많은 시간을 할애함으로써 획일적인 업무의 부담을 덜 수 있습니다.

모던 애플리케이션 개발 모범 사례를 성공적으로 도입하는 조직은 신속하게 실험하고 혁신할 수 있으며, 네이티브 AWS 서비스를 활용하여 이러한 방식을 구현하면 혁신을 더 빠르게 실현할 수 있습니다.

기고자

이 문서를 작성하는 데 도움을 주신 분들입니다.

- Atsushi Fukui, 솔루션스 아키텍트, Amazon Web Services
- Kevin Bell, 솔루션스 아키텍트, Amazon Web Services

추가 자료

자세한 내용은 다음 리소스를 참조하십시오.

AWS 서비스

- Amazon API Gateway
  https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
- AWS Cloud Map
  https://docs.aws.amazon.com/cloud-map/latest/dg/what-is-cloud-map.html
- AWS Lambda
  https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- Amazon Kinesis Data Streams
  https://docs.aws.amazon.com/streams/latest/dev/introduction.html
- Amazon Simple Queue Service
  https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
- Amazon Simple Notification Service
  https://docs.aws.amazon.com/sns/latest/dg/welcome.html
- Amazon Elastic Container Service
  https://docs.aws.amazon.com/AmazonECS/latest/userguide/Welcome.html
- Amazon EKS
  https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html
- AWS CodePipeline
  https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html
- AWS CodeCommit
  https://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html
- AWS CodeBuild
  https://docs.aws.amazon.com/codebuild/latest/userguide/welcome.html

- AWS CodeDeploy
  https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html
- AWS CodeDeploy에서 Amazon ECS로의 블루/그린 배포
  https://docs.aws.amazon.com/AmazonECS/latest/userguide/deployment-type-bluegreen.html
- AWS CodeStar
  https://docs.aws.amazon.com/codestar/latest/userguide/welcome.html
- AWS Cloud9
  https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html
- 메시징 서비스
  https://aws.amazon.com/messaging/
- 서버리스 서비스
  https://aws.amazon.com/serverless/

백서

- AWS 기반 마이크로서비스
  https://d1.awsstatic.com/whitepapers/microservices-on-aws.pdf
- AWS 기반 DevOps 소개
  https://d1.awsstatic.com/whitepapers/AWS_DevOps.pdf
- 코드형 인프라
  https://d1.awsstatic.com/whitepapers/infrastructure-as-code.pdf
- AWS에서 지속적 통합 및 지속적 전달 적용
  https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration- continuous-delivery-on-AWS.pdf

동영상

Choosing the Right Messaging Service for Your Distributed App (API305)
https://www.youtube.com/watch?v=4-JmX6MIDDI

문서 개정

날짜	설명
2019년 10월	최초 게시

참고

1. Amazon EC2 – https://aws.amazon.com/ec2/
2. Microservices – https://martinfowler.com/articles/microservices.html
3. AWS Lambda – https://aws.amazon.com/lambda/
4. AWS Fargate – https://aws.amazon.com/fargate/
5. Amazon S3 – https://aws.amazon.com/s3/
6. Amazon DynamoDB – https://aws.amazon.com/dynamodb/
7. Amazon Aurora 서버리스 – https://aws.amazon.com/rds/aurora/serverless/
8. 서버리스 – https://aws.amazon.com/serverless/
9. AWS CloudFormation – https://aws.amazon.com/cloudformation/
10. AWS Serverless Application Model –https://aws.amazon.com/serverless/sam/
11. AWS CDK – https://docs.aws.amazon.com/cdk/latest/guide/what-is.html
12. Amazon API Gateway – https://aws.amazon.com/api-gateway/
13. Envoy Proxy – https://www.envoyproxy.io/
14. Amazon Kinesis – https://aws.amazon.com/kinesis/
15. Amazon Simple Queue Service – https://aws.amazon.com/sqs/
16. Amazon MQ – https://aws.amazon.com/amazon-mq/
17. Amazon MSK – https://aws.amazon.com/msk/
18. Amazon CloudWatch – https://aws.amazon.com/cloudwatch/
19. Fluentd – https://www.fluentd.org/
20. Fluent Bit – https://fluentbit.io/
21. Container Insights 사용 – https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContainerInsights.html
22. AWS X-Ray – https://aws.amazon.com/xray/
23. Amazon RDS – https://aws.amazon.com/rds/
24. Amazon RDS – https://aws.amazon.com/rds/
25. AWS CodeStar – https://aws.amazon.com/codestar/

26. AWS CodePipeline – https://aws.amazon.com/codepipeline/
27. AWS CodeCommit https://aws.amazon.com/codecommit/
28. Git – https://git-scm.com/
29. AWS CodeBuild – https://aws.amazon.com/codebuild/
30. AWS CodeDeploy – https://aws.amazon.com/codedeploy/
31. AWS Amplify Console – https://aws.amazon.com/amplify/console/
32. GraphQL – https://graphql.org/
33. React – https://reactjs.org/
34. Angular – https://angular.io/
35. Vue – https://vuejs.org/index.html
36. Gatsby – https://www.gatsbyjs.org/
37. GitHub – https://github.com/
38. Amazon CloudFront – https://aws.amazon.com/cloudfront/
39. Amazon Elastic Container Registry – https://aws.amazon.com/ecr/
40. Amazon Elastic Load Balancing – https://aws.amazon.com/elasticloadbalancing/