AWS 백서

AWS에서 지속적인 통합 및 지속적인 전송 적용

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS에서 지속적인 통합 및 지속적인 전송 적용: AWS 백서

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계 여부에 관계없이 해당 소유자의 자산입니다.

Table of Contents

요약 1
- 요약 1

소프트웨어 전송의 당면 과제 2
지속적 통합 및 지속적 전달/배포란 무엇입니까? 3
- 지속적 통합 3
- 지속적 전달 및 배포 3
- 지속적 배포가 아닌 지속적 전달 4

지속적인 전송의 이점 5
- 소프트웨어 릴리스 프로세스의 자동화 5
- 개발자 생산성 향상 5
- 코드 품질 개선 5
- 업데이트를 더 빠르게 제공 5

지속적 통합 및 지속적 전달 구현 6
- 지속적 통합/지속적 전달을 위한 경로 6
  - 지속적 통합 7
  - 지속적 전달: 스테이징 환경 만들기 8
  - 지속적 전달: 프로덕션 환경 생성 8
  - 지속적 배포 9
  - 기대를 뛰어넘는 완성도 9
- 팀 9
  - 애플리케이션 팀 10
  - 인프라 팀 10
  - 도구 팀 11
- 지속적 통합 및 지속적 전달의 테스트 단계 11
  - 소스 설정 12
  - 빌드 설정 및 실행 13
  - 빌드 13
  - 스테이징 13
  - 프로덕션 14
- 파이프라인 구축 14
  - 지속적 통합을 위한 최소한의 실행 가능한 파이프라인부터 시작 14
  - 지속적 전달 파이프라인 20
  - Lambda 작업 추가 21
  - 수동 승인 21

CI/CD 파이프라인에 인프라 코드 변경 배포 22
서버리스 애플리케이션용 CI/CD 22
여러 팀, 분기 및 AWS 리전을 위한 파이프라인 23
AWS CodeBuild과(와) 파이프라인 통합 23
Jenkins와 파이프라인 통합 24
배포 방법 26
한 번에 모두(바로 배포) 27
롤링 배포 27
변경 불가능 및 블루/그린 배포 28
데이터베이스 스키마 변경 사항 29
모범 사례 요약 30
결론 32
추가 자료 33
기여자 34
문서 개정 35
고지 사항 36

AWS에서 지속적인 통합 및 지속적인 전송 적용

게시 날짜: 2021년 10월 27일(문서 개정)

요약

이 백서에서는 소프트웨어 개발 환경에서 지속적 통합/지속적 전송(CI/CD) 및 Amazon Web Services(AWS) 도구를 사용할 때의 이점과 기능을 설명합니다. 지속적인 통합과 지속적인 전송은 모범 사례이며 DevOps 이니셔티브의 중요한 부분입니다.

소프트웨어 전송의 당면 과제

오늘날 기업은 급변하는 경쟁 환경, 진화하는 보안 요구 사항 및 성능 확장성으로 인한 과제에 직면해 있습니다. 기업은 운영 안정성과 신속한 기능 개발 간의 격차를 해소해야 합니다. 지속적인 통합 및 지속적인 전송(CI/CD)은 시스템 안정성과 보안을 유지하면서 신속한 소프트웨어 변경을 가능하게 하는 관행입니다.

Amazon은 Amazon.com 소매 고객, Amazon 자회사 및 Amazon Web Services(AWS)를 위한 기능을 제공해야 하는 비즈니스 요구 사항에는 새롭고 혁신적인 소프트웨어 제공 방법이 필요하다는 사실을 일찍이 깨달았습니다. Amazon과 같은 규모의 회사에서는 수천 개의 독립 소프트웨어 팀이 병렬로 작업하여 소프트웨어를 빠르고 안전하며 안정적으로 제공할 수 있어야 하며 가동 중단을 전혀 허용하지 않아야 합니다.

Amazon 및 기타 진보적인 조직들은 소프트웨어를 신속하게 제공하는 방법을 학습하게 되면서 DevOps를 선보였습니다. DevOps는 문화적 철학, 원칙 및 도구의 조합으로, 애플리케이션과 서비스를 더 빠르게 제공할 수 있도록 조직 역량을 향상합니다. 조직은 DevOps 원칙을 사용하여 기존 소프트웨어 개발 및 인프라 관리 프로세스를 사용하는 조직보다 더 빠른 속도로 제품을 개발하고 개선할 수 있습니다. 이러한 빠른 속도를 통해 조직은 고객을 더 잘 지원하고 시장에서 좀 더 효과적으로 경쟁할 수 있습니다.

피자 두 판 규모의 팀 및 마이크로서비스/서비스 지향 아키텍처(SOA) 와 같은 일부 원칙은 이 백서의 범위를 벗어납니다. 이 백서에서는 Amazon에서 구축하여 지속적으로 개선한 CI/CD 기능에 대해 설명합니다. CI/CD는 소프트웨어 기능을 빠르고 안정적으로 제공하는 데 핵심적인 역할을 합니다.

AWS은(는) 이제 AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy 및AWS CodeArtifact 등과 같은 CI/CD 기능을 개발자 서비스 집합으로 제공합니다. DevOps를 실행하는 개발자와 IT 운영 전문가는 이러한 서비스를 사용하여 소프트웨어를 빠르고 안전하게 제공할 수 있습니다. 함께 사용하면 애플리케이션의 소스 코드에 버전 제어를 안전하게 저장하고 적용할 수 있습니다. AWS CodeStar를 사용하면 이 서비스로 종단 간 소프트웨어 릴리스 워크플로를 신속하게 조정할 수 있습니다. 기존 환경의 경우, AWS CodePipeline은(는) 각 서비스를 기존 도구와 독립적으로 통합할 수 있는 유연성을 갖추었습니다. 이러한 서비스는 다른 AWS 서비스처럼 AWS Management Console, AWS API(애플리케이션 프로그래밍 인터페이스) 및 AWS SDK(소프트웨어 개발 도구 키트)를 통해 액세스할 수 있는 고가용성 통합 서비스입니다.

지속적 통합 및 지속적 전달/배포란 무엇입니까?

이 단원에서는 지속적 통합 및 지속적 전달의 관행에 대해 알아보고 지속적 전달과 지속적 배포의 차이점에 대해 설명합니다.

지속적 통합

지속적 통합(CI)은 개발자가 코드 변경 사항을 정기적으로 중앙 리포지토리에 병합한 후 자동 구축 및 테스트를 실행하는 소프트웨어 개발 방식입니다. CI는 대부분 소프트웨어 릴리스 프로세스의 빌드 또는 통합 단계를 의미하며 자동화 구성 요소(예: CI 또는 빌드 서비스)와 문화적 요소(예: 작은 통합)가 필요합니다. CI의 핵심 목표는 버그를 신속하게 찾아 해결하고, 소프트웨어 품질을 향상하고, 새로운 소프트웨어 업데이트를 검증 및 릴리스하는 데 걸리는 시간을 단축하는 것입니다.

지속적인 통합은 더 작은 커밋과 통합할 더 작은 코드 변경에 중점을 둡니다. 개발자는 최소 하루에 한 번 정기적으로 코드를 커밋합니다. 개발자는 코드 리포지토리에서 코드를 가져와 빌드 서버로 푸시하기 전에 로컬 호스트의 코드가 병합되었는지 확인합니다. 이 단계에서 빌드 서버는 다양한 테스트를 실행하고 코드 커밋을 승인하거나 거부합니다.

CI 구현의 기본 과제로는 공통 코드베이스에 대한 커밋 빈도 증가, 단일 소스 코드 리포지토리 유지 관리, 빌드 자동화 및 테스트 자동화 등이 있습니다. 추가 과제로는 프로덕션과 유사한 환경에서 테스트하고, 팀에 프로세스 가시성을 제공하고, 개발자가 애플리케이션의 모든 버전을 쉽게 얻을 수 있도록 하는 것입니다.

지속적 전달 및 배포

지속적 전달(CD)은 코드 변경 사항이 자동으로 빌드, 테스트 및 프로덕션 릴리스에 맞게 준비되는 소프트웨어 개발 방식입니다. 빌드 단계가 완료된 후 모든 코드 변경 사항을 테스트 환경, 프로덕션 환경 또는 둘 다에 배치함으로써 지속적인 통합으로 확장됩니다. 지속적 전달은 워크플로 프로세스를 통해 완전히 자동화되거나 중요한 시점에서 수동 단계를 통해 부분적으로 자동화될 수 있습니다. 지속적 전달이 적절하게 구현되면 개발자는 언제나 즉시 배포할 수 있고 표준화된 테스트 프로세스를 통과한 빌드 아티팩트를 보유하게 됩니다.

지속적인 배포를 통해 개발자의 명시적인 승인 없이 수정 버전이 프로덕션 환경에 자동으로 배치되어 전체 소프트웨어 릴리스 프로세스가 자동화됩니다. 이를 통해 제품 수명 주기 초기에 지속적인 고객 피드백 루프가 가능합니다.

지속적 배포가 아닌 지속적 전달

지속적 전달에 대한 오해 중 하나는 커밋된 모든 변경 사항이 자동화된 테스트를 통과한 직후 프로덕션에 적용된다는 것입니다. 그러나 지속적인 전달의 요점은 모든 변경 사항을 프로덕션에 즉시 적용하는 것이 아니라 모든 변경 사항을 프로덕션에 적용할 준비가 되었는지 확인하는 것입니다.

프로덕션에 변경 사항을 배포하기 전에 결정 프로세스를 구현하여 프로덕션 배포가 승인되고 감사되는지 확인할 수 있습니다. 사용자가 이러한 결정은 내린 다음 도구로 실행할 수 있습니다.

지속적 전달을 통해 준비된 결정은 기술적 결정이 아닌 비즈니스 결정이 됩니다. 기술 검증은 모든 커밋에서 이루어집니다.

프로덕션에 변경 사항을 도입하는 것은 지장을 주지 않습니다. 배포 시 기술 팀은 다음 변경 작업을 중단할 필요가 없으며 프로젝트 계획, 인계 문서 또는 유지 관리 기간이 필요하지 않습니다. 배포는 테스트 환경에서 여러 번 수행되고 검증된 반복 가능한 프로세스가 됩니다.

지속적인 전송의 이점

CD는 프로세스 자동화, 개발자 생산성 향상, 코드 품질 개선, 고객에게 신속하게 업데이트 제공 등 소프트웨어 개발 팀에 다양한 이점을 제공합니다.

소프트웨어 릴리스 프로세스의 자동화

CD는 소프트웨어 전송의 효율성, 복원력, 신속성 및 보안을 위해 자동으로 빌드, 테스트 및 릴리스 준비 코드를 프로덕션으로 체크인할 수 있는 방법을 제공합니다.

개발자 생산성 향상

CD 사례를 통해 개발자가 수동 작업에서 벗어나 복잡한 종속성을 벗어나고 소프트웨어의 새 기능을 제공하는 데 집중하여 팀의 생산성을 향상할 수 있습니다. 개발자는 코드를 비즈니스의 다른 부분과 통합하고 이 코드를 플랫폼에 배포하는 방법에 시간을 할애하는 대신 필요한 기능을 제공하는 코딩 로직에 집중할 수 있습니다.

코드 품질 개선

CD를 통해 버그가 나중에 더 큰 문제로 발생하기 전 전송 프로세스 초기에 버그를 발견하고 해결할 수 있습니다. 전체 프로세스가 자동화되어 있으므로, 팀에서 다른 유형의 코드 테스트를 쉽게 수행할 수 있습니다. 더 자주 테스트하는 규율을 통해 팀은 변경이 미치는 영향에 대한 즉각적인 피드백으로 더 빠르게 반복할 수 있습니다. 이를 통해 팀은 안정성과 보안에 대한 높은 보장으로 고품질 코드를 실행할 수 있습니다. 개발자는 즉각적인 피드백을 통해 새 코드의 작동 여부와 주요 변경 사항이나 버그가 도입되었는지 여부를 알 수 있습니다. 개발 프로세스 초기에 발견된 실수는 가장 쉽게 수정할 수 있습니다.

업데이트를 더 빠르게 제공

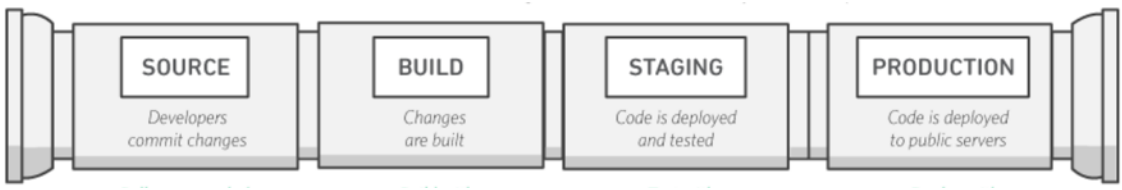
CD를 사용하면 팀에서 고객에게 자주 신속하게 업데이트를 제공할 수 있습니다. CI/CD가 구현되면 기능 릴리스 및 버그 수정을 포함하여 전체 팀의 속도가 빨라집니다. 기업은 시장 변화, 보안 문제, 고객 요구 및 비용 압박에 보다 신속하게 대응할 수 있습니다. 예를 들어, 새로운 보안 기능이 필요한 경우 팀에서 자동화된 테스트를 통해 CI/CD를 구현하여 높은 확신을 가지고 프로덕션 시스템에 빠르고 안정적으로 수정 사항을 도입할 수 있습니다. 몇 주, 몇 개월이 걸렸던 작업을 이제 며칠 또는 몇 시간 만에 완료할 수 있습니다.

지속적 통합 및 지속적 전달 구현

이 단원에서는 조직에서 CI/CD 모델을 구현할 수 있는 방법에 대해 설명합니다. 이 백서에서는 완성된 DevOps 및 클라우드 전환 모델을 갖춘 조직이 CI/CD 파이프라인을 구축하고 사용하는 방법에 대해서는 다루지 않습니다. DevOps 여정을 돕고자 AWS에는 리소스와 도구를 제공할 수 있는 다수의 DevOps 인증 파트너가 있습니다. AWS 클라우드로의 전환을 준비하는 방법에 대한 자세한 내용은 클라우드 운영 모델 구축을 참조하십시오.

지속적 통합/지속적 전달을 위한 경로

CI/CD는 파이프라인(다음 그림 참조) 으로 나타낼 수 있으며, 여기서 새 코드는 한쪽 끝에서 제출되고 일련의 단계(소스, 빌드, 스테이징 및 프로덕션)를 걸쳐 테스트된 다음 프로덕션 준비 코드로 게시됩니다. CI/CD를 처음 사용하는 조직의 경우 반복적인 방식으로 이 파이프라인에 접근할 수 있습니다. 즉, 소규모로 시작하여 각 단계에서 반복해야 조직의 성장에 도움이 되는 방식으로 코드를 이해하고 개발할 수 있습니다.



CI/CD 파이프라인

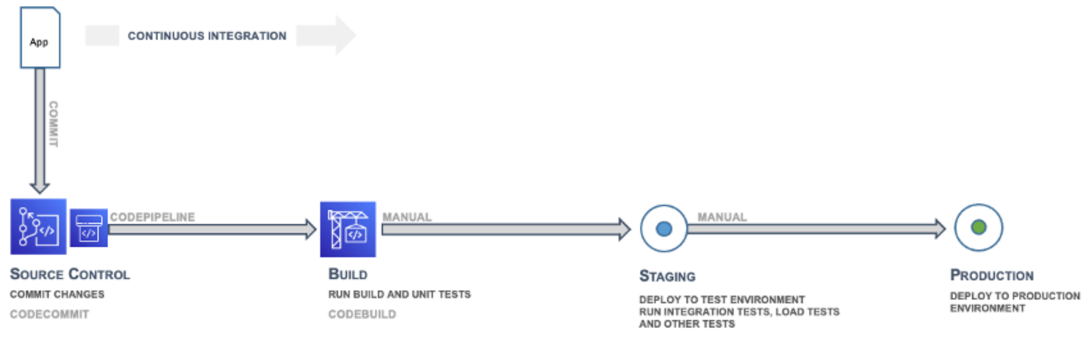
CI/CD 파이프라인의 각 단계는 전달 프로세스에서 논리적 단위로 구성됩니다. 또한 각 단계는 코드의 특정 측면을 확인하는 게이트 역할을 합니다. 파이프라인을 통해 코드가 진행됨에 따라 코드의 더 많은 측면이 계속 검증되기 때문에 이후 단계에서 코드 품질이 더 높다고 가정됩니다. 초기 단계에서 발견된 문제로 인해 코드가 파이프 라인을 통해 진행되지 않습니다. 테스트 결과는 즉시 팀으로 전송되며, 소프트웨어가 단계를 통과하지 못하면 이후의 모든 빌드 및 릴리스가 중지됩니다.

이 단계들은 제안 사항입니다. 비즈니스 요구 사항에 따라 단계를 조정할 수 있습니다. 여러 유형의 테스트, 보안 및 성능에 대해 일부 단계를 반복할 수 있습니다. 프로젝트의 복잡성과 팀 구조에 따라 일부 단계를 다른 수준에서 여러 번 반복할 수 있습니다. 예를 들어 한 팀의 최종 제품이 다음 팀의 프로젝트에 종속될 수 있습니다. 즉, 첫 번째 팀의 최종 제품은 이후 다음 팀의 프로젝트에서 아티팩트로 준비됩니다.

CI/CD 파이프라인이 있으면 조직의 역량 향상에 큰 영향을 미칩니다. 조직은 처음부터 여러 환경, 많은 테스트 단계 및 모든 단계의 자동화를 통해 완전한 파이프라인을 구축하려 하지 말고 작은 단계부터 시작해야 합니다. CI/CD 환경의 완성도가 높은 조직도 파이프라인을 지속적으로 개선해야 합니다.

CI/CD를 지원하는 조직을 구축하는 것은 하나의 여정이며, 그 과정에는 여러 목표가 있습니다. 다음 단원에서는 지속적 전달 수준을 통한 지속적 통합부터 시작하여 조직에서 취할 수 있는 가능한 경로에 대해 설명합니다.

지속적 통합



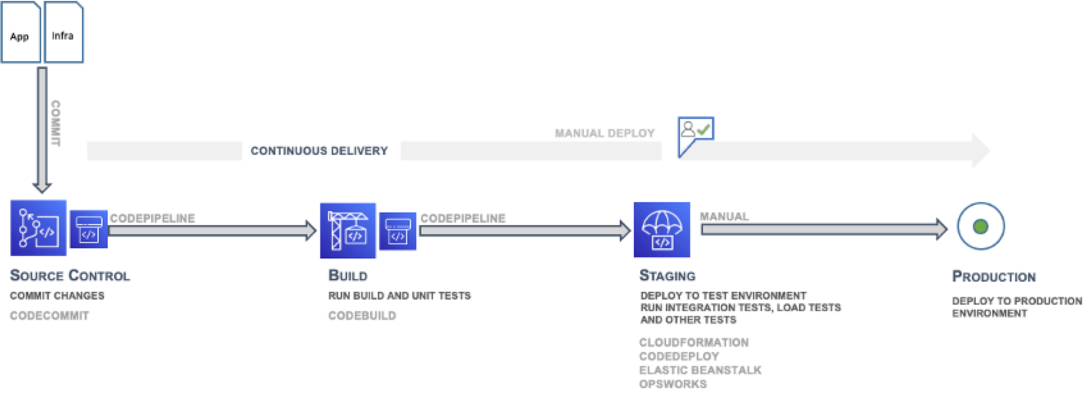
지속적 통합 - 소스 및 빌드

CI/CD 여정의 첫 번째 단계는 지속적 통합의 완성도를 개발하는 것입니다. 모든 개발자가 정기적으로 코드를 중앙 리포지토리(예: CodeCommit 또는 GitHub에서 호스팅되는 항목)에 커밋하고 모든 변경 사항을 애플리케이션의 릴리스 분기에 병합해야 합니다. 어떤 개발자도 코드를 따로 보관해서는 안 됩니다. 일정 기간 동안 기능 분기가 필요한 경우, 가능한 한 자주 업스트림에서 병합하여 최신 상태로 유지해야 합니다. 팀이 규율을 개발하고 그 과정을 통해 더욱 힘을 얻을 수 있도록 전체 작업 단위와 자주 커밋하고 병합하는 것이 좋습니다. 개발자가 코드를 초기에 자주 병합하면 향후 통합 문제가 줄어들 가능성이 높습니다.

또한 개발자가 애플리케이션에 대한 단위 테스트를 최대한 일찍 만들고 코드를 중앙 리포지토리에 푸시하기 전에 이러한 테스트를 실행하도록 권장해야 합니다. 소프트웨어 개발 프로세스 초기에 발견된 오류가 가장 저렴하고 쉽게 수정할 수 있습니다.

코드가 소스 코드 리포지토리의 분기로 푸시되면 해당 분기를 모니터링하는 워크플로 엔진이 빌더 도구로 명령을 보내 코드를 빌드하고 제어된 환경에서 단위 테스트를 실행합니다. 빠른 피드백을 위해 커밋 단계에서 발생할 수 있는 푸시 및 테스트를 포함한 모든 활동을 처리할 수 있도록 빌드 프로세스의 크기를 적절하게 조정해야 합니다. 단위 테스트 적용 범위, 스타일 체크 및 정적 분석과 같은 다른 품질 검사도 이 단계에서 수행될 수 있습니다. 마지막으로, 빌더 도구는 하나 이상의 바이너리 빌드와 기타 아티팩트(예: 이미지, 스타일시트 및 애플리케이션 문서)을 생성합니다.

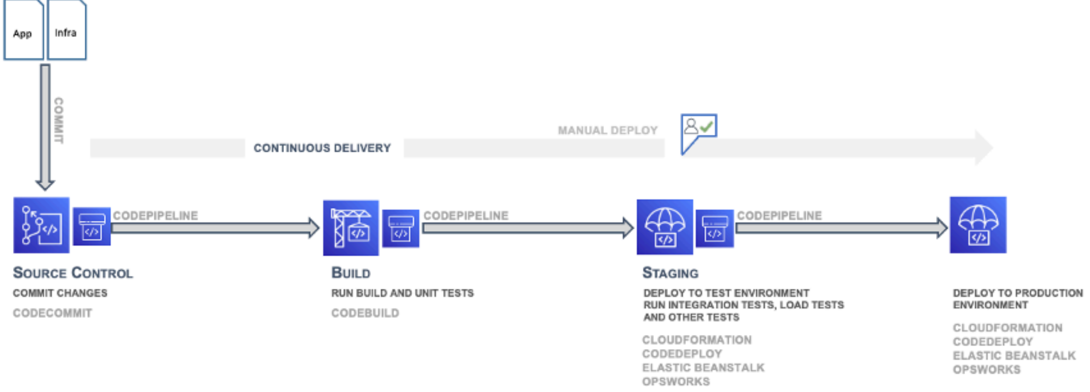
지속적 전달: 스테이징 환경 만들기



지속적 전달 - 스테이징

지속적 전달(CD)은 다음 단계이며, 프로덕션 스택의 복제본인 스테이징 환경에 애플리케이션 코드를 배포하고 더 많은 기능 테스트를 실행해야 합니다. 스테이징 환경은 테스트용으로 미리 만들어진 정적 환경이거나 애플리케이션 코드를 테스트하고 배포하기 위해 커밋된 인프라 및 구성 코드로 동적 환경을 프로비저닝하고 구성할 수 있습니다.

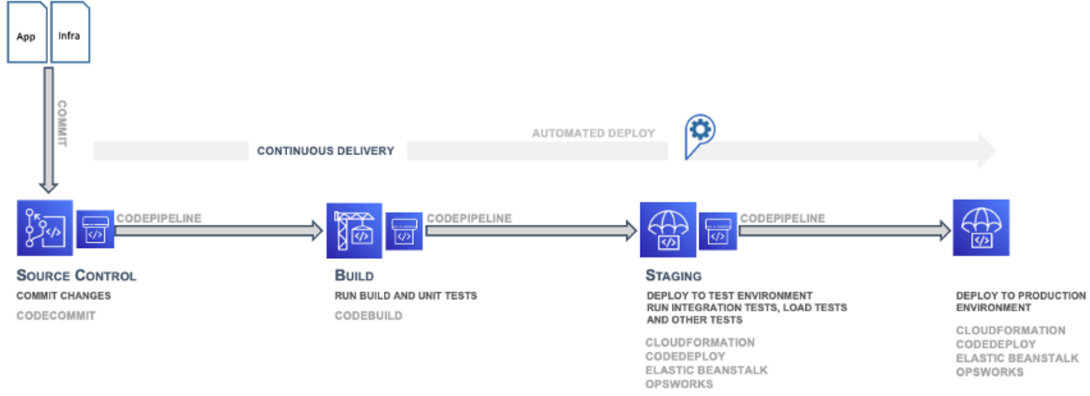
지속적 전달: 프로덕션 환경 생성



지속적 전달 - 프로덕션

배포/제공 파이프라인 순서에서 스테이징 환경 이후의 프로덕션 환경은 코드형 인프라(IaC)를 사용하여 구축됩니다.

지속적 배포



지속적 배포

CI/CD 배포 파이프라인의 마지막 단계는 지속적 배포이며, 여기에는 프로덕션 환경에 대한 배포를 포함하여 전체 소프트웨어 릴리스 프로세스의 전체 자동화가 포함될 수 있습니다. 완성된 CI/CD 환경에서는 프로덕션 환경으로 가는 경로가 완전히 자동화되어 코드를 매우 안정적으로 배포할 수 있습니다.

기대를 뛰어넘는 완성도

조직이 성장함에 따라 다음과 같은 개선 사항을 더 많이 포함하도록 CI/CD 모델을 계속 개발할 것입니다.

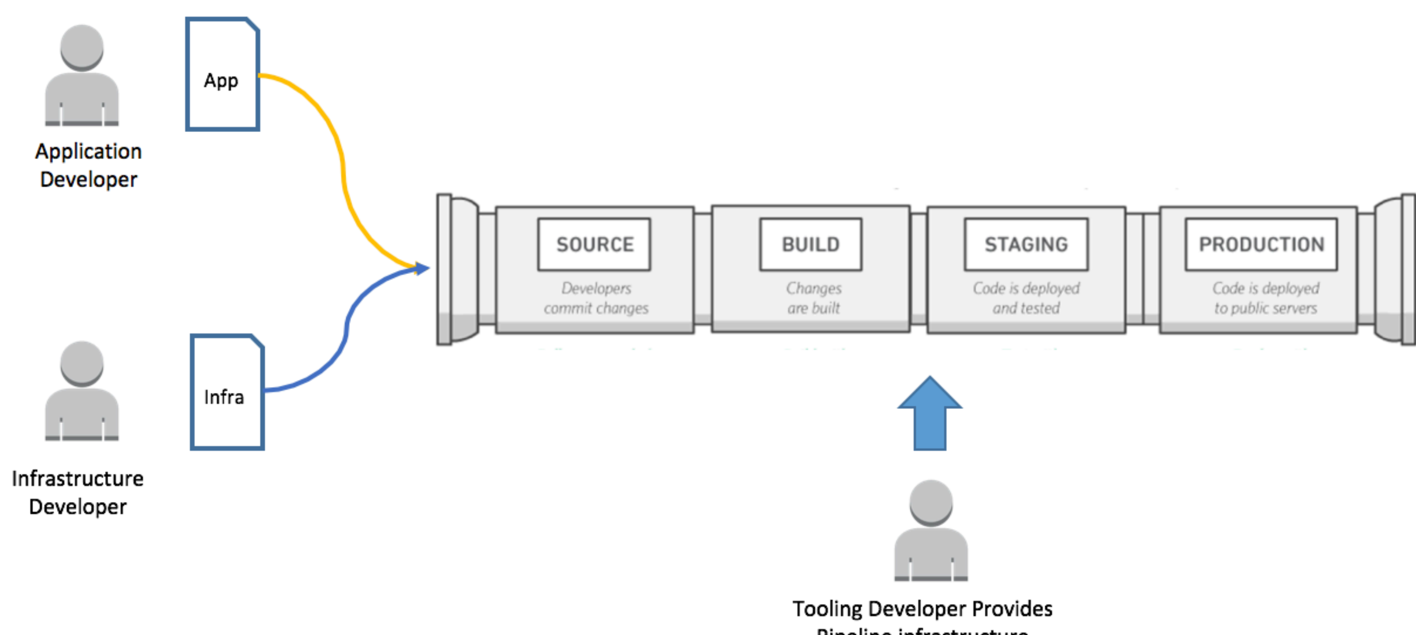
- 특정 성능, 규정 준수, 보안 및 UI(사용자 인터페이스) 테스트를 위한 더 많은 스테이징 환경
- 애플리케이션 코드와 함께 인프라 및 구성 코드의 단위 테스트
- 코드 검토, 문제 추적 및 이벤트 알림과 같은 다른 시스템 및 프로세스와의 통합
- 데이터베이스 스키마 마이그레이션과 통합(해당되는 경우)
- 감사 및 비즈니스 승인을 위한 추가 단계

복잡한 다중 환경 CI/CD 파이프라인을 갖춘 완성도가 높은 조직도 계속해서 개선을 모색하고 있습니다. DevOps는 목적지가 아닌 여정입니다. 파이프라인에 대한 피드백을 지속적으로 수집하고 개발 팀의 여러 부서 간의 협업을 통해 속도, 규모, 보안 및 안정성을 개선합니다.

팀

AWS에서는 CI/CD 환경을 구현하기 위해 애플리케이션 팀, 인프라 팀, 도구 팀 등 세 개의 개발자 팀을 구성할 것을 권장합니다(다음 그림 참조). 이 조직은 빠르게 변화하는 스타트업, 대기업 및 Amazon 자

체에서 개발되고 적용된 일련의 모범 사례를 나타냅니다. 팀은 피자 두 판 규모의 그룹 또는 약 10~12 명보다 크지 않아야 합니다. 이는 그룹 규모가 증가하고 소통이 늘어남에 따라 의미있는 대화가 한계에 도달한다는 통신 규칙을 따릅니다.



애플리케이션, 인프라 및 도구 팀

애플리케이션 팀

애플리케이션 팀에서 애플리케이션을 생성합니다. 애플리케이션 개발자는 백로그, 스토리 및 단위 테스트를 소유하며 지정된 애플리케이션 대상을 기반으로 기능을 개발합니다. 이 팀의 조직 목표는 개발자들이 비핵심 애플리케이션 작업에 소비하는 시간을 최소화하는 것입니다.

애플리케이션 팀은 애플리케이션 언어로 함수 프로그래밍 기술을 보유하는 것 외에도 플랫폼 기술과 시스템 구성에 대한 이해도 갖추어야 합니다. 이를 통해 기능 개발 및 애플리케이션 강화에만 집중할 수 있습니다.

인프라 팀

인프라 팀은 애플리케이션을 실행하는 데 필요한 인프라를 만들고 구성하는 코드를 작성합니다. 이 팀은 AWS CloudFormation과(와) 같은 기본 AWS 도구나 Chef, Puppet 또는 Ansible과 같은 일반 도구를 사용할 수 있습니다. 인프라 팀은 필요한 리소스를 지정할 책임이 있으며 애플리케이션 팀과 긴밀하게 협력합니다. 인프라 팀은 소규모 애플리케이션을 위해 한두 명으로만 구성될 수 있습니다.

팀은 AWS CloudFormation 또는 HashiCorp Terraform과 같은 인프라 프로비저닝 방법에 대한 기술을 갖추어야 합니다. 또한 팀은 Chef, Ansible, Puppet 또는 Salt와 같은 도구를 사용하여 구성 자동화 기술을 개발해야 합니다.

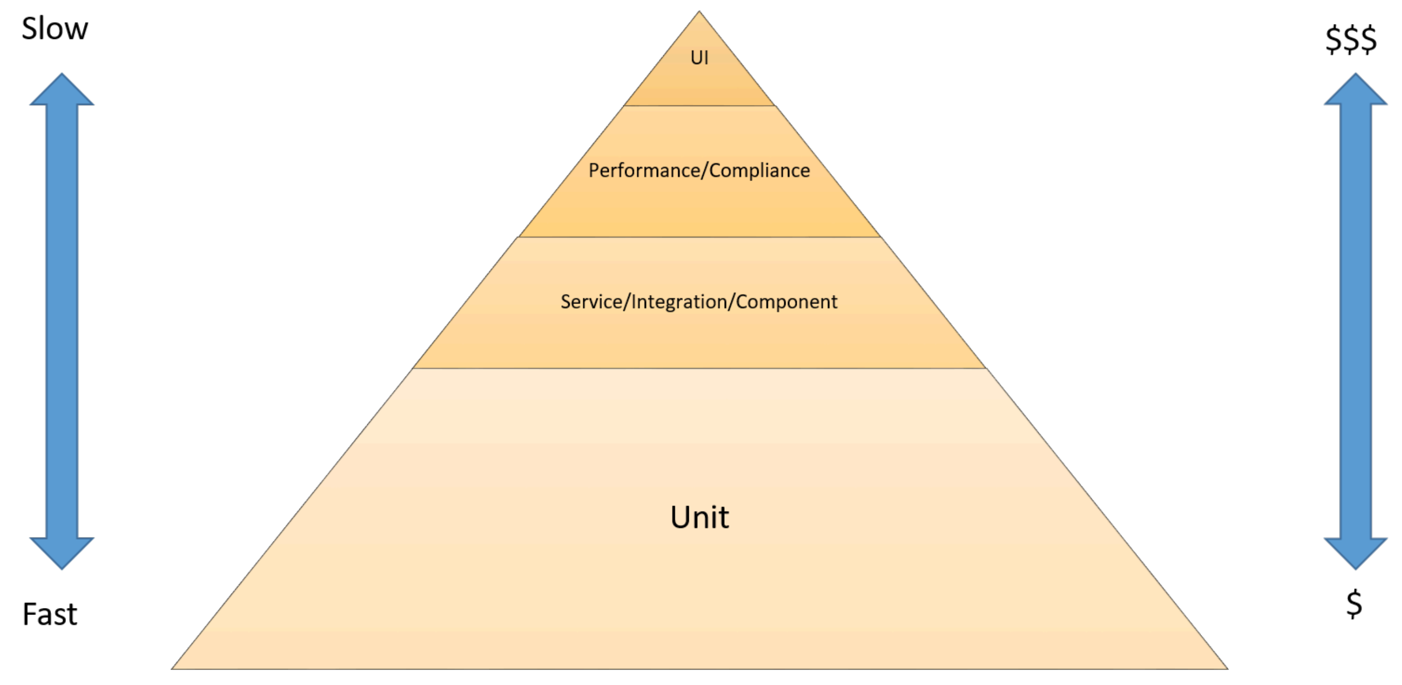
도구 팀

도구 팀은 CI/CD 파이프라인을 구축하고 관리합니다. 파이프라인을 구성하는 인프라와 도구를 담당합니다. 피자 두 판 규모의 팀에 속하지는 않지만, 조직의 애플리케이션 및 인프라 팀에서 사용하는 도구를 만듭니다. 조직은 완성도가 높은 애플리케이션 및 인프라 팀보다 도구 팀이 한 발 앞서 나갈 수 있도록 도구 팀을 지속적으로 발전시켜야 합니다.

도구 팀은 CI/CD 파이프라인의 모든 부분을 구축하고 통합하는 데 능숙해야 합니다. 여기에는 소스 제어 리포지토리, 워크플로 엔진, 빌드 환경, 테스트 프레임워크 및 아티팩트 리포지토리의 구축이 포함됩니다. 이 팀은 Jenkins, GitHub, Artifactory, TeamCity 및 기타 유사한 도구와 함께 AWS CodeStar, AWS CodePipeline, AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild 및 AWS CodeArtifact 등의 소프트웨어를 구현할 수 있습니다. 일부 조직에서는 이를 DevOps 팀이라고 부를 수 있지만, AWS 은(는) 이를 권장하지 않고 대신 DevOps를 소프트웨어 제공의 인력, 프로세스 및 도구의 총합으로 간주할 것을 권장합니다.

지속적 통합 및 지속적 전달의 테스트 단계

세 개의 CI/CD 팀은 CI/CD 파이프라인의 여러 단계에서 소프트웨어 개발 수명 주기에 테스트를 통합해야 합니다. 전반적으로 테스트는 최대한 빨리 시작해야 합니다. 다음 테스트 피라미드는 Mike Cohn 이 Agile로 거둔 성과에서 제공하는 개념입니다. 실행 비용 및 속도와 관련된 다양한 소프트웨어 테스트가 표시됩니다.



CI/CD 테스트 피라미드

단위 테스트는 피라미드 맨 아래에 있습니다. 실행 속도가 가장 빠르고 비용도 가장 저렴합니다. 따라서 단위 테스트가 테스트 전략의 대부분을 차지해야 합니다. 경험상 약 70%를 차지하는 것이 좋습니다. 이 단계에서 잡은 버그는 빠르고 저렴하게 수정할 수 있기 때문에 단위 테스트의 코드 적용 범위는 거의 완전해야 합니다.

서비스, 구성 요소 및 통합 테스트는 피라미드의 단위 테스트 위에 있습니다. 이러한 테스트에는 상세한 환경이 필요하므로 인프라 요구 사항에 더 많은 비용이 들고 실행 속도가 느려집니다. 다음 단계는 성능 및 규정 준수 테스트입니다. 이 테스트에는 프로덕션 품질 환경이 필요하며 아직 더 비쌉니다. UI 및 사용자 승인 테스트는 피라미드의 최상위에 있으며 프로덕션 품질 환경도 필요합니다.

이 모든 테스트는 고품질 소프트웨어를 보장하기 위한 전체 전략의 일부입니다. 그러나 개발 속도를 위해서는 피라미드 아래쪽의 테스트 횟수와 적용 범위에 중점을 둡니다.

다음 단원에서는 CI/CD 단계에 대해 설명합니다.

소스 설정

프로젝트를 시작할 때는 원시 코드와 구성 및 스키마 변경 사항을 저장할 수 있는 소스를 설정해야 합니다. 소스 단계에서 GitHub 또는 AWS CodeCommit에서 호스팅되는 소스 코드 리포지토리와 같은 소스 코드를 선택합니다.

빌드 설정 및 실행

빌드 자동화는 CI 프로세스에 필수입니다. 빌드 자동화를 설정할 때 첫 번째 작업은 올바른 빌드 도구를 선택하는 것입니다. 다음과 같은 여러 빌드 도구가 있습니다.

- Java용 Ant, Maven 및 Gradle
- C/C++용 Make
- JavaScript용 Grunt
- Ruby용 Rake

사용자에게 가장 적합한 빌드 도구는 프로젝트의 프로그래밍 언어와 팀의 기술 집합에 따라 다릅니다. 빌드 도구를 선택한 후에는 빌드 단계와 함께 빌드 스크립트에서 모든 종속성을 명확하게 정의해야 합니다. 또한 최종 빌드 아티팩트의 버전을 관리하는 것이 가장 좋으며, 이렇게 하면 배포가 더 쉬워지고 문제를 추적할 수 있습니다.

빌드

빌드 단계에서 빌드 도구는 소스 코드 리포지토리에 대한 변경 사항을 입력으로 받아 소프트웨어를 빌드하고 다음 유형의 테스트를 실행합니다.

단위 테스트 – 코드의 특정 섹션을 테스트하여 코드가 예상대로 작동하는지 확인합니다. 단위 테스트는 개발 단계에서 소프트웨어 개발자가 수행합니다. 이 단계에서는 정적 코드 분석, 데이터 흐름 분석, 코드 적용 범위 및 기타 소프트웨어 검증 프로세스를 적용할 수 있습니다.

정적 코드 분석 – 이 테스트는 빌드 및 단위 테스트 후 애플리케이션을 실제로 실행하지 않고 수행됩니다. 이 분석은 코딩 오류와 보안 허점을 찾는 데 도움이 될 수 있으며 코딩 지침을 준수하는지 확인할 수 있습니다.

스테이징

스테이징 단계에서는 최종 프로덕션 환경을 반영하는 전체 환경이 생성됩니다. 다음 테스트가 수행됩니다.

통합 테스트 – 소프트웨어 설계와 비교하여 구성 요소 간의 인터페이스를 검증합니다. 통합 테스트는 반복적인 프로세스이며 강력한 인터페이스 및 시스템 무결성을 쉽게 구축할 수 있습니다.

구성 요소 테스트 – 다양한 구성 요소와 결과 간에 전달되는 메시지를 테스트합니다. 이 테스트의 주요 목표는 구성 요소 테스트에서 멱등성이 될 수 있습니다. 테스트에는 매우 큰 데이터 볼륨 또는 에지 상황 및 비정상적인 입력이 포함될 수 있습니다.

시스템 테스트 – 시스템을 종단 간 테스트하고 소프트웨어가 비즈니스 요구 사항을 충족하는지 확인합니다. 여기에는 UI(사용자 인터페이스), API, 백엔드 로직 및 최종 상태의 테스트가 포함될 수 있습니다.

성능 테스트 – 특정 워크로드에서 수행되는 시스템의 응답성 및 안정성을 결정합니다. 성능 테스트는 확장성, 안정성 및 리소스 사용량과 같은 시스템의 다른 품질 속성을 조사, 측정, 검증 또는 확인하는 데에도 사용됩니다. 성능 테스트 유형에는 부하 테스트, 스트레스 테스트 및 스파이크 테스트가 포함될 수 있습니다. 성능 테스트는 사전 정의된 기준에 대한 벤치마킹에 사용됩니다.

규정 준수 테스트 – 코드 변경이 비작동 사양 및/또는 규정의 요구 사항을 준수하는지 확인합니다. 정의된 표준을 구현하고 충족하는지 여부를 결정합니다.

사용자 승인 테스트 – 종단 간 비즈니스 흐름을 검사합니다. 이 테스트는 스테이징 환경에서 최종 사용자가 실행하며 시스템이 요구 사항 사양의 요구 사항을 충족하는지 확인합니다. 일반적으로 고객은 이 단계에서 알파 및 베타 테스트 방법을 사용합니다.

프로덕션

마지막으로 이전 테스트를 통과한 후 프로덕션 환경에서 스테이징 단계가 반복됩니다. 이 단계에서는 전체 프로덕션 환경에 코드를 배포하기전에 서버의 작은 하위 집합이나 한 서버 또는 AWS 리전에만 새 코드를 배포하여 최종 카나리 테스트를 완료할 수 있습니다. 프로덕션에 안전하게 배포하는 방법에 대한 자세한 내용은 배포 방법 단원에서 다룹니다.

다음 단원에서는 이러한 단계와 테스트를 통합하기 위한 파이프라인 구축에 대해 설명합니다.

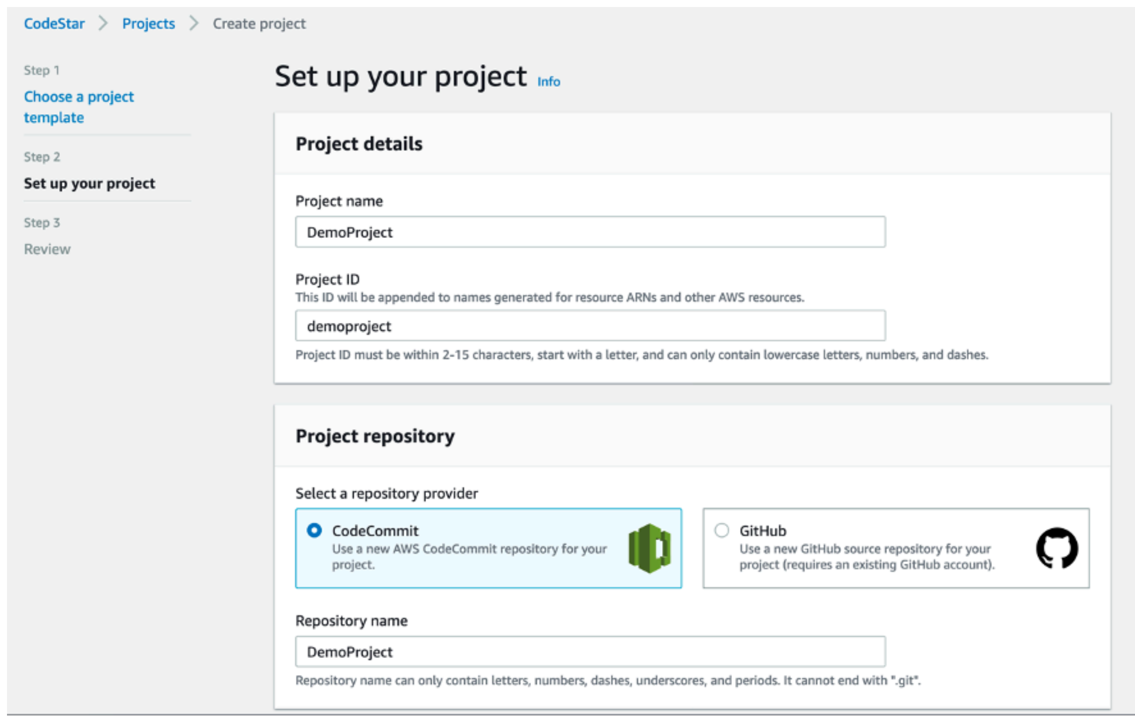
파이프라인 구축

이 단원에서는 파이프라인 구축에 대해 설명합니다. 먼저 CI에 필요한 구성 요소만으로 파이프라인을 설정한 다음 나중에 더 많은 구성 요소와 단계가 있는 지속적인 전달 파이프라인으로 전환합니다. 이 단원에서는 대규모 프로젝트에 대한 AWS Lambda 기능 및 수동 승인 사용, 여러 팀, 분기 및 AWS 리전에 대한 계획을 고려하는 방법에 대해서도 설명합니다.

지속적 통합을 위한 최소한의 실행 가능한 파이프라인부터 시작

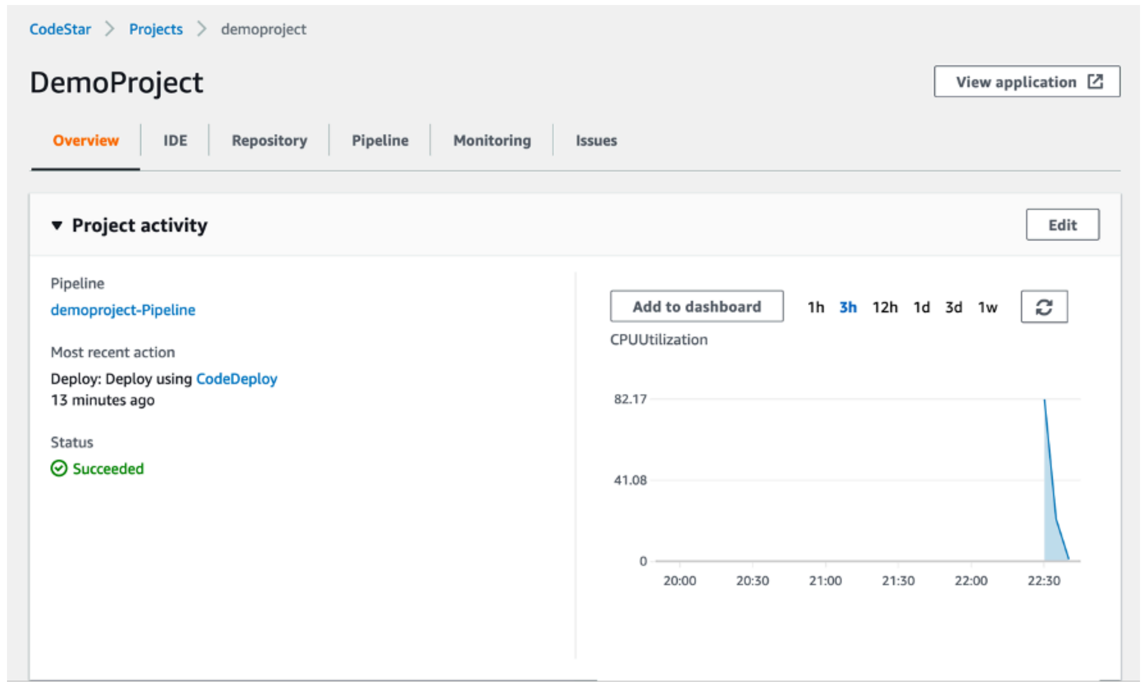
지속적 전달을 위한 조직의 여정은 최소한의 실행 가능한 파이프라인(MVP)에서 시작됩니다. 지속적 통합 및 지속적 전달 구현에서 설명한 것처럼 팀은 코드 스타일 검사를 수행하는 파이프라인이나 배포 없이 단일 단위 테스트를 수행하는 파이프라인 구현과 같은 매우 간단한 프로세스로 시작할 수 있습니다.

핵심 구성 요소는 지속적 전달 오케스트레이션 도구입니다. 이 파이프라인을 구축할 수 있도록 Amazon은 AWS CodeStar를 개발했습니다.



AWS CodeStar 설정 페이지

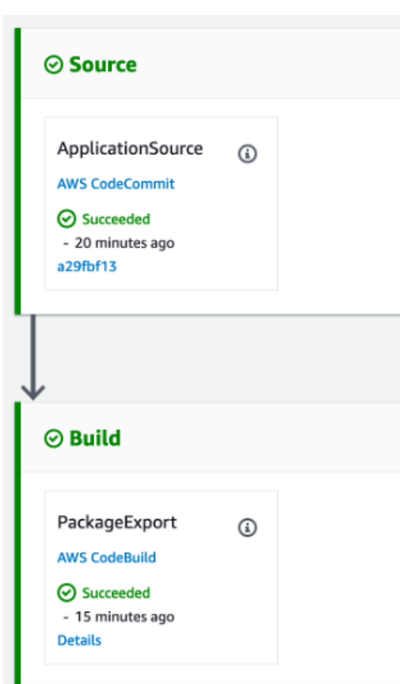
AWS CodeStar는 통합 설정 프로세스, 도구, 템플릿 및 대시보드와 함께 AWS CodePipeline, AWS CodeBuild, AWS CodeCommit 및 AWS CodeDeploy을(를) 사용합니다. AWS CodeStar는 AWS에서 애플리케이션을 신속하게 개발, 빌드 및 배포하는 데 필요한 모든 것을 제공합니다. 이렇게 하면 코드 릴리스를 더 빨리 시작할 수 있습니다. 이미 AWS Management Console에 익숙하고 더 높은 수준의 제어를 원하는 고객은 선택한 개발자 도구를 수동으로 구성하고 필요에 따라 개별 AWS 서비스를 프로비저닝할 수 있습니다.



AWS CodeStar 대시보드

AWS CodePipeline은(는) 빠르고 안정적인 애플리케이션 및 인프라 업데이트를 위해 AWS CodeStar 또는 AWS Management Console을(를) 통해 사용할 수 있는 CI/CD 서비스입니다. AWS CodePipeline 은(는) 사용자가 정의한 릴리스 프로세스 모델을 기반으로 코드가 변경될 때마다 코드를 빌드, 테스트 및 배포합니다. 이를 통해 기능과 업데이트를 안정적으로 신속하게 제공할 수 있습니다. GitHub 등 인기 있는 서드 파티 서비스를 위한 사전 제작 플러그 인을 사용하거나 사용자 지정 플러그 인을 릴리스 프로세스 단계에 통합함으로써 종단 간 솔루션을 손쉽게 구축할 수 있습니다. AWS CodePipeline은(는) 사용한 만큼만 비용을 지불하면 됩니다. 선수금이나 장기 약정을 적용하지 않습니다.

AWS CodeStar 및 AWS CodePipeline의 단계는 소스, 빌드, 스테이징 및 프로덕션 CI/CD 단계에 직접 매핑합니다. 지속적인 전달이 바람직하지만, 소스 리포지토리를 확인하고 빌드 작업을 수행하는 간단한 2단계 파이프라인부터 시작할 수 있습니다.

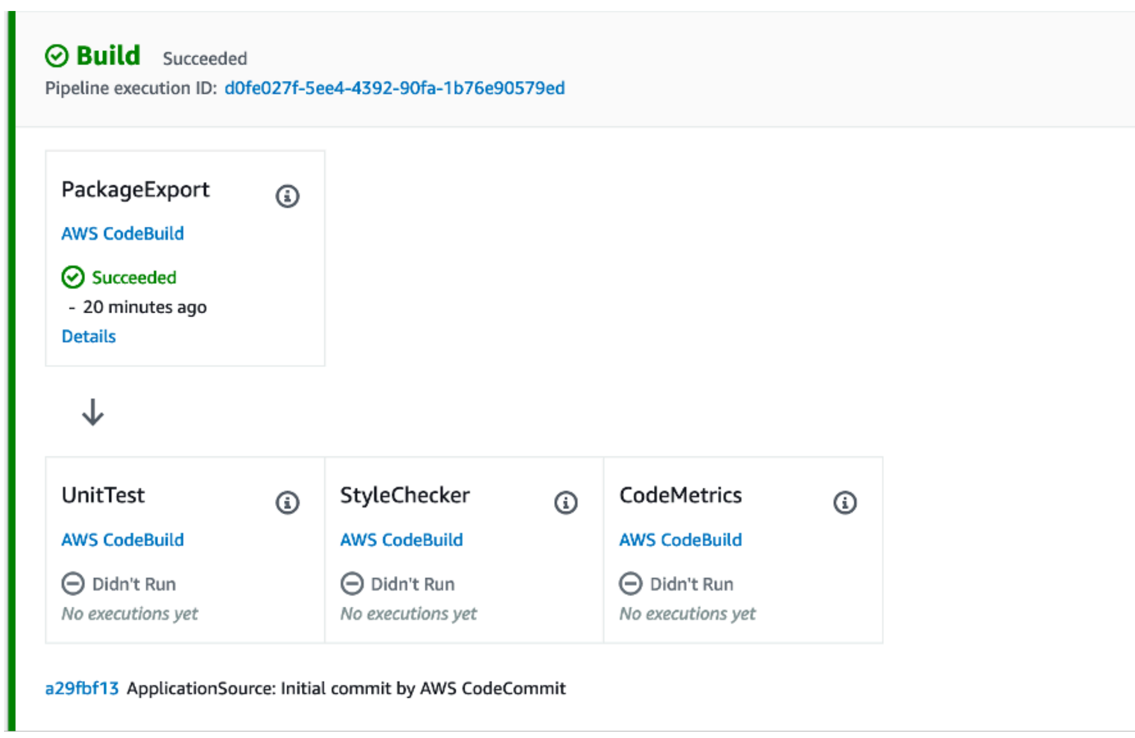


AWS CodePipeline - 소스 및 빌드 단계

AWS CodePipeline의 경우 소스 단계에서는 GitHub, AWS CodeCommit 및 Amazon Simple Storage Service(Amazon S3)의 입력을 받을 수 있습니다. 빌드 프로세스의 자동화는 지속적인 전달을 구현하고 지속적인 배포로 이동하기 위한 중요한 첫 단계입니다. 빌드 아티팩트를 생성할 때 사람이 개입하지 않으면 팀의 부담이 줄어들고 수동 패키징으로 인한 오류가 최소화되며 소모성 아티팩트를 더 자주 패키징할 수 있습니다.

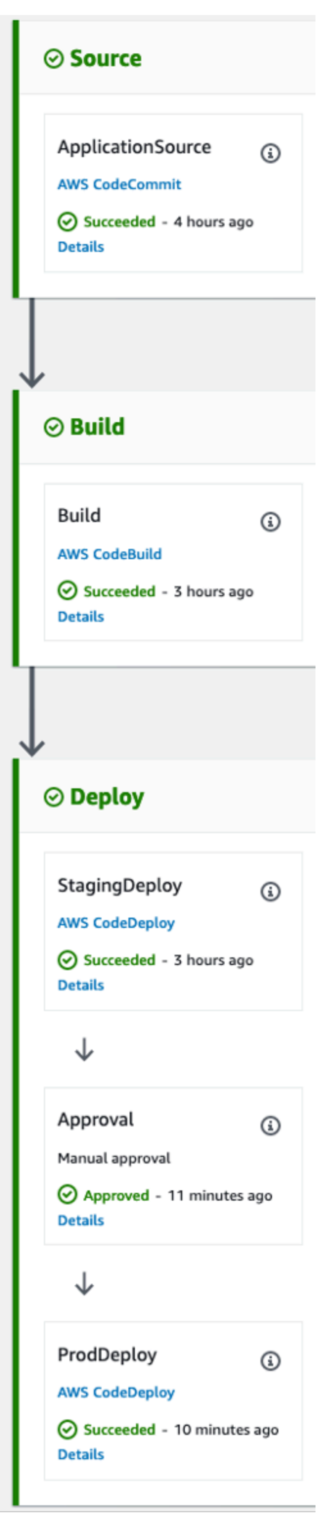
AWS CodePipeline은(는) 완전 관리형 빌드 서비스인 AWS CodeBuild과(와) 원활하게 작동하므로 파이프라인 내에서 코드를 패키징하고 단위 테스트를 실행하는 빌드 단계를 더 쉽게 설정할 수 있습니다. AWS CodeBuild을(를) 사용하면 자체 빌드 서버를 프로비저닝, 관리 또는 확장할 필요가 없습니다. AWS CodeBuild은(는) 지속적으로 확장되고 여러 빌드를 동시에 처리하므로 빌드가 대기열에서 대기하지 않습니다. 또한 AWS CodePipeline은(는) Jenkins, Solano CI 및 TeamCity와 같은 빌드 서버와 통합됩니다.

예를 들어 다음 빌드 단계에서는 세 가지 작업(단위 테스트, 코드 스타일 확인 및 코드 메트릭 수집)이 병렬로 실행됩니다. AWS CodeBuild을(를) 사용하면 부하를 처리하기 위해 빌드 서버를 구축하거나 설치할 필요 없이 이러한 단계를 새 프로젝트로 추가할 수 있습니다.



AWS CodePipeline - 빌드 기능

그림(AWS CodePipeline - 소스 및 빌드 단계)에 표시된 소스 및 빌드 단계는 지원 프로세스 및 자동화와 함께 팀이 지속적인 통합으로 전환하는 데 도움이 됩니다. 이러한 완성도 수준에서 개발자는 주기적으로 빌드 및 테스트 결과에 주의를 기울여야 합니다. 또한 상태가 양호한 단위 테스트 기반을 개발하고 유지해야 합니다. 그 결과 CI/CD 파이프라인에 대한 팀 전체의 신뢰가 강화되고 채택을 가속화합니다.



AWS CodePipeline 단계

지속적 전달 파이프라인

지속적 통합 파이프라인이 구현되고 지원 프로세스가 구축되면 팀은 지속적 전달 파이프라인으로 전환할 수 있습니다. 이러한 전환을 위해서는 팀이 애플리케이션 구축과 배포를 모두 자동화해야 합니다.

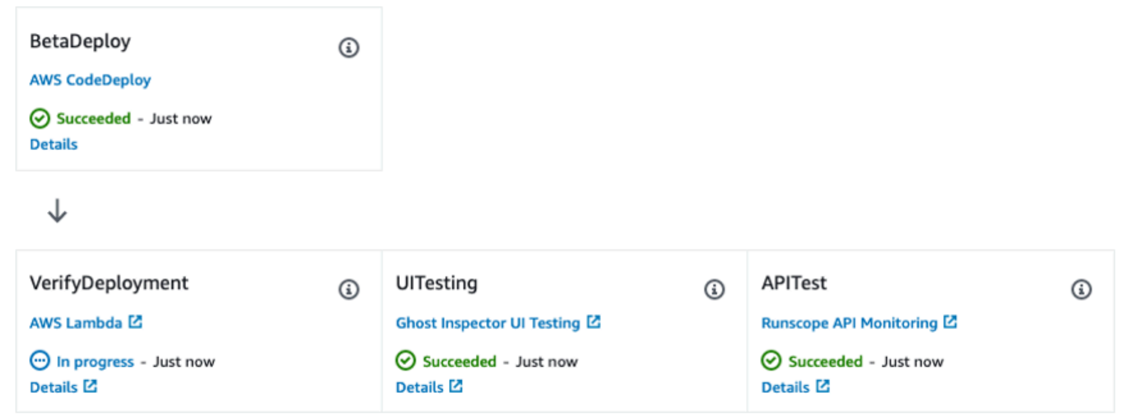
지속적 전달 파이프라인의 특징은 수동 승인 후 프로덕션 단계를 수행하는 스테이징 및 프로덕션 단계가 있다는 것입니다.

지속적 통합 파이프라인을 구축한 것과 같은 방식으로 팀은 배포 스크립트를 작성하여 점진적으로 지속적 전달 파이프라인을 구축할 수 있습니다.

애플리케이션의 필요에 따라 일부 배포 단계를 기존AWS 서비스로 추상화할 수 있습니다. 예를 들어 AWS CodePipeline은(는) Amazon EC2 인스턴스 및 온프레미스에서 실행 중인 인스턴스에 대한 코드 배포를 자동화하는 서비스인 AWS CodeDeploy, Chef를 사용하여 애플리케이션을 운영하는 데 도움이 되는 구성 관리 서비스인 AWS OpsWorks, 웹 애플리케이션 및 서비스를 배포 및 확장하는 서비스인 AWS Elastic Beanstalk에 직접 통합됩니다.

AWS에는 인프라 및 파이프라인에 AWS CodeDeploy을(를) 구현하고 통합하는 방법에 대한 자세한 설명서가 있습니다.

팀이 애플리케이션 배포를 자동화한 후 다양한 테스트를 통해 배포 단계를 확장할 수 있습니다. 예를 들어 다음 그림처럼 Ghost Inspector, Runscope 등과 같은 서비스와의 다른 기본 통합을 추가할 수 있습니다.



AWS CodePipeline- 배포 단계의 코드 테스트

Lambda 작업 추가

AWS CodeStar 및 AWS CodePipeline은(는) AWS Lambda과(와)의 통합을 지원합니다. 이러한 통합을 통해 사용자 환경에서 사용자 지정 리소스 생성, 서드 파티 시스템(예: Slack)과의 통합, 새로 배포된 환경에 대한 확인 수행 등 광범위한 작업을 구현할 수 있습니다.

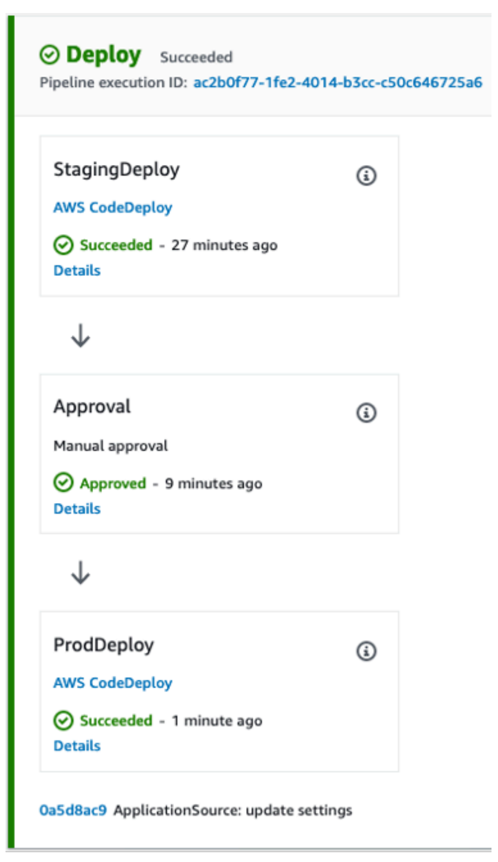
CI/CD 파이프라인에서 Lambda 함수를 사용하여 다음 작업을 수행할 수 있습니다.

- AWS CloudFormation 템플릿을 적용하거나 업데이트하여 사용자 환경에 대한 변경 사항을 롤아웃합니다.
- AWS CloudFormation을(를) 사용하여 파이프라인의 한 단계에서 리소스를 주문형으로 생성하고 다른 단계에서 삭제합니다.
- 표준 이름 레코드(CNAME) 값을 교체하는 Lambda 함수를 사용하여 AWS Elastic Beanstalk에 가동 중지 시간 없이 애플리케이션 버전을 배포합니다.
- Amazon Elastic Container Service(ECS) Docker 인스턴스에 배포합니다.
- AMI 스냅샷을 생성하여 빌드하거나 배포하기 전에 리소스를 백업합니다.
- IRC(인터넷 중계 채팅) 클라이언트에 메시지를 게시하는 등 서드 파티 제품과의 통합을 파이프라인에 추가합니다.

수동 승인

필수 AWS Identity and Access Management(IAM) 권한이 있는 다른 사용자가 작업을 승인하거나 거부할 수 있도록 파이프라인 처리를 중지할 지점의 파이프라인 단계에 승인 작업을 추가할 수 있습니다.

작업이 승인되면 파이프라인 처리가 재개됩니다. 작업이 거부되거나 파이프라인이 작업에 도달하여 중지된 지 7일 이내에 아무도 작업을 승인하거나 거부하지 않는 경우, 작업이 실패한 것과 같으며 파이프라인 처리가 계속되지 않습니다.



AWS CodeDeploy- 수동 승인

CI/CD 파이프라인에 인프라 코드 변경 배포

AWS CodePipeline은(는) 파이프라인의 모든 단계에서 배포 작업으로 AWS CloudFormation을(를) 선택할 수 있습니다. 그런 다음 스택 생성 또는 삭제, 변경 세트 생성 또는 실행 등 AWS CloudFormation 에서 수행할 특정 작업을 선택할 수 있습니다. 스택은 AWS CloudFormation 개념이며 관련 AWS 리소스 그룹을 나타냅니다. 코드형 인프라를 프로비저닝하는 방법에는 여러 가지가 있지만, AWS CloudFormation은(는) 가장 포괄적인 AWS 리소스 세트를 코드로 설명할 수 있는 확장 가능하고 완전한 솔루션으로 AWS에서 권장하는 포괄적인 도구입니다. AWS에서는 AWS CodePipeline 프로젝트의 AWS CloudFormation을(를) 사용하여 인프라 변경 및 테스트를 추적할 것을 권장합니다.

서버리스 애플리케이션용 CI/CD

AWS CodeStar, AWS CodePipeline, AWS CodeBuild 및AWS CloudFormation을(를) 사용하여 서버리스 애플리케이션용 CI/CD 파이프라인을 구축할 수도 있습니다. 서버리스 애플리케이션은 Amazon Cognito, Amazon S3, Amazon DynamoDB 및 AWS Lambda과(와) 같은 관리형 서비스를 이벤트 기반 서비스와 통합하고 서버 관리가 필요하지 않은 방식으로 애플리케이션을 배포합니다. 서버리스 애플리케이션 개발자인 경우 AWS CodePipeline, AWS CodeBuild 및 AWS CloudFormation을(를) 조합하

여 AWS Serverless Application Model로 구축된 템플릿에 표현되는 서버리스 애플리케이션의 구축, 테스트 및 배포를 자동화할 수 있습니다. 자세한 내용은 Lambda 기반 애플리케이션의 배포 자동화의 AWS Lambda 설명서를 참조하십시오.

AWS Serverless Application Model Pipelines(AWS SAM 파이프라인)를 사용하여 조직의 모범 사례를 따르는 안전한 CI/CD 파이프라인을 생성할 수도 있습니다. AWS SAM Pipelines는 몇 분 내에 배포 빈도 가속화, 변경을 위한 리드 타임 단축 및 배포 오류 감소와 같은 CI/CD의 이점에 대한 액세스를 제공하는 새로운 AWS SAM CLI 기능입니다. AWS SAM Pipelines에는 AWS 배포 모범 사례를 따르는 AWS CodeBuild/CodePipeline용 기본 파이프라인 템플릿 집합이 함께 제공됩니다. 자세한 내용과 자습서를 보려면 AWS SAM Pipelines 소개 블로그를 참조하십시오.

여러 팀, 분기 및 AWS 리전을 위한 파이프라인

대규모 프로젝트의 경우 여러 프로젝트 팀이 서로 다른 구성 요소에서 작업하는 경우가 일반적입니다. 여러 팀이 단일 코드 리포지토리를 사용하는 경우, 각 팀이 자체 분기를 갖도록 매핑할 수 있습니다. 또한 프로젝트의 최종 병합을 위한 통합 또는 릴리스 분기가 있어야 합니다. 서비스 지향 또는 마이크로서비스 아키텍처를 사용하는 경우, 각 팀은 자체 코드 리포지토리를 보유할 수 있습니다.

첫 번째 시나리오에서는 단일 파이프라인을 사용하는 경우 파이프라인을 차단하여 한 팀이 다른 팀의 진행 상황에 영향을 미칠 수 있습니다. AWS는 팀 분기에 대한 특정 파이프라인과 최종 제품 제공을 위한 또 다른 릴리스 파이프라인을 생성할 것을 권장합니다.

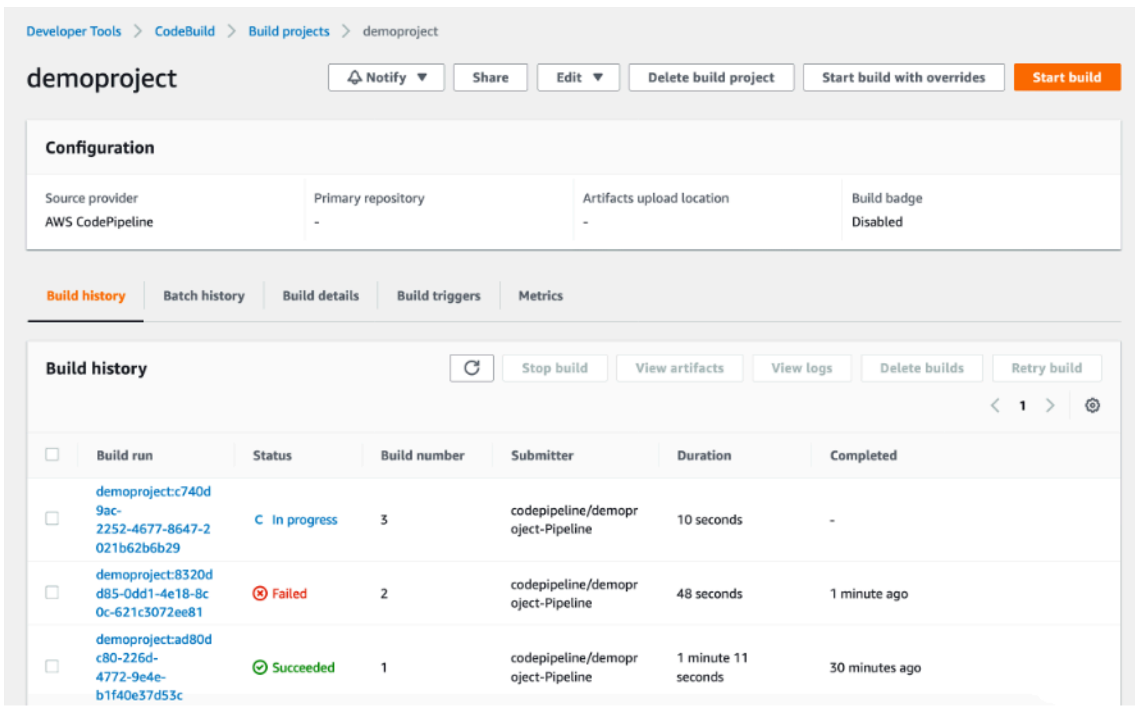
AWS CodeBuild과(와) 파이프라인 통합

AWS CodeBuild은(는) 조직에서 거의 무제한으로 고가용성 빌드 프로세스를 구축할 수 있도록 설계되었습니다. AWS CodeBuild에서는 널리 사용되는 여러 언어에 대한 빠른 시작 환경과 사용자가 지정한 Docker 컨테이너를 실행할 수 있는 기능을 제공합니다.

Git 및 CodePipeline Lambda 작업 외에도 AWS CodeCommit, AWS CodePipeline 및 AWS CodeDeploy과(와)의 긴밀하게 통합되는 이점을 갖춘CodeBuild 도구는 매우 유연합니다.

빌드 전/후 작업을 포함하여 각 빌드 단계를 식별하는 `buildspec.yml` 파일을 포함하거나 CodeBuild 도구로 지정된 작업을 통해 소프트웨어를 빌드할 수 있습니다.

CodeBuild 대시보드를 사용하여 각 빌드의 자세한 기록을 볼 수 있습니다. 이벤트는 Amazon CloudWatch Logs 로그 파일로 저장됩니다.



AWS CodeBuild의 CloudWatch Logs 로그 파일

Jenkins와 파이프라인 통합

Jenkins 빌드 도구를 사용하여 제공 파이프라인을 생성할 수 있습니다. 이 파이프라인은 지속적 전달 단계를 구현하기 위한 단계를 정의하는 표준 작업을 사용합니다. 그러나 Jenkins가 다시 시작될 때 파이프라인의 현재 상태가 유지되지 않고 수동 승인을 구현하는 것이 간단하지 않으며 복잡한 파이프라인의 상태를 추적하는 것이 복잡할 수 있기 때문에 이러한 접근 방식은 대규모 프로젝트에 적합하지 않을 수 있습니다.

대신 AWS에서는 AWS Code Pipeline Plugin을 사용하여 Jenkins로 지속적인 전달을 구현하는 것이 좋습니다. 이 플러그 인을 사용하면 Groovy와 유사한 도메인 별 언어를 사용하여 복잡한 워크 플로를 설명할 수 있으며 복잡한 파이프라인을 오케스트레이션하는 데 사용할 수 있습니다. AWS Code Pipeline Plugin의 기능은 파이프라인에 정의된 단계의 현재 진행 상황을 시각화하는 Pipeline Stage View Plugin 또는 서로 다른 분기에서 빌드를 그룹화하는 Pipeline Multibranch Plugin과 같은 위성 플러그 인을 사용하여 향상할 수 있습니다.

AWS에서는 파이프라인 구성을 Jenkinsfile에 저장하고 소스 코드 리포지토리에 체크인할 것을 권장합니다. 이를 통해 파이프라인 코드의 변경 사항을 추적할 수 있으며 Pipeline Multibranch Plugin으로 작업할 때 더욱 중요해집니다. 또한 AWS에서는 파이프라인을 여러 단계로 나누는 것이 좋습니다. 이렇게 하면 파이프라인 단계를 논리적으로 그룹화하고 Pipeline Stage View Plugin이 파이프라인의 현재 상태를 시각화할 수 있습니다.

다음 그림은 Pipeline Stage View Plugin으로 시각화된 네 가지 정의된 단계가 있는 샘플 Jenkins 파이프라인을 보여줍니다.

Pipeline Stage View Plugin으로 시각화된 Jenkins 파이프라인의 정의된 단계

배포 방법

지속적 전달 프로세스에서 새 버전의 소프트웨어를 출시하기 위한 여러 배포 전략과 변형을 고려할 수 있습니다. 이 단원에서는 가장 일반적인 배포 방법인 한 번에 모두(바로 배포), 롤링, 변경 불가능 및 블루/그린에 대해 설명합니다. AWS는 이러한 방법 중 AWS CodeDeploy 및 AWS Elastic Beanstalk에서 지원되는 방법을 나타냅니다.

다음 표에는 각 배포 방법의 특성이 요약되어 있습니다.

방법	배포 실패로 인한 영향	배포 시간	가동 중지 없음	DNS 변경 없음	롤백 프로세스	코드 배포 위치
바로 배포	가동 중지	🕒	X	✓	재배포	기존 인스턴스
롤링	단일 배치가 서비스에서 제외됨. 실패하기 전 성공한 배치가 새 애플리케이션 버전 실행.	🕒🕒†	✓	✓	재배포	기존 인스턴스
추가 배치를 사용한 롤링 (beanstalk)	첫 번째 배치가 실패할 경우 최소화. 그렇지 않은 경우 롤링과 유사함.	🕒🕒🕒†	✓	✓	재배포	새 인스턴스 및 기존 인스턴스
변경 불가능	최소화	🕒🕒🕒	✓	✓	재배포	새 인스턴스

방법	배포 실패로 인한 영향	배포 시간	가동 중지 없음	DNS 변경 없음	롤백 프로세스	코드 배포 위치
		🕒				
트래픽 분할	최소화	🕒🕒🕒🕒	✓	✓	트래픽 재라우팅 및 새 인스턴스 종료	새 인스턴스
블루/그린	최소화	🕒🕒🕒🕒	✓	X	이전 환경으로 다시 전환	새 인스턴스

한 번에 모두(바로 배포)

한 번에 모두(바로 배포)는 기존 서버 플릿에 새 애플리케이션 코드를 롤아웃하는 데 사용할 수 있는 방법입니다. 이 방법은 한 배포 작업에서 모든 코드를 대체합니다. 플릿의 모든 서버가 한 번에 업데이트되므로 가동 중단이 필요합니다. 기존 DNS 레코드를 업데이트할 필요가 없습니다. 배포에 실패한 경우 작업을 복원하는 유일한 방법은 모든 서버에 코드를 다시 배포하는 것입니다.

AWS Elastic Beanstalk에서 이 배포는 한 번에 모두라고 하며, 단일 및 부하 분산 애플리케이션에 사용할 수 있습니다. AWS CodeDeploy에서 이 배포 방법은 배포 구성이 `AllAtOnce`인 바로 배포라고 합니다.

롤링 배포

롤링 배포를 사용하면 모든 플릿이 한 번에 업그레이드되지 않도록 여러 부분으로 나뉩니다. 배포 프로세스 중에 두 가지 소프트웨어 버전(새 버전과 이전 버전)이 동일한 플릿에서 실행됩니다. 이 방법을 사용하면 가동 중지 없이 업데이트할 수 있습니다. 배포에 실패하면 플릿의 업데이트된 부분만 영향을 받습니다.

카나리 릴리스라고 하는 롤링 배포 방법의 변형은 처음에는 아주 적은 비율의 서버에 새 소프트웨어 버전을 배포하는 작업이 포함됩니다. 이렇게 하면 변경 사항의 영향을 최소화하면서 소수의 서버에서 소

프트웨어가 프로덕션에서 어떻게 동작하는지 관찰할 수 있습니다. 카나리 배포에서 오류 발생률이 높아지면 소프트웨어가 롤백됩니다. 그렇지 않으면 새 버전을 사용하는 서버의 비율이 점차 증가합니다.

AWS Elastic Beanstalk는 두 가지 배포 옵션인 추가 배치를 사용한 롤링 및 롤링으로 롤링 배포 패턴을 따랐습니다. 이러한 옵션을 사용하면 서버의 서비스를 중단하기 전에 애플리케이션을 먼저 확장하여 배포 중에 전체 기능을 유지할 수 있습니다. AWS CodeDeploy에서는 이 패턴을 OneAtTime 및 HalfAtaTime과 같은 패턴을 사용하여 바로 배포의 변형으로 수행합니다.

변경 불가능 및 블루/그린 배포

변경 불가능 패턴은 애플리케이션 코드의 새 구성 또는 버전으로 완전히 새로운 서버 집합을 시작하여 애플리케이션 코드의 배포를 지정합니다. 이 패턴은 간단한 API 호출로 새 서버 리소스가 생성되는 클라우드 기능을 활용합니다.

블루/그린 배포 전략은 변경 불가능 배포 유형이며 다른 환경을 만들어야 합니다. 새 환경이 가동되고 모든 테스트를 통과하면 트래픽이 이 새 배포로 이동합니다. 결정적으로 이전 환경인 “블루” 환경은 롤백이 필요한 경우 유휴 상태로 유지됩니다.

AWS Elastic Beanstalk는 변경 불가능 및 블루/그린 배포 패턴을 지원합니다. AWS CodeDeploy도 블루/그린 패턴을 지원합니다. AWS 서비스가 이러한 변경 불가능 패턴을 달성하는 방법에 대한 자세한 내용은 AWS 기반 블루/그린 배포 백서를 참조하세요.

데이터베이스 스키마 변경 사항

최신 소프트웨어에는 일반적으로 데이터베이스 계층이 있습니다. 일반적으로 데이터와 데이터 구조를 모두 저장하는 관계형 데이터베이스가 사용됩니다. 지속적인 전송 프로세스에서 데이터베이스를 수정해야 하는 경우가 흔히 있습니다. 관계형 데이터베이스의 변경 사항을 처리하려면 특별한 고려 사항이 필요하며 애플리케이션 바이너리를 배포할 때 발생하는 문제 이외에도 다른 문제가 있습니다. 일반적으로 애플리케이션 바이너리를 업그레이드할 때 애플리케이션을 중지하고 업그레이드한 다음 다시 시작합니다. 애플리케이션 상태는 애플리케이션 외부에서 처리되므로 신경 쓰지 않아도 됩니다.

데이터베이스를 업그레이드할 때는 데이터베이스에 많은 상태가 포함되어 있지만 로직과 구조는 비교적 적기 때문에 상태를 고려해야 합니다.

변경 사항이 적용되기 전과 후의 데이터베이스 스키마는 데이터베이스의 다른 버전으로 간주되어야 합니다. Liquibase 및 Flyway와 같은 도구를 사용하여 버전을 관리할 수 있습니다.

일반적으로 이러한 도구에는 다음 방법 중 몇 가지 변형이 사용됩니다.

- 데이터베이스 버전이 저장된 데이터베이스에 테이블을 추가합니다.
- 데이터베이스 변경 명령을 추적하고 버전이 지정된 변경 세트에 함께 결합합니다. Liquibase의 경우, 이러한 변경 사항은 XML 파일에 저장됩니다. Flyway는 변경 세트가 별도의 SQL 파일로 처리되거나 때로는 더 복잡한 전환을 위해 별도의 Java 클래스로 처리되는 약간 다른 방법을 사용합니다.
- Liquibase에서 데이터베이스를 업그레이드하라는 요청을 받으면 메타데이터 테이블을 보고 데이터베이스를 최신 버전으로 업데이트하기 위해 실행할 변경 세트를 결정합니다.

모범 사례 요약

다음은 CI/CD에 대한 몇 가지 모범 사례의 규범입니다.

해야 할 일:

- 코드형 인프라로 간주합니다.
  - 인프라 코드에 버전 제어를 사용합니다.
  - 버그 추적/티켓팅 시스템을 활용합니다.
  - 변경 사항이 적용되기 전에 동료가 이를 검토하도록 합니다.
  - 인프라 코드 패턴/설계를 설정합니다.
  - 코드 변경과 같은 인프라 변경을 테스트합니다.
- 최대 12명의 지지 구성원으로 구성된 통합 팀에 개발자를 배치합니다.
- 모든 개발자가 장기 실행 기능 분기 없이 기본 트렁크에 코드를 자주 커밋하도록 합니다.
- 조직 전체에서 Maven 또는 Gradle과 같은 빌드 시스템을 일관되게 채택하고 빌드를 표준화합니다.
- 개발자가 코드 베이스를 100% 적용하도록 단위 테스트를 빌드하도록 합니다.
- 단위 테스트가 기간, 횟수 및 범위에서 전체 테스트의 70%를 차지하는지 확인합니다.
- 단위 테스트가 최신 상태이며 간과하지 않도록 합니다. 단위 테스트 실패는 무시하지 말고 수정해야 합니다.
- 지속적인 전송 구성을 코드로 취급합니다.
- 역할 기반 보안 제어(즉, 수행 작업, 사용자 및 시기)를 설정합니다.
  - 가능한 모든 리소스를 모니터링/추적합니다.
  - 서비스, 가용성 및 응답 시간에 대해 경고합니다.
  - 캡처하고, 학습하고 개선합니다.
  - 팀원 모두와 액세스 권한을 공유합니다.
  - 수명 주기에 대한 지표 및 모니터링을 계획합니다.
- 표준 지표를 유지 및 추적합니다.
  - 빌드 수입니다.
  - 배포 횟수입니다.
  - 변경 사항이 프로덕션에 도달하는 데 걸리는 평균 시간입니다.
  - 첫 번째 파이프라인 단계에서 각 단계까지의 평균 시간입니다.
  - 프로덕션에 도달하는 변경 횟수입니다.

  - 평균 빌드 시간입니다.
- 각 분기 및 팀에 고유한 파이프라인을 사용합니다.

하지 말아야 할 일:

- 크고 복잡한 병합이 있는 장기 실행 분기가 있습니다.
- 수동 테스트를 실시합니다.
- 수동 승인 프로세스, 게이트, 코드 검토 및 보안 검토를 수행합니다.

결론

지속적인 통합과 지속적인 전송은 조직의 애플리케이션 팀에 이상적인 시나리오를 제공합니다. 개발자는 코드를 리포지토리에 푸시하기만 하면 됩니다. 이 코드는 통합, 테스트, 배포, 다시 테스트, 인프라와 병합되고 보안 및 품질 검토를 거쳐 매우 높은 신뢰도로 배포될 수 있습니다.

CI/CD를 사용하면 코드 품질이 향상되고 중대한 변경 사항이 없을 것이라는 확신을 갖고 소프트웨어 업데이트가 신속하게 제공됩니다. 모든 릴리스의 영향은 프로덕션 및 운영 데이터와 관련이 있을 수 있습니다. 조직의 클라우드 전환에서 중요한 DevOps 사례인 다음 주기를 계획하는 데도 사용할 수 있습니다.

추가 자료

이 백서에서 설명하는 주제에 대한 자세한 내용은 다음 AWS 백서를 참조하십시오.

- AWS 기반의 배포 옵션 개요
- AWS 기반 블루/그린 배포
- Jenkins를 AWS CodeBuild 및 AWS CodeDeploy와 통합하여 CI/CD 파이프라인 설정
- AWS 기반 마이크로서비스
- AWS 기반 Docker: 클라우드상의 컨테이너 실행하기

기여자

다음은 본 문서를 작성하는 데 도움을 준 개인 및 조직입니다.

- Amrish Thakkar, AWS 수석 솔루션 아키텍트
- David Stacy, 선임 컨설턴트 - DevOps, AWS Professional Services
- Asif Khan, AWS 솔루션스 아키텍트
- Xiang Shen, AWS 선임 솔루션스 아키텍트

문서 개정

이 백서의 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하세요.

업데이트 기록-변경	update-history-description	update-history-date
최초 게시	처음 게시된 백서	2021년 10월 27일
최초 게시	처음 게시된 백서	2017년 6월 1일

고지 사항

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

© 2021 Amazon Web Services, Inc. 또는 자회사. All rights reserved.