MLOps

지속적 통합과 지속적 배포(CI/CD)

강사 양석환

CI / CD 개요

CI/CD (지속적 통합/지속적 제공 또는 배포)

- CI/CD
  - 지속적 통합(CI, Continuous Integration) 및 지속적 제공(CD, Continuous Delivery) 또는 지속적 배포(CD, Continuous Deployment)
  - 새 버전의 소프트웨어를 제공하기 위해 수행해야 할 일련의 단계를 가리킴
  - 애플리케이션 개발팀이 더 자주, 안정적으로 코드 변경을 제공하기 위해 사용하는 일련의 작업 방식으로 구성
  - 소프트웨어 개발 라이프사이클을 간소화하고 가속화하는 것을 목표로 함
    - 사용자 정의 애플리케이션이 기업 차별화의 핵심 요소가 되면서 코드를 얼마나 빨리 배포할 수 있느냐가 경쟁력의 차별화 요소가 되었음

CI/CD의 역할

- CI/CD는
  - DevOps팀을 위한 권장 사항이자 애자일 방법론의 권장 사항의 하나

    → 개발자와 IT 운영 팀이 협업하여 소프트웨어를 배포하는 DevOps 방법론의 근간이 됨
  - 통합과 제공을 자동화 함으로써
    - 소프트웨어 개발팀이 코드 품질과 소프트웨어 보안을 보장하는 동시에
    - 비즈니스 요구사항을 충족하는데 집중할 수 있게 해 줌

CI/CD의 중요성

- CI/CD를 사용하면
  - 버그 및 코드 오류 예방
  - 동시에 지속적인 SW 개발 및 업데이트 주기의 유지에 도움이 됨
  - 지속적 제공: 애플리케이션 제공 속도를 저하시키는 수동 프로세스로 인한 운영 팀의 업무 과다 문제 해결
  - 지속적 배포: 파이프라인의 다음 단계를 자동화함으로써 지속적 제공의 장점을 활용

CI/CD의 중요성

- CI/CD 기능의 활용
  - 애플리케이션이 커짐에 따라 증가하는 복잡성을 줄이고 효율성을 높이며, 워크플로우를 간소화할 수 있음
  - 기존에 새 코드를 커밋에서 프로덕션으로 가져오는 데 필요했던 수동 개입을 CI/CD가 자동화

    → 다운타임 최소화, 코드 릴리스 주기 단축 효과 유발
  - 코드의 업데이트와 변경 사항을 더 빠르게 통합 → 사용자 피드백을 더 자주 효과적으로 통합

    → 그 결과 사용자에게 긍정적인 결과를 제공할 수 있으며 전체적인 고객 만족도가 향상됨

실제 사례에서의 CI/CD

- 실제 사례에서 지속적 배포
  - 개발자가 애플리케이션에 변경 사항을 작성한 후 몇 분 이내에 클라우드 애플리케이션을 자동으로 실행할 수 있는 것을 의미 (자동화된 테스트를 통과한 것으로 간주)
    → 이를 통해 사용자 피드백을 지속적으로 수신하고 통합하는 일이 훨씬 수월해짐
  - 이러한 연결된 모든 CI/CD 사례는 애플리케이션 배포의 리스크를 줄여줌
    - 애플리케이션 변경 사항을 한꺼번에 배포하지 않고 작은 단위로 세분화하여 더욱 손쉽게 배포할 수 있음
  - 그러나 프로덕션 이전의 파이프라인 단계에는 수동 게이트가 없으므로 지속적 배포는 잘 설계된 테스트 자동화에 크게 의존 → 따라서 CI/CD 파이프라인에서 다양한 테스트 및 릴리스 단계를 수용하기 위해 자동화된 테스트를 작성해야 하므로 지속적 배포에는 많은 선행 투자가 필요함

지속적 통합

지속적 통합

- 지속적 통합(CI, Continuous Integration)
  - 코드의 변경 사항을 더욱 빈번하게 병합하는 것을 용이하게 해 주는 개발자용 자동화 프로세스
    - 작은 코드 변경을 수시로 구현해 버전 제어 저장소에 체크인하도록 유도하는 코딩 원칙이자 일련의 방식
    - 코드 변경 사항을 공유 소스 코드 저장소에 자동으로 자주 통합하는 것으로 CI를 수행함
    - 지속적 통합이 이루어지면 병합된 코드 변경 사항의 신뢰성을 보장하기 위해 자동화된 테스트 단계가 트리거 됨
    - 동시에 개발 중인 애플리케이션의 분기가 너무 많아 상호 충돌할 가능성이 있는 문제에 대한 해결책으로 CI를 고려할 수 있음

지속적 통합

- 지속적 통합의 필요성
  - 현대적인 애플리케이션 개발은...
    - 여러 개발자들이 동일한 애플리케이션의 각기 다른 기능을 동시에 작업할 수 있도록 하는 것이 목표
    - 그러나 특정한 날을 정해 모든 분기 소스 코드를 병합하는 경우, 결과적으로 반복적인 수작업에 많은 시간 소모
      - 고립된 상태에서 작업하는 개발자가 애플리케이션을 변경하는 경우 다른 개발자들이 동시에 적용하는 여러 변경 사항들과 충돌할 가능성이 있기 때문
      - 팀이 하나의 클라우드 기반 통합 개발 환경(Integrated Development Environment, IDE)에 동의하지 않고 각 개발자가 각자의 로컬 IDE를 커스터마이징 하는 경우 문제는 더욱 심화됨

지속적 통합

- 대부분의 현대 애플리케이션에서는 다양한 플랫폼과 툴을 사용해 코드를 개발해야 하므로 팀은 변경을 통합하고 검증할 일관적인 메커니즘이 필요함
  - 애플리케이션을 빌드, 패키징, 테스트하기 위한 자동화된 방법을 구축

    → 일관적인 통합 프로세스를 두면 개발자는 자연스럽게 더 자주 코드 변경을 커밋하게 되고

    → 이것이 더 나은 협업과 코드 품질로 이어짐

지속적 통합은 어떻게 동작하는가?

- 지속적 통합은 어떻게 동작하는가?
  - 지속적 통합은 프로세스 역학과 자동화를 기반으로 하는 개발 철학
  - 지속적 통합을 실천하는 개발자는 버전 제어 리포지토리에 자주 코드를 커밋함
  - 대부분 팀은 최소 일일 단위의 코드 커밋 표준을 설정
    - 장기간에 걸쳐 개발된 큰 코드보다 작은 규모로 작게 변화된 코드에서 결함과 기타 소프트웨어 품질 문제를 파악하기가 더 쉽기 때문
    - 또한 개발자가 짧은 커밋 주기에 따라 작업하면 여러 개발자가 동일한 코드를 편집하고 커밋할 때 병합해야 하는 상황이 발생할 가능성도 낮출 수 있음

지속적 통합은 어떻게 동작하는가?

- **지속적 통합을 구현하는 팀은 버전 제어 구성과 실행 정의부터 시작하는 경우가 많음**
  - 코드 체크인은 빈번하게 이뤄지지만 애자일 팀이 기능과 수정을 개발하는 기간은 그보다 더 짧거나 길 수 있음
  - 따라서 지속적 통합을 실행하는 개발 팀은 다양한 방법으로 프로덕션에 배포할 준비가 된 기능과 코드를 제어함

- **많은 팀이 런타임에 기능과 코드를 켜고 끄기 위한 구성 메커니즘인 기능 플래그를 사용함**
  - 아직 개발 중인 기능은 코드에서 기능 플래그로 래핑되어 주 분기와 함께 프로덕션에 배포되지만, 사용할 준비가 될 때까지 꺼진 상태로 유지됨
  - 최근 연구에서 기능 플래그를 사용하자 데브옵스팀의 개발 빈도가 9배 증가한 것으로 나타남
    - 클라우드비스(CloudBees), 옵티마이즐리 롤아웃(Optimizely Rollouts), 런치다클리(LaunchDarkly)와 같은 기능 플래그 툴은 CI/CD 툴과 통합되어 기능 수준 구성을 지원

그만큼 더 열심히 일했다???

지속적 통합은 어떻게 동작하는가?

- **지속적 통합을 구현하는 팀은 버전 제어 구성과 실행 정의부터 시작하는 경우가 많음**
  - 코드 체크인은 빈번하게 이뤄지지만 애자일 팀이 기능과 수정을 개발하는 기간은 그보다 더 짧거나 길 수 있음
  - 따라서 지속적 통합을 실행하는 개발 팀은 다양한 방법으로 프로덕션에 배포할 준비가 된 기능과 코드를 제어함

- **많은 팀이 런타임에 기능과 코드를 켜고 끄기 위한 구성 메커니즘인 기능 플래그를 사용함**
  - 아직 개발 중인 기능은 코드에서 기능 플래그로 래핑되어 주 분기와 함께 프로덕션에 배포되지만, 사용할 준비가 될 때까지 꺼진 상태로 유지됨
  - 최근 연구에서 기능 플래그를 사용하자 데브옵스팀의 개발 빈도가 9배 증가한 것으로 나타남
    - 클라우드비스(CloudBees), 옵티마이즐리 롤아웃(Optimizely Rollouts), 런치다클리(LaunchDarkly)와 같은 기능 플래그 툴은 CI/CD 툴과 통합되어 기능 수준 구성을 지원

그만큼 더 열심히 일했다???

지속적 통합은 어떻게 동작하는가?

- **자동화된 빌드**
  - 자동화된 빌드 프로세스에서는 모든 소프트웨어, 데이터베이스 및 기타 구성요소가 함께 패키징 됨
  - 예를 들어 자바 애플리케이션을 개발한다면 지속적 통합은 HTML, CSS, 자바스크립트와 같은 모든 정적 웹 서버 파일을 자바 애플리케이션 및 모든 데이터베이스 스크립트와 함께 패키징 함
  - 대부분 CI/CD 툴은 개발자가 필요에 따라 빌드를 실행할 수 있게 해 줌
    - 버전 제어 리포지토리의 코드 커밋에 의해 또는 정해진 일정에 따라 트리거되는 방식
    - 팀은 규모에 가장 잘 맞는 빌드 일정과 일일 예상 커밋 수, 그리고 기타 애플리케이션 고려 사항을 결정
    - 최선의 방법은 커밋과 빌드가 빠른지 여부를 확인하는 것
    - 빠르지 않을 경우 신속한 코딩과 수시 커밋을 실현하는 데 지장이 발생할 수 있음

지속적 통합은 어떻게 동작하는가?

- **자동화된 테스트**
  - 지속적 통합에서 모든 소프트웨어 및 데이터베이스 구성요소를 패키징 할 뿐만 아니라 자동화에서 단위 테스트 및 기타 유형의 테스트도 실행함
  - 테스트는 개발자에게 코드 변경이 장애를 유발하지 않는지 확인하기 위한 중요한 피드백을 제공함

- **지속적 테스트와 보안 자동화**
  - 자동화된 테스트 프레임워크는 품질 보장 엔지니어가 다양한 유형의 테스트를 정의, 실행하고 자동화하는 데 유용함
  - 이런 테스트는 개발팀이 소프트웨어 빌드의 통과 또는 실패 여부를 파악할 수 있게 해 줌
  - 여기에는 모든 스프린트의 끝에 개발되고 전체 애플리케이션의 회귀 테스트에 들어가는 기능 테스트도 포함됨
  - 회귀 테스트는 코드 변경이 테스트 커버리지가 있는 애플리케이션 기능 영역 전반에서 개발된 하나 또는 여러 테스트의 실패를 유발하는지 여부를 팀에 알려줌

지속적 통합은 어떻게 동작하는가?

- **지속적 테스트와 보안 자동화**
  - 최선의 방법은 개발자가 로컬 환경에서 회귀 테스트 전체 또는 일부를 실행할 수 있도록 하고, 이를 의무화하는
  - 이렇게 하면 개발자는 코드 변경이 회귀 테스트를 통과한 이후에만 버전 제어에 코드를 커밋하게 됨
  - 그러나 회귀 테스트는 시작일 뿐, 데브옵스팀은 성능, API, 브라우저 및 디바이스 테스트도 자동화함
    - 팀은 시프트-레프트(shift-left) 테스트를 위해 CI/CD 파이프라인에 정적 코드 분석과 보안 테스트도 내장할 수 있음
    - 또한 애자일 팀은 서비스 가상화를 사용하여 서드파티 API, SaaS, 및 기타 자체 제어 범위를 벗어난 시스템과의 상호작용도 테스트할 수 있음
    - 핵심은 명령줄, 웹훅 또는 웹 서비스를 통해 이러한 테스트를 트리거하고 성공 또는 실패 응답을 받을 수 있다는 것
  - 지속적 테스트는 CI/CD 파이프라인이 테스트 자동화를 통합한다는 것을 의미함
    - 일부 단위 및 기능 테스트는 지속적 통합 프로세스 전에, 또는 도중에 문제에 플래그를 지정
    - 성능 및 보안 테스트와 같이 전체 제공 환경이 필요한 테스트는 지속적 제공 내에 통합되어 빌드가 타겟 환경에 전달된 이후 수행되는 경우가 많음

성공적인 지속적 통합

- 성공적인 CI란?
  - 개발자가 애플리케이션에 적용한 변경 사항들이 병합된 후
  - 이러한 변경 사항이 애플리케이션을 손상시키지 않도록 자동으로 애플리케이션을 빌드하고 다양한 수준의 자동화된 테스트(일반적으로 단위 테스트와 통합 테스트)를 실행하여 해당 변경 사항을 검증하는 것
  - 클래스와 기능에서부터 전체 애플리케이션을 구성하는 다양한 모듈에 이르기까지 모든 것을 테스트함
  - 자동화된 테스트를 통해 새 코드와 기존 코드 간 충돌이 발견되는 경우 CI를 적용하면 해당 버그를 빠르게, 자주 수정하기가 더 용이해짐

지속적 제공/배포

지속적 제공/배포

- 지속적 제공/배포
  - 코드 변경 사항의 통합, 테스트, 제공을 나타내는 프로세스
  - 지속적 통합(CI)이 끝나는 지점부터 시작
  - CI에서 빌드와 단위 및 통합 테스트를 자동화한 다음 검증된 코드를 저장소(Repository)로 배포(Release)하는 것을 자동화함
    - 따라서 효과적인 지속적 제공 프로세스를 마련하려면 CI가 개발 파이프라인에 이미 구축되어 있어야 함
  - 프로덕션, 개발, 테스트 환경을 포함해 선택한 환경으로 애플리케이션을 제공하는 과정을 자동화함
    - CD는 이런 환경으로 코드 변경을 적용(push)하기 위한 자동화된 방법을 가리킴

지속적 제공/배포

- **CD (Continuous Delivery / Deployment)**
  - 지속적 제공(Continuous Delivery) 또는 지속적 배포(Continuous Deployment)를 의미하며 이 두 용어는 상호 교환하여 사용
  - 두 가지 모두 파이프라인의 추가 단계를 자동화하는 것이 목적이지만 자동화가 얼마나 많이 진행되고 있는지를 나타내기 위해 별도로 사용될 수도 있음
  - 지속적 제공과 지속적 배포 중 어느 것을 선택할지는 개발 팀과 운영 팀의 위험 허용 범위와 구체적인 요구 사항에 따라 달라짐

- **지속적 제공과 지속적 배포의 차이**
  - 지속적 제공: 자동 프로덕션 배포 기능이 없음
  - 지속적 배포: 업데이트를 프로덕션 환경에 자동으로 Release(배포)

지속적 제공/배포

- 지속적 제공(또는 지속적 전달)
  - 개발자의 애플리케이션 변경 사항이 자동으로 버그 테스트를 거치고 저장소(예: GitHub, 컨테이너 레지스트리)로 업로드 된다는 것을 의미
    - 코드 변경 사항의 병합부터 프로덕션 레디 빌드의 제공에 이르기까지 모든 단계에 테스트 자동화와 코드 릴리스 자동화가 수반됨
  - 이 프로세스가 종료되면 운영 팀은 변경 사항을 프로덕션 환경으로 신속하게 배포할 수 있음
    - 그 결과 개발 팀과 비즈니스 팀 간 가시성 및 의사 소통 부족 문제가 해결될 수 있음
  - 이를 위한 지속적 제공의 목표
    - 언제나 프로덕션 환경으로 배포할 준비가 되어 있는 코드베이스를 갖추고 새로운 코드를 배포하는 데 필요한 노력을 최소화하는 것

지속적 제공(전달)은 어떻게 동작하는가?

- 지속적 제공 파이프라인 내의 단계
  - 지속적 제공은 애플리케이션을 하나 이상의 제공 환경으로 푸시하는 자동화
  - 개발팀은 일반적으로 테스트 및 리뷰를 위해 애플리케이션 변경을 준비하는 여러 환경을 준비함
    - 데브옵스 엔지니어는 주로 다음과 같은 CI/CD 툴을 사용해서 이와 같은 단계를 자동화하고 보고함
      - 젠킨스(Jenkins), 서클CI(CircleCI), AWS 코드빌드(CodeBuild), 애저 데브옵스(DevOps), 아틀라시안 뱀부(Bamboo), 아르고 CD(Argo CD), 버디(Buddy), 드론(Drone), 트래비스 CI(Travis CI)

지속적 제공(전달)은 어떻게 동작하는가?

- 지속적 제공 파이프라인에는 빌드, 테스트, 배포 단계가 있으며 다음 활동은 다양한 단계에 포함될 수 있음
  - 버전 제어에서 코드를 풀링해서 빌드 실행
  - 단계 게이트에서 자동화된 보안, 품질 및 규정 준수 확인과 필요한 경우 이를 뒷받침하는 승인 활성화.
  - 코드로 자동화된 필요 인프라 단계를 실행해 클라우드 인프라 구축 또는 해체
  - 타깃 컴퓨팅 환경으로 코드 이동
  - 환경 변수를 관리하고 타겟 환경에 맞게 구성
  - 애플리케이션 구성요소를 웹 서버, API, 데이터베이스 서비스와 같은 적절한 서비스로 푸시
  - 새 코드 푸시에 필요한 서비스 엔드포인트를 호출하거나 서비스를 재시작하기 위해 필요한 단계 실행
  - 지속적 테스트 실행과 테스트 실패 시 롤백
  - 제공 상태에 대한 로그 데이터 및 알림 제공
  - 구성 관리 데이터베이스를 업데이트하고 IT 서비스 관리 워크플로우에 완료된 배포에 대한 알림 전송

지속적 제공(전달)은 어떻게 동작하는가?

- 더 세밀한 지속적 제공 파이프라인에는 데이터 동기화, 정보 리소스 아카이빙, 애플리케이션 및 라이브러리 패치와 같은 부가적인 단계도 포함될 수 있음

지속적 제공/배포

- 지속적 배포
  - 성숙한 CI/CD 파이프라인의 최종 단계
  - 지속적 제공의 확장으로, 개발자의 변경 사항을 저장소에서 프로덕션으로 배포(Release)하는 것을 자동화하여 고객이 사용할 수 있도록 하는 것을 말함

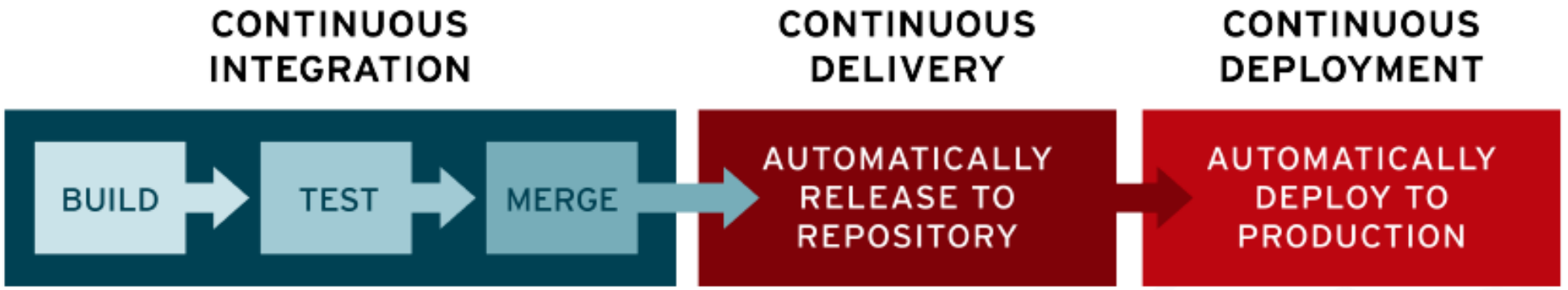
지속적 배포는 어떻게 동작하는가?

- 지속적 배포에서는
  - 애플리케이션 변경이 CI/CD 파이프라인을 거쳐 실행
  - 통과한 빌드는 프로덕션 환경에 바로 배포
  - 견고한 CI/CD 파이프라인을 갖춘 성숙한 데브옵스팀은 지속적 배포도 구현할 수 있음
  - 지속적 배포를 사용하여 프로덕션으로 배포하는 팀
    - 다운타임을 최소화하고 배포 위험을 관리하기 위해 다양한 방법을 사용함
      - 예: 이전 소프트웨어 버전에서 새 버전으로 트래픽 사용의 전환을 조율하는 카나리 배포 구성하기
  - 지속적 배포가 모든 비즈니스 애플리케이션에서 최적의 방법은 아니지만, 지속적 배포를 실천하는 팀 중에서는 매일, 심지어 매시간 프로덕션으로 배포하는 팀도 있음

CI / CD 파이프라인

CI/CD 파이프라인

- CI/CD 파이프라인



  - 위와 같이 지속적 통합과 지속적 제공/배포의 과정이 연결된 흐름을 "CI/CD 파이프라인"이라 부름
  - 소프트웨어 개발 라이프사이클에서 새 버전의 소프트웨어를 제공하기 위해 수행해야 할 일련의 단계를 자동화하는 전략 (코드를 빌드, 테스트, 배포하는 과정을 거쳐 소프트웨어 개발을 추진하는 프로세스)
  - 개발, 테스트, 프로덕션, 모니터링 단계에서 소프트웨어 제공을 개선하는데 중점을 둠
  - 자동화된 CI/CD 파이프라인을 사용하여 빠르고 안전하게 고품질의 코드를 개발할 수 있음

CI/CD 파이프라인

- 개발자와 IT 운영팀이 협업해 소프트웨어를 빠르게 제공하고, 안정성과 품질을 유지하는데 중요한 역할
  - 일반적으로 개발 팀과 운영 팀이 DevOps 또는 SRE(사이트 신뢰성 엔지니어링)를 통해 애자일 방식으로 협력함으로써 이런 흐름을 지원함
    - SRE(사이트 신뢰성 엔지니어링)
      - 소프트웨어 엔지니어링을 사용하여 시스템 관리자가 수작업으로 수행했던 IT 운영작업 (예: 프로덕션 시스템 관리, 변경 관리, 인시던트(Incident, 사건) 대응, 긴급 상황 대응 등을 포함)을 자동화하는 기술
    - 애자일(Agile)
      - 소프트웨어 개발이나 프로젝트 관리에서 사용되는 유연하고 반복적인 업무를 빠르게 처리하는 방식
- 프로세스를 자동화함으로써 인적 오류를 최소화하고 소프트웨어 출시 방식에 일관된 프로세스를 유지하는 것을 목표로 함

CI/CD 파이프라인의 개발

- CI/CD 파이프라인 개발
  - 애플리케이션을 수시로 개선하고 안정적인 제공 프로세스가 필요한 기업이 요구하는 표준적인 개발 방식
  - CI/CD 파이프라인을 구축하면
    - 팀은 애플리케이션을 향상시키는 데 더 초점을 두면서
    - 이를 다양한 환경으로 제공하기 위한 세부 사항에 대해 신경을 덜 쓸 수 있음
  - DevOps팀이 CI/CD를 시작하기 위해서는
    - 기술, 작업 방식 및 우선순위에 대한 협업이 필요
    - 팀은 비즈니스와 기술에 대한 적절한 접근 방식을 합의
    - 파이프라인이 구축된 이후에는 CI/CD 방식을 일관적으로 따라야 함

CI/CD 파이프라인 자동화 개발 과정

1. 소스 코드 관리 (Source Code Management, SCM)

- 개념: 소스 코드를 버전 관리하고 협업하는 단계
- 기술/도구: Git, GitHub, GitLab, Bitbucket 등
- 주의사항: 브랜치 전략, 커밋 메시지 규칙을 준수해야 함

2. 지속적인 통합 (Continuous Integration, CI)

- 개념: 코드 변경 사항을 자주 통합하고 빌드, 테스트
- 기술/도구: Jenkins, GitLab CI, Travis CI, CircleCI 등
- 주의사항: 자동화된 빌드와 테스트를 통해 코드 품질을 유지해야 함

자동 빌드(Automated Build), 자동 테스트(Automated Testing)는 CI에 포함하기도 하고 CI 이전 단계에서 별도로 진행하기도 함

자동 빌드(Automated Build)
- 코드가 저장소에 통합될 때마다 자동으로 빌드 과정 실행
- 코드가 올바르게 컴파일 되었는지, 기본적인 품질 기준을 만족하는지 검증

자동 테스트(Automated Testing)
- 빌드가 완료된 후 다양한 자동화 된 테스트(단위/통합/기능 테스트)를 수행
- 코드의 안정성과 기능성 확인
- 버그의 조기 발견

CI/CD 파이프라인 자동화 개발 과정

3. 지속적인 배포 (Continuous Deployment, CD)

- 개념: 코드 변경 사항이 자동으로 배포되는 단계
- 기술/도구: Kubernetes, Docker, Tekton, ArgoCD 등
- 주의사항: 안정성과 보안을 고려하여 배포를 자동화해야 함

4. 환경 설정 및 프로덕션 배포

- 개념: 프로덕션 환경에 코드를 배포함
- 기술/도구: Kubernetes 클러스터, Helm, Istio 등
- 주의사항: 롤백 전략, 모니터링 설정을 고려해야 함

CI/CD 파이프라인 자동화 개발 과정

5. 모니터링 및 로깅

- 개념: 애플리케이션의 상태를 모니터링하고 로그를 수집함
- 기술/도구: Prometheus, Grafana, ELK 스택 등
- 주의사항: 장애 대응 및 성능 최적화를 위해 모니터링을 설정해야 함

CI/CD 파이프라인 개발 시 사용하는 도구들

- CI/CD 관련 도구의 역할
  - 각 제공 단계에서 패키징 해야 하는 환경 별 매개변수의 저장에 도움이 됨
  - 재시작해야 하는 웹서버, DB 및 기타 서비스를 대상으로 필요한 서비스 호출 수행
  - 배포 후에 요구되는 다른 절차의 실행
  - 지속적 테스트 지원(CI/CD 파이프라인의 목표는 양질의 코드와 애플리케이션의 제공이므로)
    - 자동화된 일련의 회귀(반복되는 작업)
    - 성능 및 기타 테스트

CI/CD 파이프라인 개발 시 사용하는 도구들

- 파이프라인의 개발 과정에 포함되는 툴의 종류
  - 버전 관리 시스템 (Version Control System)
    - 코드 변경 사항의 추적 및 관리에 사용됨
    - Git, SVN 등 (최근에는 Git 개발 환경이 인기가 있음)
  - CI 서버 (CI Server)
    - 코드가 저장소에 통합될 때 자동으로 빌드와 테스트를 수행함
    - Jenkins, Travis CI, CircleCI 등
    - 각각의 툴은 고유한 특성과 장점을 가지므로 프로젝트의 요구사항에 따라 적절한 선택이 필요함

CI/CD 파이프라인 개발 시 사용하는 도구들

- 자동화된 테스트 도구 (Automated Testing Tools)
  - 코드의 품질을 보장하기 위해 다양한 유형의 테스트를 자동으로 실행함
  - Selenium, Junit, TestNG 등
  - 테스트 케이스의 작성과 실행을 자동화하여 버그를 빠르게 발견하고 수정하는데 도움을 줌

- 배포 자동화 도구 (Deployment Automation Tools)
  - 코드가 테스트를 통과하면 이를 실제 운영 환경에 자동으로 배포함
  - Ansible, Chef, Puppet 등
  - 서버 구성과 애플리케이션 배포를 자동화하여 시간과 노력을 절약해 줌

CI/CD 파이프라인 개발 시 사용하는 도구들

- 컨테이너화 및 오케스트레이션 (Containerization and Orchestration)
  - 애플리케이션을 컨테이너로 패키징하고 관리하는데 사용됨
  - Docker, Kubernetes 등
  - 애플리케이션의 일관된 배포와 확장성 보장에 중요한 역할을 담당

- 그 외에도 다양한 코드 컴파일, 유닛 테스트, 코드 분석, 보안, 바이너리 생성 등의 도구가 있음
- 컨테이너화된 환경에서는 하이브리드 클라우드 전반에 배포할 컨테이너 이미지에 코드를 패키징하는 경우도 이러한 파이프라인에 포함됨

CI/CD 파이프라인 개발 시 사용하는 도구들

- 일반적인 CI/CD 툴
  - CI/CD 툴은 팀이 개발, 배포, 테스트를 자동화하도록 지원함
  - 통합(CI) 측면을 다루는 툴, 개발 및 배포(CD)를 관리하는 툴, 지속적인 테스트 또는 관련 기능에 특화된 툴 등 다양한 도구가 존재함
  - 다양한 벤더가 제공하는 관리형 CI/CD 툴
    - 주요 퍼블릭 클라우드 공급업체는 모두 GitLab, CircleCI, Travis CI, Atlassian Bamboo 등과 함께 CI/CD 솔루션을 제공함
    - 또한 DevOps의 기본 툴은 CI/CD 프로세스에 속해 있을 가능성이 높음
      - 구성 자동화(예: Ansible, Chef, Puppet), 컨테이너 런타임(예: Docker, rkt, cri-o), 컨테이너 오케스트레이션(쿠버네티스)을 위한 툴은 엄밀하게는 CI/CD 툴이 아니지만 많은 CI/CD 워크플로우에 표시됨

CI/CD 파이프라인 개발 시 사용하는 도구들

- 선호하는 애플리케이션 개발 전략과 클라우드 제공업체에 따라 CI/CD를 구현하는 다양한 방법 존재
  - Red Hat®, OpenShift®, Service on AWS
    - Tekton과 OpenShift Pipelines와 같은 여러 옵션이 제공되어 자체 CI/CD 워크플로우를 더 쉽게 만들 수 있음
    - Red Hat, OpenShift의 경우, 기업은 다수의 온프레미스 및 클라우드 플랫폼에서 애플리케이션을 자동으로 빌드, 테스트, 배포할 수 있음

CI/CD 파이프라인 개발 시 사용하는 도구들

- 도구별 간략한 소개
  - Tekton Pipelines: 표준 클라우드 네이티브 CI/CD 경험과 컨테이너를 제공하는 쿠버네티스 플랫폼을 위한 CI/CD 프레임워크
  - Jenkins: 단순 CI 서버에서 완전한 CD 허브까지 모든 것을 처리하도록 설계된 툴
  - Spinnaker: 멀티클라우드 환경을 위해 구축된 CD 플랫폼
  - GoCD: 모델링 및 시각화에 중점을 둔 CI/CD 서버
  - Concourse: "지속적인 오픈소스 작업 툴"
  - Screwdriver: CD용으로 설계된 빌드 플랫폼
  - 기타…

CI/CD 파이프라인 개발 시 사용하는 도구들

- CI/CD 툴과 플러그인
  - CI/CD 툴은 일반적으로 플러그인 마켓플레이스를 지원
    - 예: 젠킨스에는 서드파티 플랫폼과 사용자 인터페이스, 관리, 소스 코드 관리 및 빌드 관리와의 통합을 지원하는 1,800개 이상의 플러그인이 있음
  - 개발팀은 CI/CD 툴을 선택하고 나면 모든 환경 변수가 애플리케이션 외부에서 구성되도록 해야 함
    - 변수를 설정하고 암호 및 계정 키와 같은 변수를 마스킹하고 타겟 환경 배포 시 구성할 수 있음
  - 지속적 제공 툴은 대시보드 및 보고 기능도 제공함
    - 데브옵스팀은 관찰 가능한 CI/CD 파이프라인을 구현해 이 기능을 더 강화할 수 있음
      - 빌드 또는 제공이 실패할 경우 개발자에게 알림이 전달
      - 대시보드와 보고 기능은 버전 제어 및 애자일 툴과 통합되어 어떤 코드 변경과 사용자 스토리가 빌드를 구성하는지를 확인하는 것을 도와줌

CI / CD와 DevOps

CI/CD와 DevOps

- CI/CD는…
  - 개발 팀과 운영 팀 간 협업 촉진을 목표로 하는 DevOps 방법론의 필수적인 부분
  - CI/CD와 DevOps는 모두 코드 통합 프로세스를 자동화하는 데 중점을 두어
  - 사용자에게 가치를 제공할 수 있는 프로덕션 환경에서
  - 아이디어(예: 새 기능, 개선 사항 요청, 버그 수정)가 개발에서 배포 단계로 이동하는 프로세스를 가속화

DevOps

- DevOps(Development Operations)란?
  - 개발 (Dev)과 운영 (Ops)을 결합하여 애플리케이션의 계획, 개발, 제공 및 운영 과정에서 사람, 프로세스 및 기술을 통합하는 방법론

- DevOps의 적용을 통해 얻을 수 있는 이점
  - 출시 시간 가속화: 팀은 지속적인 배포를 통해 제품을 빠르게 출시할 수 있음
  - 시장 및 경쟁에 적응: DevOps 문화는 고객 중심으로 민첩하게 작업하고 경쟁력을 높임
  - 시스템 안정성 및 안정성 유지 관리: 지속적인 개선 사례를 통해 제품과 서비스의 안정성을 높일 수 있음
  - 평균 복구 시간 개선: 소프트웨어 오류와 보안 위반을 관리하려면 평균 복구 시간을 측정하고 개선해야 함

DevOps의 목적

- DevOps의 목적
  - DevOps는 단순히 도구와 관행을 넘어서
  - 사고 방식과 문화적 변화를 요구하며
  - 개발자와 운영 팀이 협업하여
  - 사용자 요구 사항을 이해하고 더 나은 제품을 제공할 수 있도록 함

DevOps의 중요성이 커지는 이유

- DevOps의 중요성이 커지는 이유
  - 기업은 여러 가지 서비스를 제공하고, 사용자는 애플리케이션을 통해서 서비스를 소비하는 것이 현대의 일반적인 패턴
  - DevOps 엔지니어가 하는 일의 공통점은 현대의 모든 서비스가 겪을 수 밖에 없는 문제를 해결하기 위한 시도라는 것
  - 그 과정에는 해결해야 하는 문제가 존재하고, DevOps 기반의 기술이 이 문제를 해결하는 데 도움이 되기 때문에 시간이 갈수록 수요가 증가하고 있음

DevOps의 중요성이 커지는 이유

- 각 영역별, 단계별 핵심
  - 일반적으로 애플리케이션을 통한 서비스 제공 비즈니스는 개발, 배포, 운영이라는 3가지의 과정을 반드시 거침
  - 각 과정에서 각 영역에 대한 핵심
    - 목적 → 목적 달성을 위한 과제 → 각 과정에서 겪는 문제 → 문제 해결을 위한 솔루션

DevOps의 중요성이 커지는 이유

- **개발, 배포, 운영의 목적**
  - 개발: 필요한 것을 만들자
  - 배포: 많은 사람에게 전달하자
  - 운영: 지속 가능하게 관리하자

- **개발, 배포, 운영의 목적을 달성하기 위한 과제**
  - 개발: 기능 추가, 테스트
  - 배포: 인프라 구축
  - 운영: 피드백 확보, 분석 및 관리

DevOps의 중요성이 커지는 이유

- **개발, 배포, 운영 과제에서 겪는 문제**
  - 개발 : 반복되는 업무와 휴먼에러
  - 배포 : 인프라 확장 축소에 따른 비용 문제
  - 운영 : 데이터 관리, 서비스 중단 상황에 대한 대처

- **개발, 배포, 운영의 문제를 해결하기 위한 솔루션**
  - 개발 : CI/CD 파이프라인, 자동화
  - 배포 : 탄력적인 인프라 설계
  - 운영 : 데이터 모니터링, 무중단 서비스

CI/CD와 DevOps

- DevOps에서의 CI/CD는…
  - 협업을 중시하는 DevOps 프레임워크에서 보안은 초기 단계부터 통합되어 공동의 책임이 됨

    → DevSecOps 등장
    - DevOps 이니셔티브에 보안 기반을 구축해야 할 필요성을 강조한 용어
    - DevSecOps(개발, 보안, 운영)
      - 전체 IT 라이프사이클에 걸쳐 보안을 통합하는 방식으로 책임을 공유하는 문화, 자동화 및 플랫폼 설계에 대한 접근 방식
      - DevSecOps의 핵심 구성 요소는 보안 CI/CD 파이프라인의 도입

CI/CD의 보안

- CI/CD 보안
  - 자동화된 검사 및 테스트로 코드 파이프라인을 보호하여 소프트웨어 제공 시 취약점을 방지하는 데 사용
  - 보안을 기업의 파이프라인에 통합하는 것은 코드를 공격에서 보호하고 데이터 누출을 방지하며 정책을 준수하고 품질 보증을 보장하는 데 도움이 됨
  - 적절한 보안이 없으면?
    - 개발 및 배포의 빠른 특성으로 인해 파이프라인이 다음과 같은 위험에 노출될 수 있음
      - 민감한 데이터가 외부 소스에 노출됨
      - 안전하지 않은 코드 또는 타사 구성 요소 사용
      - 소스 코드 리포지토리 또는 빌드 툴에 대한 무단 액세스
  - 소프트웨어 개발 주기 전반에서 취약점을 식별하고 완화하면 코드 변경 사항이 프로덕션에 배포되기 전에 철저한 테스트를 거쳐 보안 표준을 준수하도록 보장할 수 있음

CI / CD의 평가

CI/CD 성공을 평가하는 DevOps KPI

- **CI/CD 파이프라인 구현 효과의 측정**
  - **DevOps 핵심성과지표(KPI, Key Performance Indicator)로 측정 가능**
    - Google에서 개발한 DORA(DevOps Research and Assessment) 지표를 기준으로 KPI를 측정함
    - 배포빈도, 변경을 위한 리드 타임, 평균 서비스 복원 시간, 변경 실패율 등을 주요 평가 항목으로 삼고, '엘리트-하이-미디엄-로우' 등의 순으로 점수를 매김

CI/CD 성공을 평가하는 DevOps KPI

- **DORA (DevOps Research and Assessment)**
  - 리드 타임 (Lead Time): 코드 변경이 트렁크 브랜치에 커밋된 후 배포 가능 상태가 되기까지 걸리는 시간. 필수 사전 릴리스 테스트를 모두 통과한 코드가 얼마나 빨리 배포되는지를 측정
  - 변경 실패율 (Change Failure Rate): 프로덕션 후 긴급 수정이 필요한 코드 변경에 대한 비율. 테스트에서 발견되어 코드 배포 전에 수정된 오류는 측정하지 않음
  - 배포 빈도 (Deployment Frequency): 새로운 코드를 프로덕션에 배포하는 빈도를 평가. DevOps의 성공을 이해하는 데 매우 중요한 지표
  - 평균 복구 시간 (MTTR, Mean Time To Recovery): 부분적인 서비스 중단 또는 전체 장애로부터 복구하는 데 걸리는 시간. 이는 중단이 최근 배포로 인한 것인지 또는 격리된 시스템 장애로 인한 것인지 여부와 관계없이 추적해야 하는 중요한 메트릭

CI/CD 성공을 평가하는 DevOps KPI

- 지속적 테스트와 함께 CI/CD를 구현하면 배포 빈도, 변경 리드 시간, 평균 복구 시간(MTTR)과 같은 지표가 개선되는 경우가 많음
- 그러나 CI/CD는 이와 같은 개선을 이끌기 위한 하나의 프로세스일 뿐이며 배포 빈도를 개선하기 위한 다른 전제 조건도 있음

다양한 환경에서의 CI/CD 접근

쿠버네티스 및 서버리스 아키텍처에서의 CI/CD

- 클라우드 환경에서의 CI/CD 파이프라인 운영
  - 클라우드 환경에서 CI/CD 파이프라인을 운영하는 많은 팀들은
    - 도커와 같은 컨테이너, 쿠버네티스와 같은 오케스트레이션 시스템도 사용
    - 컨테이너는 이식 가능한 표준적 방식으로 애플리케이션을 패키징하고 배포할 수 있게 해 줌
    - 컨테이너를 사용하면 변화하는 워크로드에 따라 환경을 확장하거나 해체하기 쉬움
  - 컨테이너, 코드형 인프라(IaC), CI/CD 파이프라인을 함께 사용하는 접근법
    - 젠킨스와 함께 쿠버네티스 사용하기
    - 애저 DevOps와 함께 쿠버네티스 사용하기 등
    - 무료 학습 자료도 많음

쿠버네티스 및 서버리스 아키텍처에서의 CI/CD

- **서버리스 아키텍처를 사용해서 애플리케이션을 배포 및 확장하는 방법**
  - 서버리스 환경에서는 클라우드 서비스 업체가 인프라를 관리하고, 애플리케이션은 구성을 기반으로 필요에 따라 리소스를 소비함
  - 예시
    - AWS에서 서버리스 애플리케이션은 람다 함수로 실행되고 플러그인을 사용해 젠킨스 CI/CD 파이프라인에 배포를 통합할 수 있음
    - 애저 서버리스 및 GPS 서버리스 컴퓨팅도 비슷한 서비스

차세대 CI/CD 애플리케이션

- CI/CD 파이프라인 개발 및 관리를 위한 고급 영역
  - 참고할 만한 자료
    - MLOps : 머신러닝 모델의 IaC 및 CI/CD. 인프라, 통합, 그리고 학습 및 프로덕션 환경으로의 배포를 지원함
    - 합성 데이터 생성(Synthetic data generation) 기법
      - 테스트 자동화 엔지니어가 API를 테스트하는 용도, 데이터 과학자가 모델을 교육시키는 용도로 사용하는 데이터 집합을 머신러닝을 사용해서 생성함
    - AIOps 플랫폼(또는 ITOps의 머신러닝 및 자동화)
      - 관찰 가능성 데이터 집계, 여러 소스의 알림을 인시던트로 연계. 자동화는 필요에 따라 CI/CD 배포와 롤백을 트리거
    - 마이크로서비스를 다루는 팀
      - 재사용 가능한 파이프라인을 생성해 애저 및 AWS에서 개발, 검토 옵션을 지원, 확장
      - 엔지니어는 네트워크 구성, 임베디드 시스템, 데이터베이스 변경, IoT, AR/VR과 같은 다른 영역에서 CI/CD를 사용

결론

- 지속적 통합
  - 소프트웨어 빌드를 패키징 및 테스트하고 변경이 단위 테스트를 통과하지 못할 경우 개발자에게 알린다.
- 지속적 제공
  - 애플리케이션과 서비스 및 기타 기술 배포를 런타임 인프라에 제공하고 경우에 따라 부가적인 테스트를 실행하기도 하는 자동화다.
- CI/CD 파이프라인 개발
  - 애플리케이션을 수시로 개선하고 안정적인 제공 프로세스가 필요한 기업에서 표준적인 개발 방식
  - CI/CD 파이프라인을 구축하면 팀은 애플리케이션을 향상시키는 데 더 초점을 두면서 이를 다양한 환경으로 제공하기 위한 세부 사항에 대해 신경을 덜 쓸 수 있음

결론

- DevOps팀이 CI/CD를 시작하기 위해서는
  - 기술, 작업 방식 및 우선순위에 대한 협업이 필요
  - 팀은 비즈니스와 기술에 대한 적절한 접근 방식을 합의
  - 파이프라인이 구축된 이후에는 CI/CD 방식을 일관적으로 따라야 함