CI/CD

CI/CD

- 개발 워크플로
  - ✓ 로컬에서 단위 테스트 실행
    - 중앙 리포지토리에서 최신 코드를 로컬로 가져온 후 요구 사항을 구현
    - 요구 사항을 구현할 때는 TDD(테스트 주도 개발) 방식을 많이 사용
    - 구현된 코드는 단위 테스트를 모두 통과 할 때 까지 반복적으로 수정
  - ✓ 중앙 리포지토리로 코드 푸시 및 병합
    - 개발자가 로컬에서 기능 구현을 완료한 후 중앙 리포지토리로 코드를 푸시하면 메인 브랜치에 병합
  - ✓ 병합 후 코드 컴파일
    - 구현을 완료한 코드가 메인 브랜치에 병합되면 코드 컴파일을 수행
    - 새로 병합된 코드로 인해 기존에 없던 컴파일 오류가 발생할 수 있음
  - ✓ 컴파일된 코드에서 테스트 실행
    - 병합된 코드가 컴파일이 완료된 후에는 단위 테스트와 통합 테스트를 실행해 회귀 테스트를 수행해서 결함이 있는지를 확인
    - 그 외에도 정적 분석(코딩 표준 준수 및 불필요한 코드의 존재 여부를 확인하는 과정) 같은 작업이 추가되기도 함
  - ✓ 아티팩트 배포
    - 병합된 코드의 단계별 품질 점검을 끝나면 모든 코드를 패키징하고 최종 사용자가 사용할 수 있도록 서버에 배포
    - 아티팩트는 보통 .war나 .jar 파일 처럼 하나의 파일로 압축된 형식

CI/CD

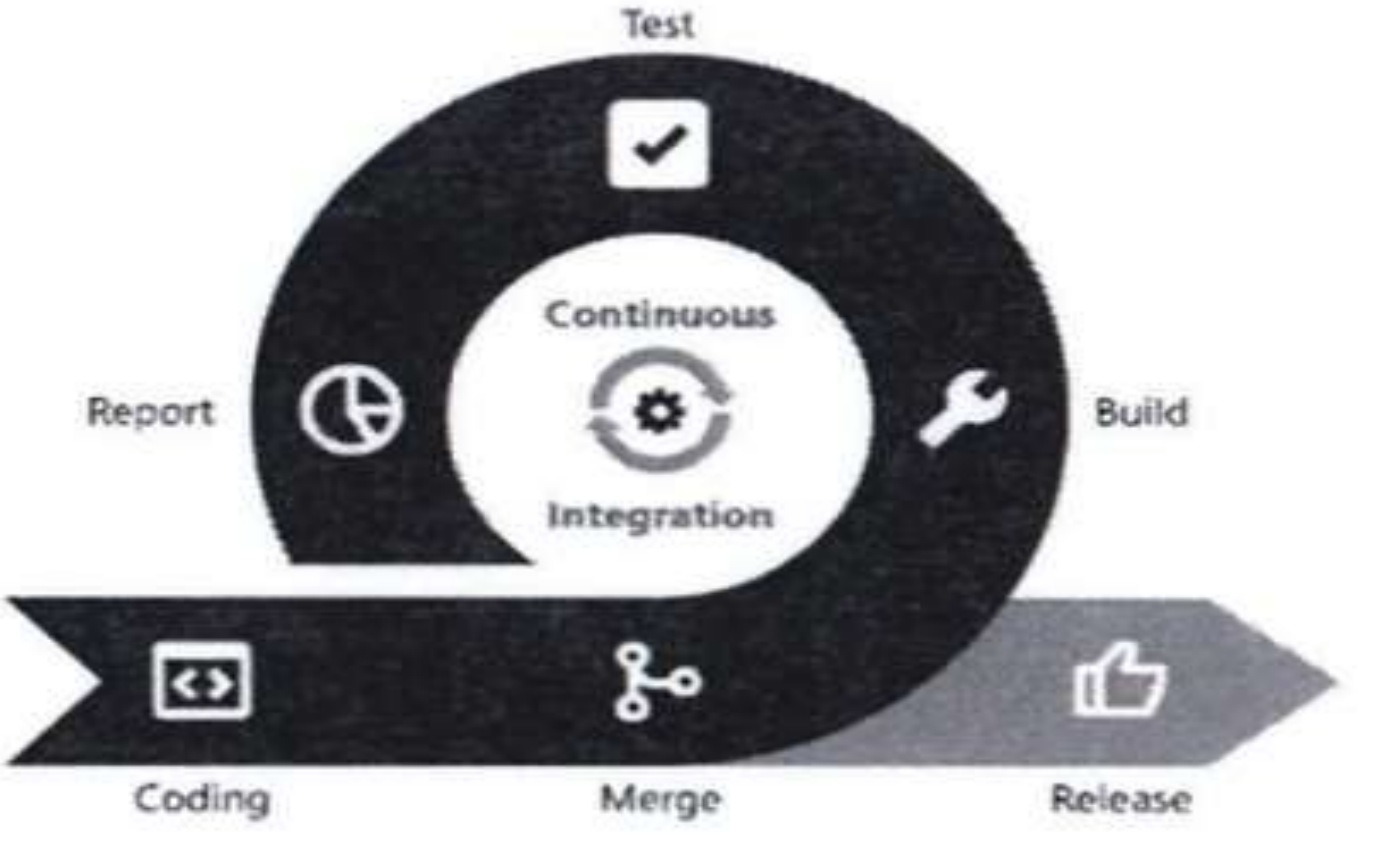
- 개요
  - ✓ CI/CD의 기본 개념은 지속적인 통합(Continuous Integration), 지속적인 서비스 제공 (Continuous Delivery), 지속적인 배포 (Continuous Deployment)
  - ✓ CI/CD는 애플리케이션 개발 단계를 자동화하여 애플리케이션을 보다 짧은 주기로 고객에게 제공하는 방법
  - ✓ CI/CD는 새로운 코드 통합으로 인해 개발 및 운영 팀에 발생하는 문제(인테그레이션 헬 – integration hell)를 해결하기 위한 방법
  - ✓ CI/CD는 지속적 통합 및 지속적 제공의 구축 사례만을 지칭할 때도 있고 지속적 통합, 지속적 제공, 지속적 배포 라는 3가지 구축 사례 모두를 의미하는 것일 수도 있음
  - ✓ 지속적인 서비스 제공은 때로 지속적인 배포의 과정까지 포함하는 방식으로 사용되기도 함
  - ✓ 결과적으로 CI/CD는 파이프라인으로 표현되는 실제 프로세스를 의미하고 애플리케이션 개발에 지속적인 자동화 및 지속적인 모니터링을 추가하는 것을 의미
  - ✓ 이 용어는 사례별로 CI/CD 파이프라인에 구현된 자동화 수준 정도에 따라 그 의미가 달라지는데 대부분의 기업에서는 CI를 먼저 추가한 다음 클라우드 네이티브 애플리케이션의 일부로서 배포 및 개발 자동화를 구현해 나감

CI/CD

- 지속적인 통합(Continuous Integration)
  - ✓ 지속적인 통합(Continuous Integration)은 개발 팀이 작은 변경 사항을 구현하고 코드를 버전 제어 레포지토리에 자주 체크인하도록 하는 코딩 철학이자 일련의 관행
  - ✓ 대부분의 최신 애플리케이션은 다양한 플랫폼과 도구에서 코드를 개발해야 하기 때문에 팀은 변경 사항을 통합하고 검증하기 위한 메커니즘이 필요
  - ✓ 기술 목표는 애플리케이션을 빌드, 패키지 및 테스트하는 일관되고 자동화된 방법을 설정하는 것
  - ✓ 통합 프로세스의 일관성을 유지하면 팀이 코드 변경을 더 자주 커밋할 가능성이 높아져 공동 작업과 소프트웨어 품질이 향상
  - ✓ 소프트웨어 공학에서 지속적 통합(continuous integration, CI)은 지속적으로 품질 관리(Quality Control)를 적용하는 프로세스를 실행하는 것
  - ✓ 작은 단위의 작업, 빈번한 적용. 지속적인 통합은 모든 개발을 완료한 뒤에 품질 관리를 적용하는 고전적인 방법을 대체하는 방법으로서 소프트웨어의 질적 향상과 소프트웨어를 배포하는데 걸리는 시간을 줄이는데 초점이 맞추어짐
  - ✓ 지속적인 통합이 제대로 구현되면 애플리케이션 코드의 새로운 변경 사항이 정기적으로 빌드 및 테스트를 거쳐 공유 레포지토리에 병합되므로 여러 명의 개발자가 동시에 애플리케이션 개발과 관련된 코드 작업을 할 경우 서로 충돌하는 문제를 해결할 수 있음

CI/CD

❖ 지속적인 통합(Continuous Integration)



CI/CD

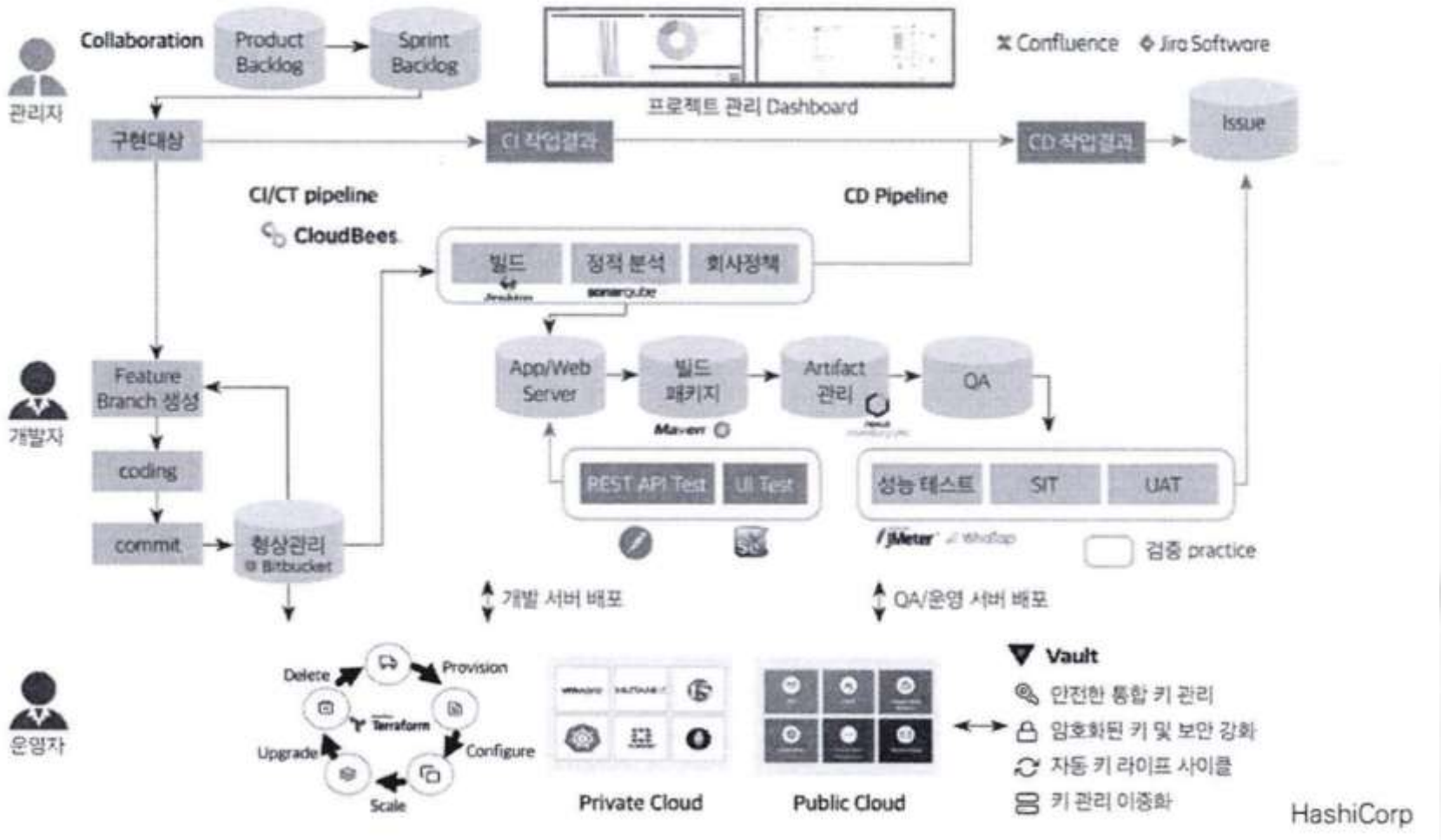
- Continuous Delivery and Continuous Deploy
  - ✓ 지속적인 제공(Continuous Delivery)
    - 개발자들이 애플리케이션에 적용한 변경 사항이 버그 테스트를 거쳐 레포지토리 (GitHub 또는 컨테이너 레지스트리)에 자동으로 업로드 되는 것
    - 운영 팀은 이 레포지토리에서 애플리케이션을 실시간 프로덕션 환경으로 배포
    - 개발 팀과 비즈니스 팀 간의 가시성과 커뮤니케이션 부족 문제를 해결
    - 지속적인 제공은 최소한의 노력으로 새로운 코드를 배포하는 것

CI/CD

- Continuous Delivery and Continuous Deploy
  - ✓ 지속적인 배포(Continuous Deployment)
    - 다른 의미의 CD(Continuous Deployment)란 개발자의 변경 사항을 레포지토리에서 고객이 사용 가능한 프로덕션 환경까지 자동으로 릴리스하는 과정
    - 애플리케이션 제공 속도를 저해하는 수동 프로세스로 인한 운영 팀의 프로세스 과부하 문제를 해결
    - 지속적인 배포는 파이프라인의 다음 단계를 자동화함으로써 지속적인 제공이 가진 장점을 활용

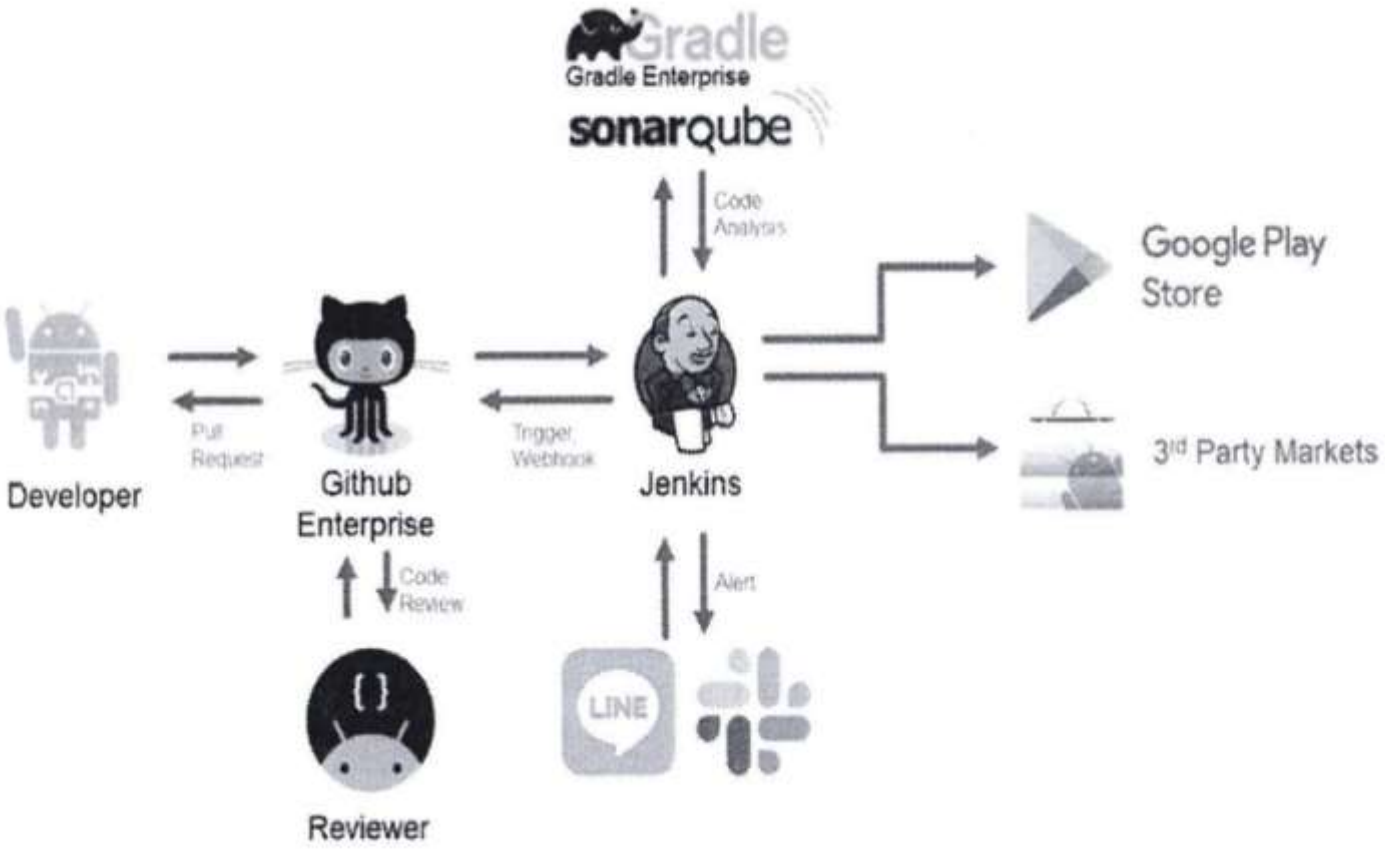
CI/CD

❖ CI/CD Flow – IaC 측면



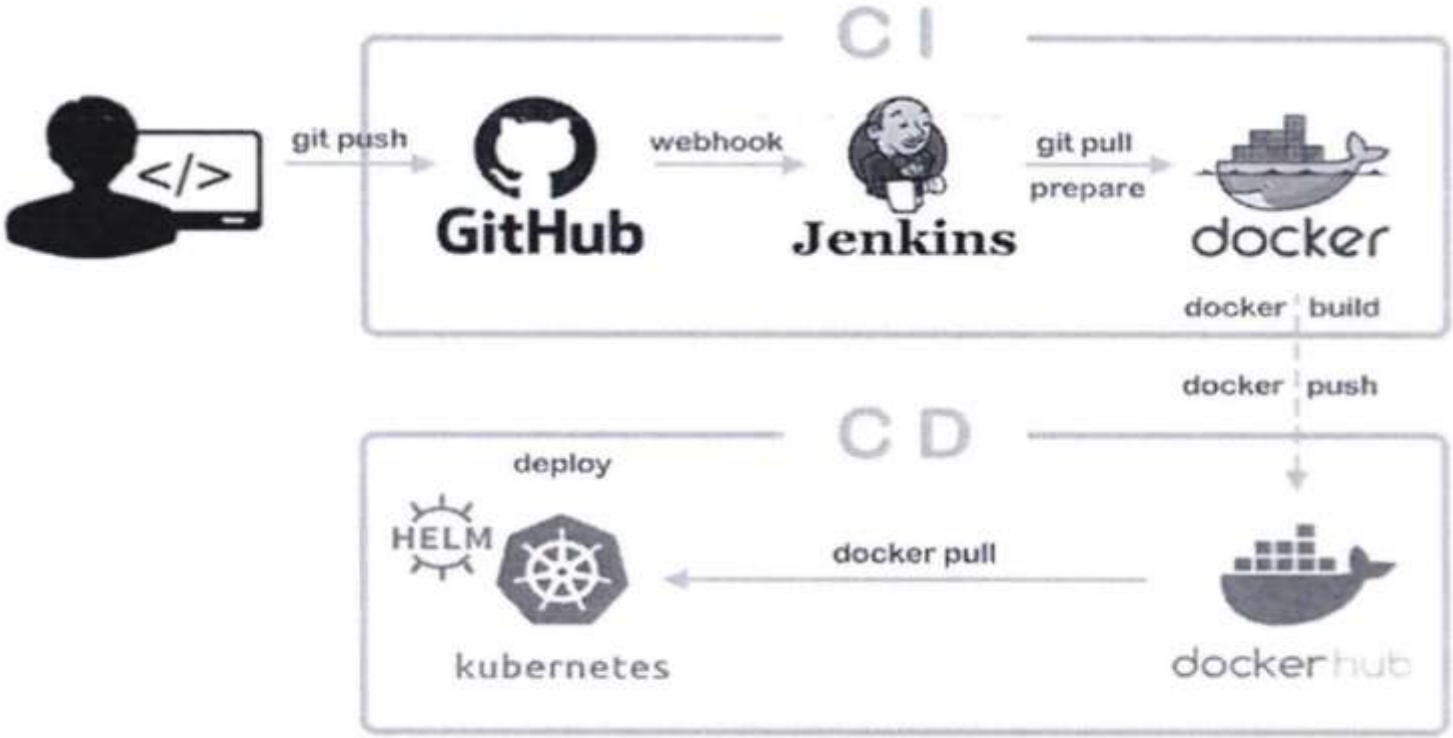
CI/CD

❖ CI/CD Flow – 실업무 측면



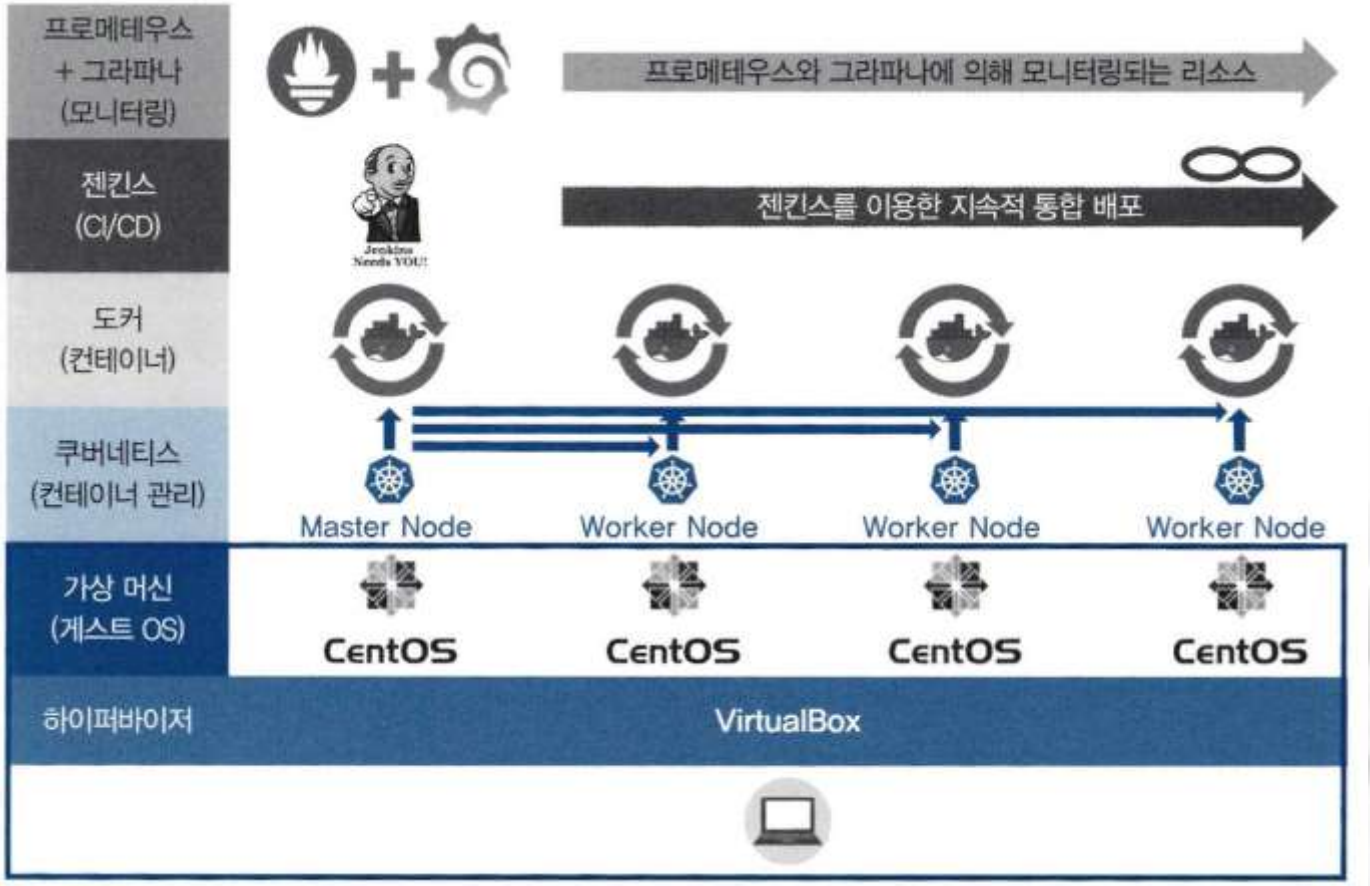
CI/CD

- CI/CD 실현 기술 요소



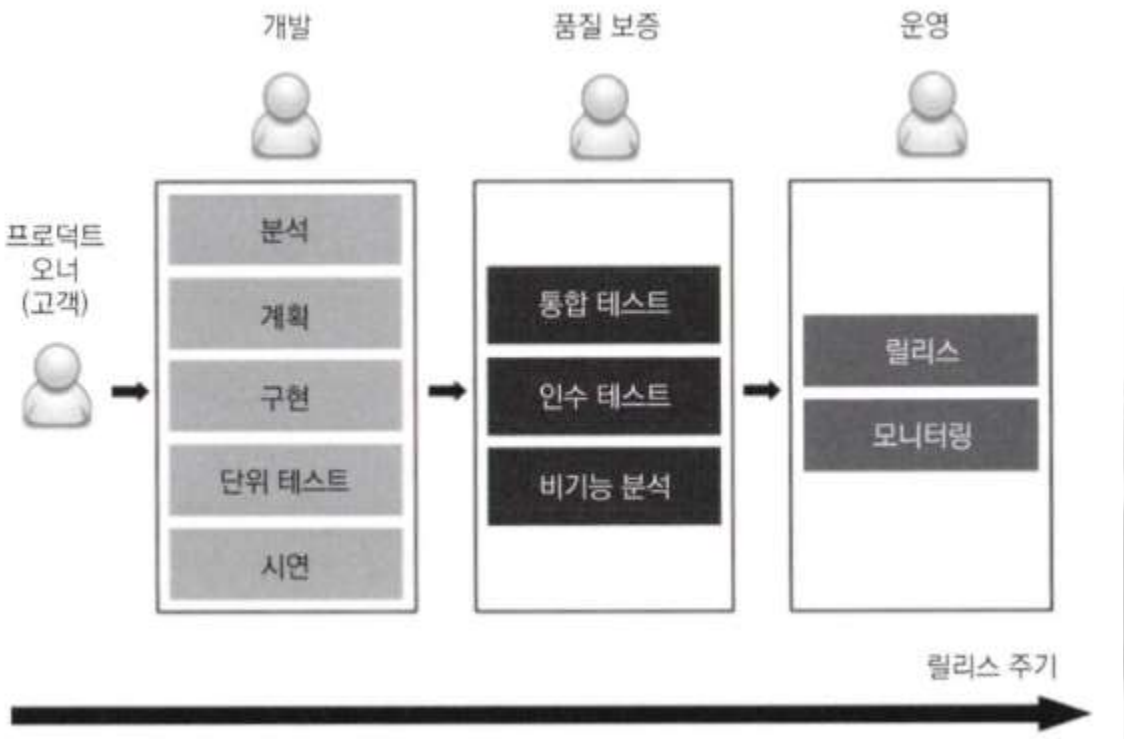
CI/CD

❖ CI/CD 실현 기술 요소



Continuous Delivery

- 지속적 인도
  - ✓ 전통적 인도 서비스
    - 고객의 요청을 정의하는 것으로 시작해서 제품을 릴리스함으로써 종료



Continuous Delivery

- ❖ 지속적 인도
  - ✓ 전통적 인도 서비스
    - ● 과정
      - ◆ 개발 단계
        - ▪ 개발자가 제품을 구현하는 과정
        - ▪ Scrum 이나 Kanban 같은 Agile 기법을 활용해서 개발 속도를 높이고 이해 관계자 와 원할하게 의견을 주고 받음
        - ▪ 제품을 시연하는 자리를 마련해 빠르게 피드백을 받고 TDD 이나 eXtreme Programming 같은 잘 알려진 개발 기법을 적극 수용
        - ▪ 구현이 끝나면 QA 팀으로 넘김
      - ◆ 품질 보증 단계
        - ▪ UAT(사용자 인수 테스트)라고 부르기도 하는데 테스트에 영향이 없도록 트렁크 코드를 프리징하고 진행
        - ▪ 통합 테스트 와 인수 테스트 및 비기능 분석 같은 일련의 테스트를 진행
        - ▪ 버그는 개발 팀에 다시 할당하고 승인을 하면 릴리스
      - ◆ 운영 단계
        - ▪ 릴리스하고 지속적으로 모니터링
        - ▪ 문제가 발생하면 다시 개발자에게 연락을 해서 수정

Continuous Delivery

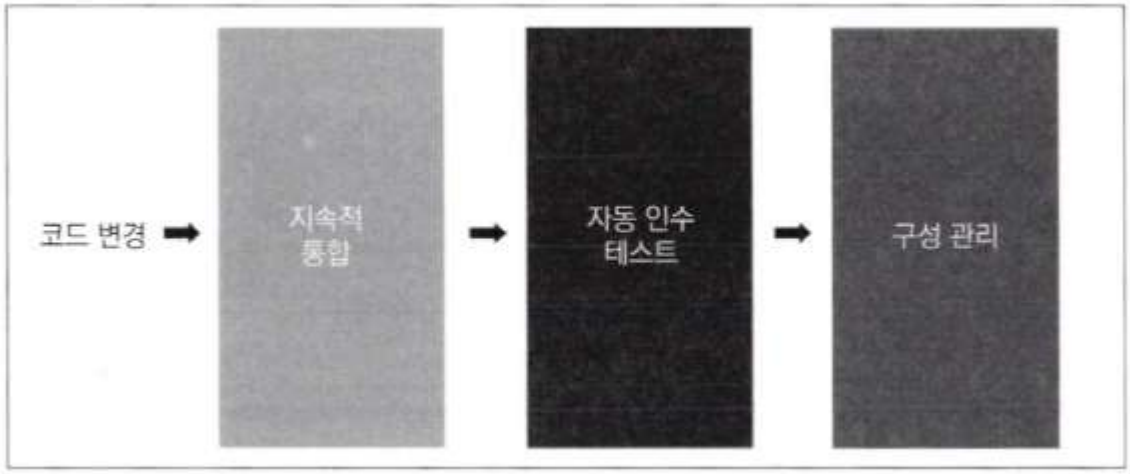
- ❖ 지속적 인도
  - ✓ 전통적 인도 서비스
    - ● 문제
      - ◆ 느린 인도 기간
      - ◆ 느린 피드백 주기
      - ◆ 자동화 미비
      - ◆ 위험한 핫픽스 – 간단한 테스트만 수행하거나 테스트 생략
      - ◆ 스트레스
      - ◆ 의사 소통 부족
      - ◆ 책임의 분산
      - ◆ 낮은 업무 만족도

Continuous Delivery

- ❖ 지속적 인도
  - ✓ 지속적 인도
    - ● 장점
      - ◆ 빠른 제품 인도
      - ◆ 빠른 피드백 주기
      - ◆ 위험도가 낮은 릴리스
      - ◆ 유연한 릴리스 가능

Continuous Delivery

- 자동 배포 파이프라인
  - ✓ 코드 레포지토리에 변경이 발생할 때마다 실행되는 일련의 스크립트로 프로세스가 성공하면 프로덕션 환경으로 배포가 되는 것으로 마무리



Continuous Delivery

- 자동 배포 파이프라인
  - ✓ 단계
    - 지속적 통합(Continuous Integration): 각기 다른 개발자가 작성한 코드가 통합했는지 확인
      - 개발자에게 첫번째 피드백을 제공하는 단계
      - Repository에서 코드를 체크 아웃하고 이를 컴파일 한 후 단위 테스트를 실행하고 코드 품질을 검증
      - 어떤 단계에서든 실패하면 파이프라인 실행이 중단되므로 개발자가 첫번째로 해야 할 일은 CI 빌드를 수정하는 것
      - 핵심 요소는 시간이며 적절한 시간에 반드시 실행돼야 하는데 이 단계를 모두 수행하는데 걸리는 시간이 1시간 인데 만약 개발자가 더 빨리 코드를 커밋해 버리면 파이프라인은 계속해서 실패
      - CI 파이프라인은 출발 단계
      - 이 단계를 설정하는 데 필요한 모든 것은 개발 팀에서 이루어지고 QA 팀이나 운영 팀의 동의가 필요하지 않기 때문에 비교적 간단한 단계

Continuous Delivery

- 자동 배포 파이프라인
  - ✓ 단계
    - 자동 인수 테스트: 개발자가 구현한 기능이 고객의 요구 사항과 맞는지 확인하는 것으로 이 테스트는 수동 품질 보증 단계를 대체
      - 수동 UAT 절차를 대체하도록 고객 과 QA 팀이 작성한 일련의 테스트
      - 이 단계는 제품이 릴리스 할 준비가 됐는지를 결정하는 품질 게이트의 역할
      - 인수 테스트 항목 중 하나라도 실패하면 파이프라인은 중단되며 이후 단계(구성 관리 단계와 릴리스 단계)로의 진행이 중단
      - 인수 테스트 단계를 자동화한다는 개념은 품질 점검을 나중에 하는 것이 아니라 개발 중에 제품에 내재시키자는 것인데 개발자가 구현을 마치는 즉시 고객이 원하는 제품인지를 검증하는 인수 테스트를 거친 후 소프트웨어를 인도하는 것
      - 소프트웨어 테스트에 있어 거대한 사고의 전환이라 할 수 있는데 더 이상 한 사람이나 한 팀이 릴리스를 승인하는 것이 아니라 모든 코드가 인수 테스트 단계를 통과해야 함
      - 이 단계가 CD 프로세스 중 가장 어려운 부분
      - 프로세스의 끝 에서가 아니라 시작 단계부터 테스트 케이스를 만들고 고객과 긴밀하게 협력해야 함
      - 자동 인수 테스트가 CD 프로세스에서 어디에 위치해야 하는지 테스트 유형은 무엇인지에 대해서는 논란이 많음

Continuous Delivery

- ❖ 자동 배포 파이프라인
  - ✓ 단계
    - ● 자동 인수 테스트: 개발자가 구현한 기능이 고객의 요구 사항과 맞는지 확인하는 것으로 이 테스트는 수동 품질 보증 단계를 대체
      - ◆ Agile Testing Metrix

Continuous Delivery

- 자동 배포 파이프라인
  - ✓ 단계
    - 자동 인수 테스트: 개발자가 구현한 기능이 고객의 요구 사항과 맞는지 확인하는 것으로 이 테스트는 수동 품질 보증 단계를 대체
      - Agile Testing Metrix
        - 인수 테스트(자동): 비즈니스 관점에서 본 기능적 요구 사항을 검증하는 테스트로 고객과 개발자가 합의한 소프트웨어의 동작 방식을 스토리나 예제 형식으로 작성
        - 단위 테스트(자동): 버그를 최소화하고 소프트웨어의 품질을 향상하도록 개발자를 지원하는 테스트
        - 탐색적 테스트(수동): 수동으로 하는 블랙박스 테스트이며 경험을 바탕으로 시스템의 문제를 찾거나 개선하는 테스트를 수행
        - 비기능 테스트(자동): 성능, 확장성, 보안 등 과 관련된 시스템 속성을 검증하는 테스트

Continuous Delivery

- 자동 배포 파이프라인
  - ✓ 단계
    - 자동 인수 테스트: 개발자가 구현한 기능이 고객의 요구 사항과 맞는지 확인하는 것으로 이 테스트는 수동 품질 보증 단계를 대체
      - ◆ 통합 테스트
        - 통합 테스트의 의미가 상황에 따라 다를 수 있음
        - 마이크로서비스 아키텍처의 경우 일반적으로 서비스 크기가 작고 단위 테스트와 인수 테스트만 필요하므로 통합 테스트를 대체할 수 있음
        - 모듈형 애플리케이션을 개발하는 경우라면 통합 테스트는 (전체 애플리케이션을 대상으로 하는 것이 아닌) 여러 개의 모듈을 통합하는 컴포넌트 테스트를 의미하는데 이 경우 통합 테스트는 단위 테스트와 인수 테스트 사이에 위치하게 되며 인수 테스트 와 유사한 방식으로 작성되지만 보통은 좀 더 기술적이며 외부 서비스 뿐 만 아니라 내부 모듈의 연동도 필요
        - 단위 테스트와 유사한 통합 테스트는 코드 관점에서 테스트하며 인수 테스트는 사용자 관점에서 테스트
        - CD 파이프라인과 관련해 통합 테스트는 프로세스상에서 별도의 단계로 간략히 구현

Continuous Delivery

- ❖ 자동 배포 파이프라인
  - ✓ 단계
    - ● 자동 인수 테스트: 개발자가 구현한 기능이 고객의 요구 사항과 맞는지 확인하는 것으로 이 테스트는 수동 품질 보증 단계를 대체
      - ◆ 테스트 피라미드

Continuous Delivery

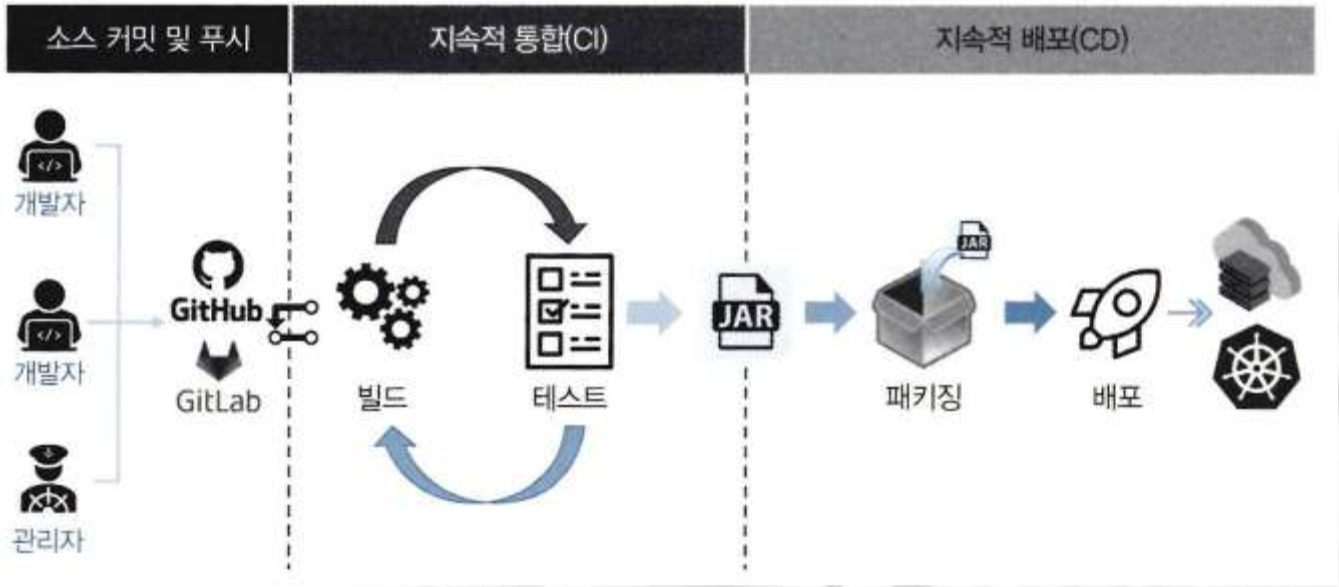
- ❖ 자동 배포 파이프라인
  - ✓ 단계
    - 구성 관리(Configuration Management): 수동 운영 단계를 대체하는데 환경을 구성하고 소프트웨어를 배포
      - ◆ 구성 관리는 소프트웨어와 환경 변화를 추적하고 제어하는 역할
      - ◆ 필수 도구 준비와 설치, 애플리케이션 배포와 관련된 다양한 서비스 인스턴스와 배포 버전, 인프라 인벤토리 및 기타 작업의 확장 관리 등
      - ◆ 구성 관리는 프로덕션 환경의 애플리케이션을 수동으로 구성하고 배포하면서 생기는 문제에 대한 해결책이 되는데 왜냐하면 일반적인 방식에서는 각 서비스가 어떤 속성을 갖고 어디서 실행되고 있는지를 더 이상 알 수 없는 문제가 있기 때문
      - ◆ 앤서블이나 셰프, 퍼핏 같은 구성 관리 도구를 사용하면 구성 관리 파일을 버전 관리 시스템에 저장 할 수 있고 프로덕션 서버에서 발생한 변경 사항을 모두 추적할 수 있음
      - ◆ 운영 팀의 수동 작업을 대체할 만한 분야는 애플리케이션 모니터링으로 운영 중인 시스템의 로그나 지표를 개발자나 데브옵스 팀이 모니터링하는 공통 대시보드에다 실시간으로 스트리밍함으로써 이루어짐

컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 일반적으로 CI는 코드를 커밋하고 빌드했을 때 정상적으로 작동하는지 반복적으로 검증해 애플리케이션의 신뢰성을 높이는 작업
  - ✓ CI 과정을 마친 애플리케이션은 신뢰할 수 있는 상태
  - ✓ CD는 CI 과정에서 생성된 신뢰할 수 있는 애플리케이션을 실제 상용 환경에 자동으로 배포하는 것을 의미
  - ✓ 애플리케이션을 상용 환경에 배포할 때 고려해야 할 사항이 여러 가지 있는데 이를 CD에 코드로 미리 정의하면 실수를 줄이고 적용 시간도 최소화할 수 있음

컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 컨테이너 인프라 관점에서의 CI/CD
    - 개발자가 소스를 커밋(Commit)하고 푸시(Push)하면 CI 단계로 진입
    - CI 단계에서는 애플리케이션이 자동 빌드되고 테스트를 거쳐 배포할 수 있는 애플리케이션인지 확인
    - 테스트를 통과하면 신뢰할 수 있는 애플리케이션으로 간주하고 CD 단계로 넘어감
    - CD 단계에서는 애플리케이션을 컨테이너 이미지로 만들어서 파드, 디플로이먼트, 스테이트 풀셋 등 다양한 오브젝트 조건에 맞춰 미리 설정한 파일을 통해 배포



컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 도구

컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 도구
    - Teamcity
      - Jetbrains에서 만든 CI/CD 도구로 Kotlin을 기반으로 만든 Kotlin DSL이라는 스크립트 언어로 작업을 구성할 수 있음
      - 빌드 작업을 수행하는 에이전트 3개와 빌드 작업 100개를 무료로 사용할 수 있고 더 많은 에이전트를 사용하려면 유료 결제
    - Github Action
      - Github에서 지원하는 워크플로(Workflow) 기반의 CI/CD 도구
      - Github에 저장한 소스 코드를 자동 분석한 결과를 기반으로 Github 액션이 추천하는 방식에 따라 워크플로를 구성하거나 사용자가 직접 워크플로를 정의하는 파일을 작성한 후 Github 저장소에 넣어 사용할 수 있음
      - Github 저장소에 소스 코드를 공개할 경우 Github 액션을 무료로 사용할 수 있으나 한 달에 2,000 분이라는 제한 시간이 있음
      - 추가로 사용할 경우에는 분 단위의 요금이 별도로 부과
    - Bamboo
      - Atlassian에서 만든 CI/CD 도구로 유료이며 사용자의 서버에 설치해 사용
      - Atlassian에서 만든 다른 협업 도구(Jira, Confluence, Trello, Bitbucket 등)를 사용 중이라면 연계해 사용하기 좋음

컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 도구
    - GitLab CICD
      - GitLab CI/CD는 GitLab에서 제공하는 강력한 DevOps 도구로 프로젝트 내에서 코드 변경 사항을 자동으로 테스트, 빌드, 배포할 수 있도록 지원
      - GitLab CI/CD는 GitLab과 통합되어 동작하기 때문에 GitLab 리포지토리와 직접 연결되어 프로젝트 관리와 배포 프로세스를 단순화하고 Github Actions와 다르게 파이프라인을 작성하고 실행하는 데 필요한 설정은 .gitlab-ci.yml 파일에 정의되어 자동화된 작업을 수행
    - Jenkins
      - 오픈 소스 CI/CD 도구로 2004년 출시 당시에는 허드슨(Hudson)이라는 이름을 사용
      - 사용자가 직접 UI에서 작업을 구성하거나 작업 순서를 코드로 정의할 수 있음
      - 오랜 시간 동안 많은 사람이 사용하고 있어서 사용에 필요한 정보를 찾기 쉽고 활용 방법 과 플러그인 개발 관련 커뮤니티 활동이 활발해서 다양한 사용 환경, 언어 및 빌드 도구와 연계할 플러그인이 필요할 경우 인터넷에서 대부분의 플러그인을 쉽게 찾을 수 있음
      - 특정 언어나 환경에 구애받지 않고 범용적인 목적으로 무난하게 사용할 수 있음

컨테이너 인프라 환경에서 CI/CD

- CI/CD
  - ✓ 도구
    - ● 기타
      - ◆ Public 클라우드 기반의 시스템일 때는 클라우드 서비스 제공 업체에서 배포하는 CI/CD 도구(AWS CodeBuild, CodePipeline, CodeDeploy, GCP CloudBuild, Azure Pipelines)를 배포가 중요한 환경에서는 CD 기능에 중점을 둔 Spinnaker, Argo CD를 선택적으로 도입할 수도 있음

Continuous Delivery

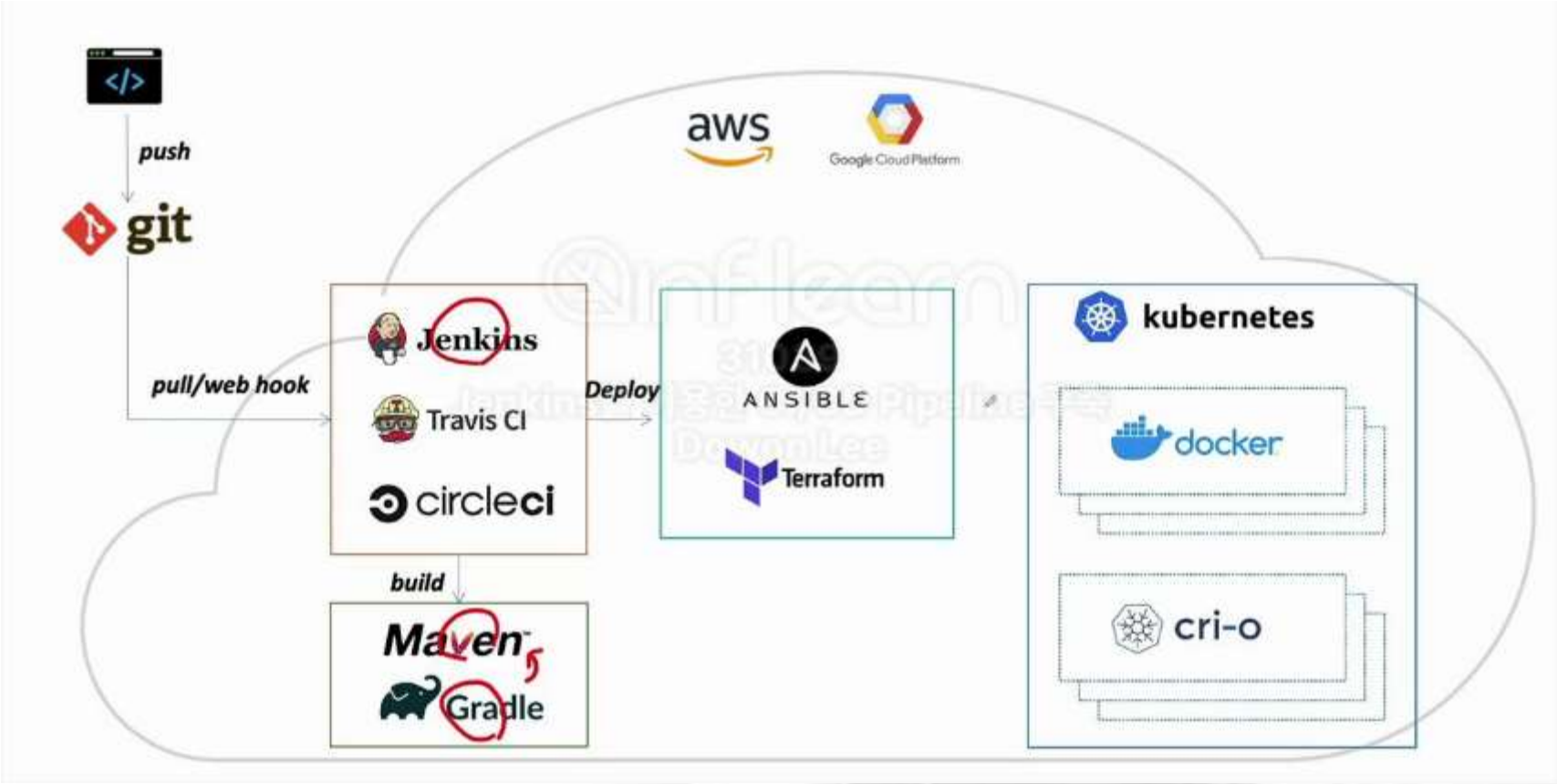
- ❖ CD Process 구축
  - ✓ 도구
    - SCM: github, gitlab, bitbucket
    - Jenkins, Circle CI: CI 및 CD 파이프라인(Build -> Test -> Package) 과 자동화 스크립트 작성
    - Docker & Kubernetes
    - Ansible, Argo CD: 프로비저닝 과 구성 관리 및 애플리케이션을 자동화하는 도구

Continuous Delivery

- ❖ CI/CD Flow
  - ✓ 과정
    - 최신 코드 가져오기
    - 단위 테스트 구현과 실행
    - 코드 개발
    - 단위 테스트 케이스 재실행
    - 코드 푸시 와 병합
    - 코드 병합 후 컴파일
    - 병합된 코드에서 테스트 실행
    - 아티팩트 배포
    - 배포 애플리케이션의 테스트 실행

Continuous Delivery

- CI/CD Flow



Continuous Delivery

- CI/CD Flow