

# (10월 22일) CI, CD

## CI, CD란?



CI : Continuous Integration(지속적인 통합) / CD : Continuous Delivery(지속적인 배달), Continuous Deployment(지속적인 배포)



### 지속적인 통합(Continuous Integration)이란?

- 개발자를 위한 자동화 프로세스
- 개발자의 수동적인 개발 통합을 자동화
- CI를 성공적으로 구현할 경우 코드 변경 사항을 정기적으로 빌드하고 테스트하여 공유된 레포지토리에 통합
- 지속적으로 품질 컨트롤을 적용하는 프로세스를 실행



어려운 단계를 깨거나 몬스터를 잡은 후 바로 저장을 하는 게임과 같이 개발이 완료된 시점에 통합이 진행되어야 한다.

### 지속적인 배달(Continuous Delivery), 지속적인 배포(Continuous Deployment)

- 통합적으로 사용 되는 개념일 수 있지만, 본래 구분하여 사용

#### 지속적인 배달

- CI 과정 이후, 유효한 코드를 최종 프로덕션 환경의 레포지토리에 배달

## 지속적인 배포

- 배포가 이뤄진 것을 수동적으로 실행하여 서비스를 배포하는 것이 아닌 자동화 시키는 것

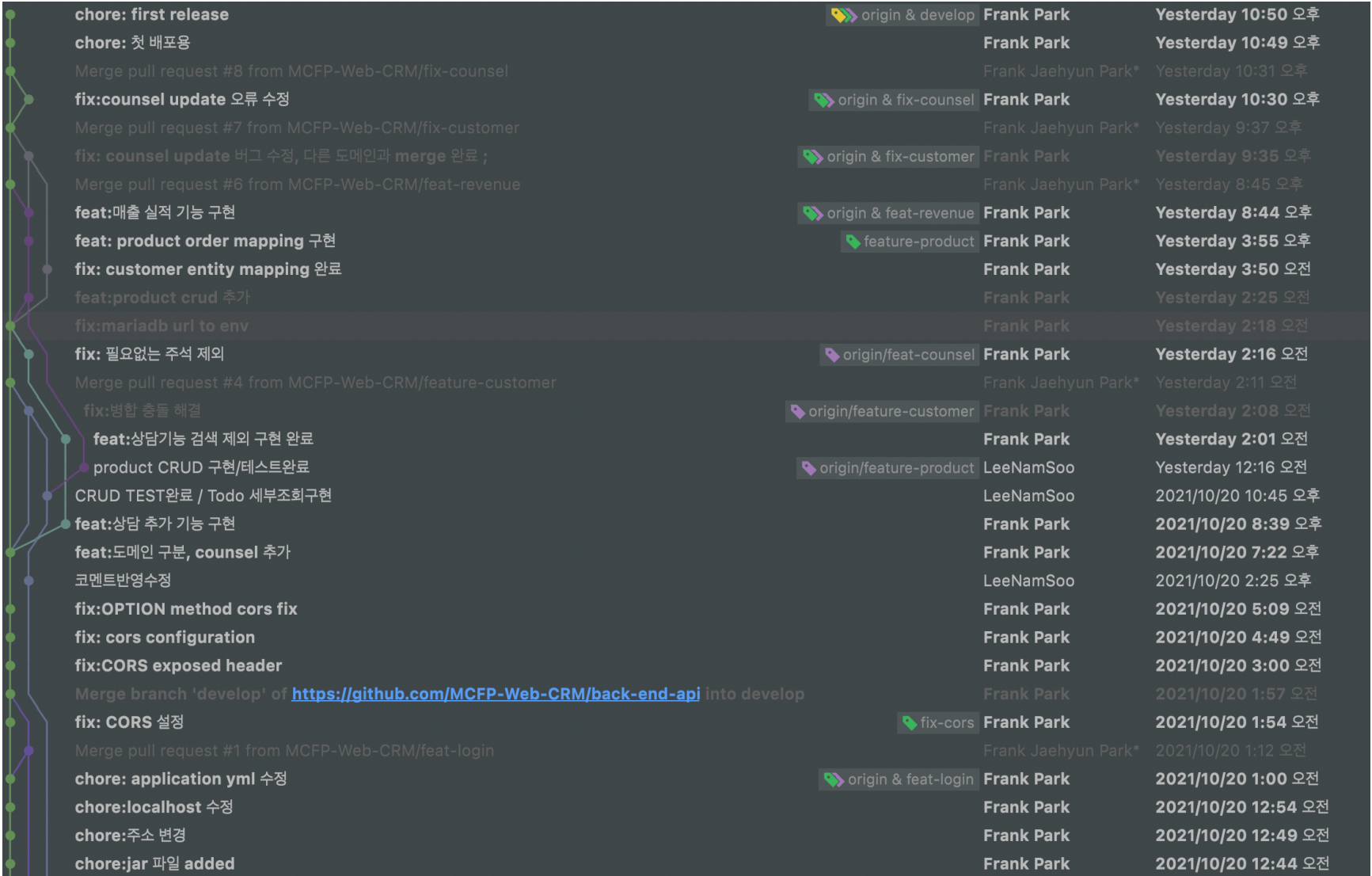


CI/CD는 소프트웨어의 개발부터 배포까지의 단계를 자동화 하는 것을 말한다!

## 왜 중요한 것인가?

- CI를 통해 개발 단계의 통합 주기를 단축
  - 특정 레포지토리에 웹 후크를 설정하여 이벤트( git push, pull request)를 연결하면, 해당 이벤트 발생 시 미리 구현된 흐름을 수행한다.
  - 개발자가 push를 할 때마다 build, test, merge를 각각 진행할 수 있기 때문에 수동적으로 진행할 때 보다 통합 주기를 단축
    - 게임 저장의 자동화!!!!
- CI에서 유효성 검사를 거치기 때문에 레포지토리 내 코드는 항상 기능을 수행할 수 있는 상태 (Ready-to-run 상태)
  - 팀에 새로운 개발자들이 합류했을 때, CI가 진행되지 않는다면 빌드가 수행되지 않게 될 수 있음
  - 그렇다면, 에러 디버깅의 헬로 빠지게 되거나 해당 개발자와 소통을 통해 문제를 해결해야함
  - 너무 많은 시간과 에너지 소요가 생김.
- CD를 적용하면 배포까지 과정이 번거롭지 않고 간소화 될 수 있기 때문에 개발 변경 사항을 프로덕션 레벨에 빠르게 적용할 수 있음
  - 장애 대응에 빨라짐
- CD를 통해 배포 프로세스를 항상 일정하게 구축
  - 수동적으로 배포 및 배포가 진행될 땐 모든 프로세스가 다를 수 있음 (human error)
  - 한 개의 서버를 운영할 때는 문제가 없지만, 다수의 서버를 관리하게 되는 경우 매우 크리티컬
- 1 명의 개발자가 1 개의 프로그램을 개발하여 1개의 서버에 배포하는 단계를 벗어나 다수의 개발자가 함께 프로그램을 개발하여 다수의 서버에 배포

## Frank, NamSoo Project Happening...



- git 브랜치를 main-develop-feat-fix 단계로 구분
    - feat 브랜치에서 기능 구현 후 develop 브랜치로 merge
    - production ready 상태가 된다면 main branch로 merge
- ⇒ 해당 과정의 관리를 프로젝트 일원이 수행

## 그 결과?

39 new commits pushed to main by jehyn923 어제 ▾

d400b6db

- feat: jwt 발급, 필터 기능 적용

d5c8a1e0

- feat: jwt 발급, 필터 기능 적용

bee4adb2

- chore:domain별 package 분리

14c96fdf

- fix:User Entity Getter 추가

b92d1fcc

- 고객기능추가

b2002ff5

- fix: User entity Data 추가

e9a44ad3

- feat:user register, login 구현

0715acb7

- chore:jar 파일 added

17227e3d

- chore:주소 변경

42604b2d

- chore:localhost 수정

16355ec8

- chore: application yml 수정

f7c0c074

- Merge pull request #1 from MCFP-Web-CRM/feat-login

5fb573a6

- fix: CORS 설정

57f345da

- Merge branch 'develop' of <https://github.com/MCFP-Web-CRM/back-end-api> into develop

a95cb4e7

- fix:CORS exposed header

39b3351a

- fix: cors configuration

77e02430

- fix:OPTION method cors fix

cdd65124

- 코멘트반영수정

1399c545

- feat:도메인 구분, counsel 추가

7d56008f

- feat:상담 추가 기능 구현

dd2475de

- CRUD TEST완료 / Todo 세부조회구현

c4454d19

- product CRUD 구현/테스트완료

55a42523

- feat:상담기능 검색 제외 구현 완료

e9e045f5

- fix:병합 충돌 해결

4e8dfafb

- Merge pull request #4 from MCFP-Web-CRM/feature-customer

b94ab31a

- fix: 필요없는 주식 제외

3ebf6c0a

- fix:mariadb url to env

28e1717f

- feat:product crud 추가

6cf69df4

- fix: customer entity mapping 완료

781eb078

- feat: product order mapping 구현

bfeff4cc

- feat:매출 실적 기능 구현

35f97d19

- Merge pull request #6 from MCFP-Web-CRM/feat-revenue

6fc3d6fb

- fix: counsel update 버그 수정, 다른 도메인과 merge 완료


766f63f5

- Merge pull request #7 from MCFP-Web-CRM/fix-customer

- 39개의 commit이 한번에 main branch 로 push 됨

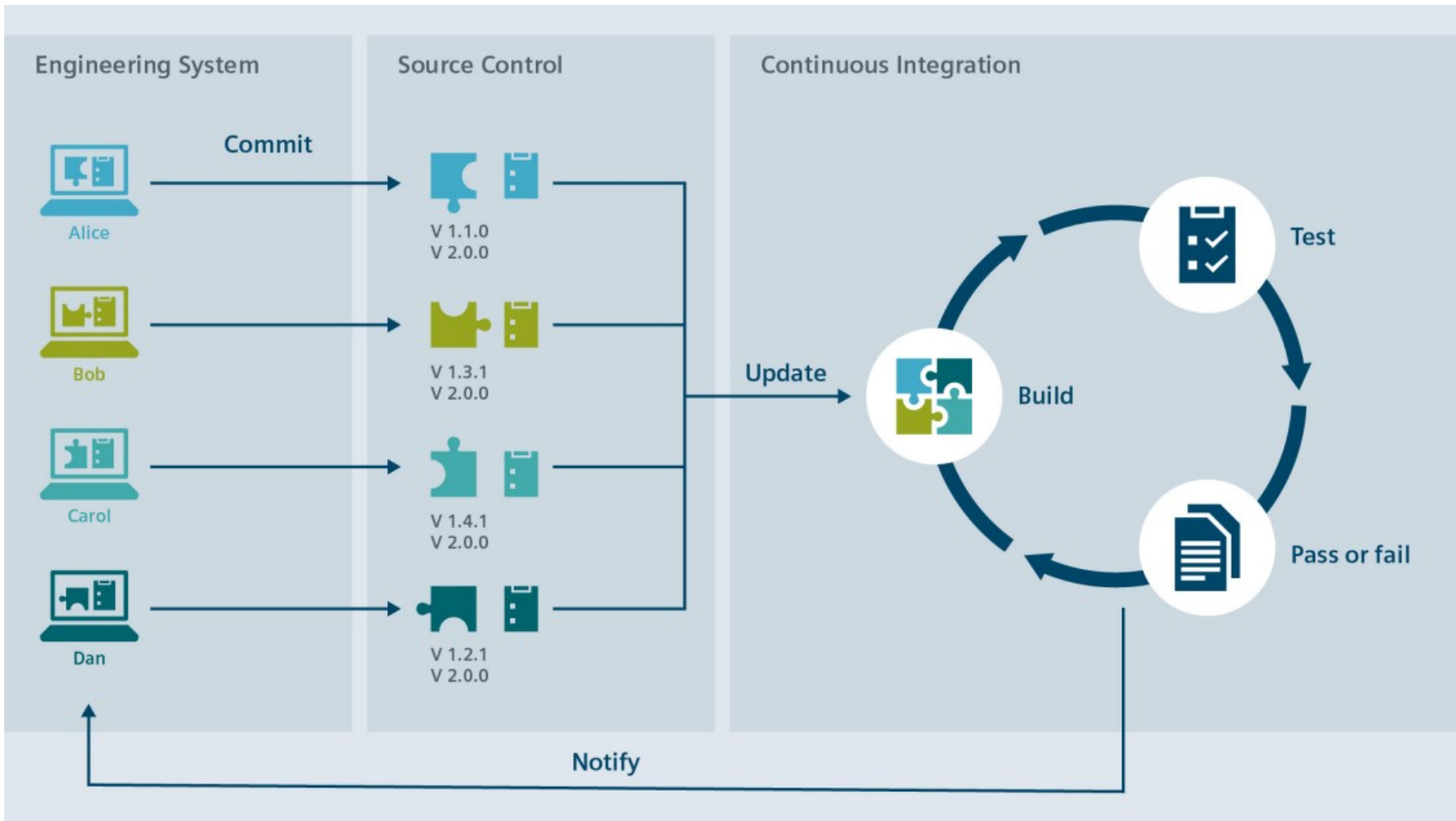
## 안 좋은 이유

- 각 기능별 개발자의 코드들의 진행 상황이 상이해져서 이 후 merge 작업이 수행될 때 **crash 가능성**
- 다른 분야의 개발자 와의 연동에 있어 delay가 생김
  - 프로덕션의 배포가 점점 늦어지면서 즉각적인 연동 수행을 할 수 없고 하나의 시스템 배포를 기다리는 상황이 되버림
  - 전체 프로젝트에 delay가 생겨버림
    - 실제로 Bithumb Project 당시 팀 내의 브랜치 관리가 되지 않아 마지막 주까지 Front-Back의 연동이 수행되지 않았음
    - CORS 문제 해결하지 못하고 프로젝트 미완성 상태로 발표 참가

 하나의 프로젝트 개발을 여러명에서 하기 때문에 **빠르고 유기적인 작업 수행을 위해 CI/CD가 필요**

## Continuation Integration의 단계



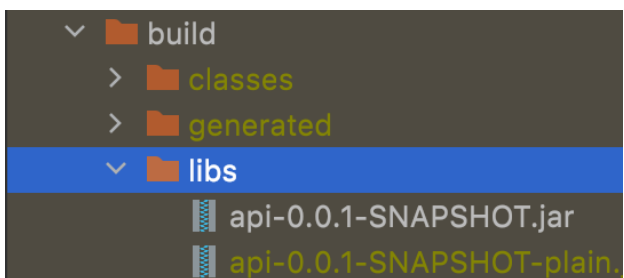


## Build → Test → Merge

### 1. Build

- 개발자가 작업한 소스코드를 수행 가능한 프로그램 파일로 가공하는 작업
- 프로그램이 로컬 혹은 배포 환경에서 수행되기 위해선 Build 작업이 완료된 파일이 필요하다~!!!

⇒ Spring Framework로 작성된 코드를 수행 가능하게 하려면 Maven / Gradle이라는 빌드 도구로 java 환경에서 수행될 수 있는 .jar 파일을 만들어야 한다.



```
FROM openjdk:11-jre-slim
LABEL maintainer="jehyn923@gmail.com"
VOLUME /tmp
ARG JAR_FILE=./build/libs/*.jar
ADD ${JAR_FILE} app.jar
EXPOSE 8081
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

**./gradlew clean build** 커맨드를 통해 빌드한 jar 산출물

jar 파일을 도커 환경에서 실행시키기 위해 작성된 Dockerfile

### 2. Test

- 단위 테스트
  - 빌드 된 프로그램의 특정 모듈, 기능이 정상적으로 수행되고 예상된 결과를 나타내는 지, 특정 상황에서 개발자가 원하는 결과를 내는지 확인할 수 있는 테스트.
  - 꼼꼼할수록 좋다

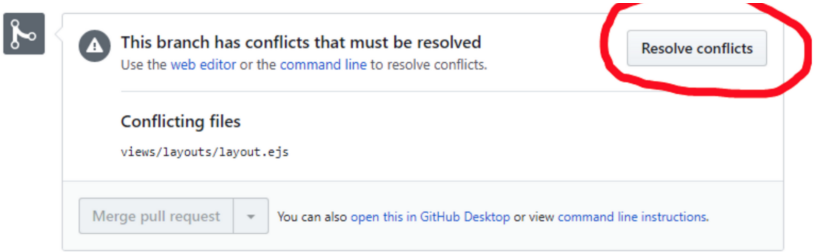
⇒ 본 단계에서 fail이 일어날 경우, CI 툴은 해당 fail을 개발자들에게 report 한다.

### 3. Merge

- 프로그램의 모듈을 하나의 프로그램으로 합치는 것
- 모듈 단위의 프로그램이 각자 다른 파일로 수행되는 것은 의미가 없고, 모두 main 브랜치에 merge가 이뤄져야 한다.
- 하나의 프로그램 모듈이 main 혹은 특정 브랜치로 merge 가 이뤄질 때 코드 상의 충돌 상황은 없는지 자동적으로 체크하며 있을 경우 report



Github merge 대상 branch가 충돌 상황이 없을 때

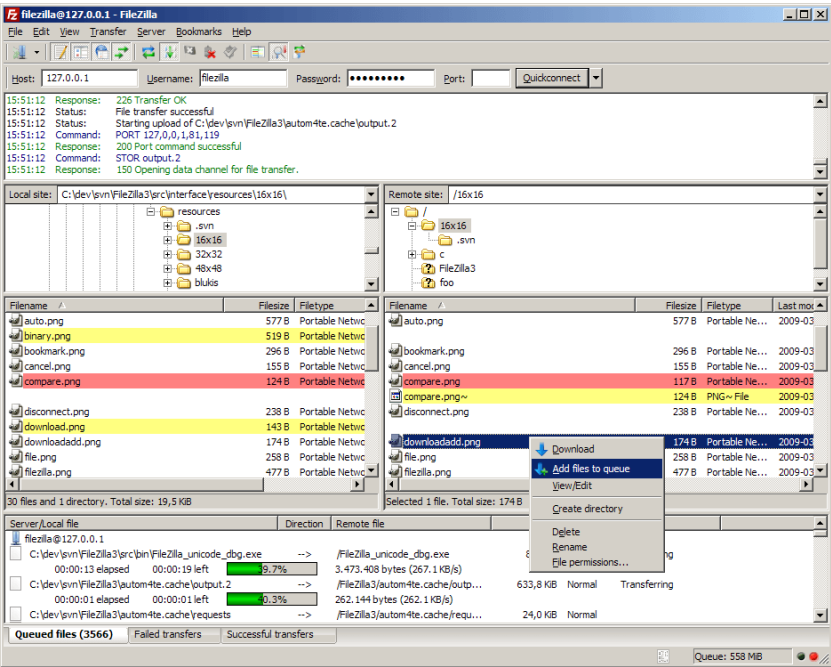
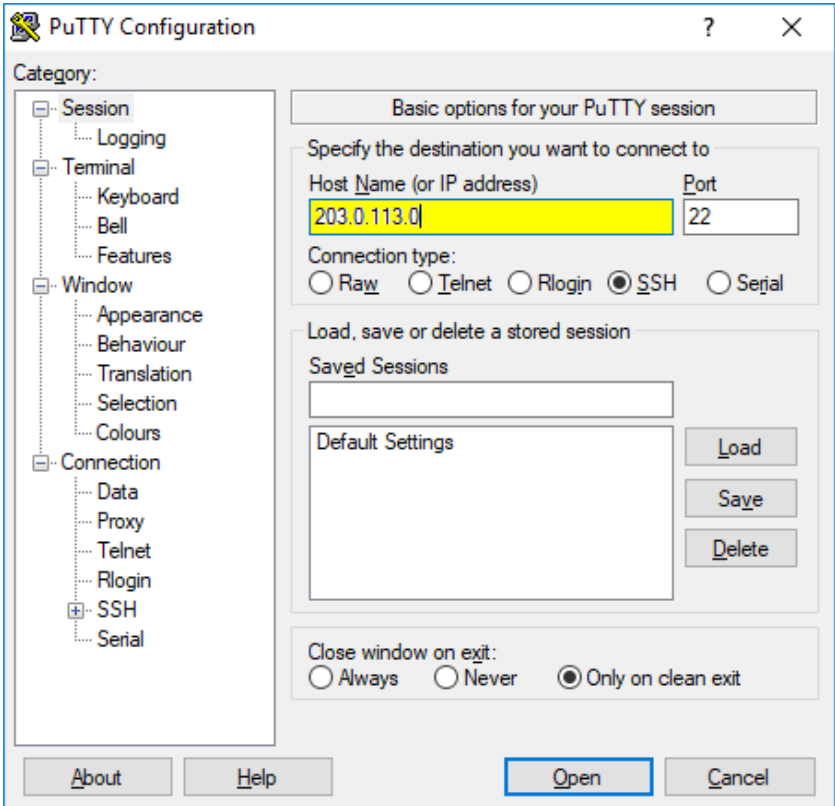


Github 충돌 상황이 있을 때(하)

## Continuous Delivery / Continuous Deployment

- 프로그램 배포 작업의 자동화

이게 생기기 이 전엔 어떻게 애플리케이션을 배포했나?



### Putty와 FTP를 이용한 배포 방식

- 원시적인 방식
- 빌드가 완료된 파일을 FTP를 통해 배포 서버에 저장
- Putty를 이용하여 서버에 접속한 후 직접 빌드 파일을 수행하는 형태

### 배포 서버에서 git pull

- 서버에 폴더를 만들어서 origin에서 pull 하는 방식
- 빌드가 안되어 있는 경우가 많음
- 서버 자체에 gradle 이나 maven을 설치 → git pull → 파일 실행

## 두 가지 방식의 문제점?

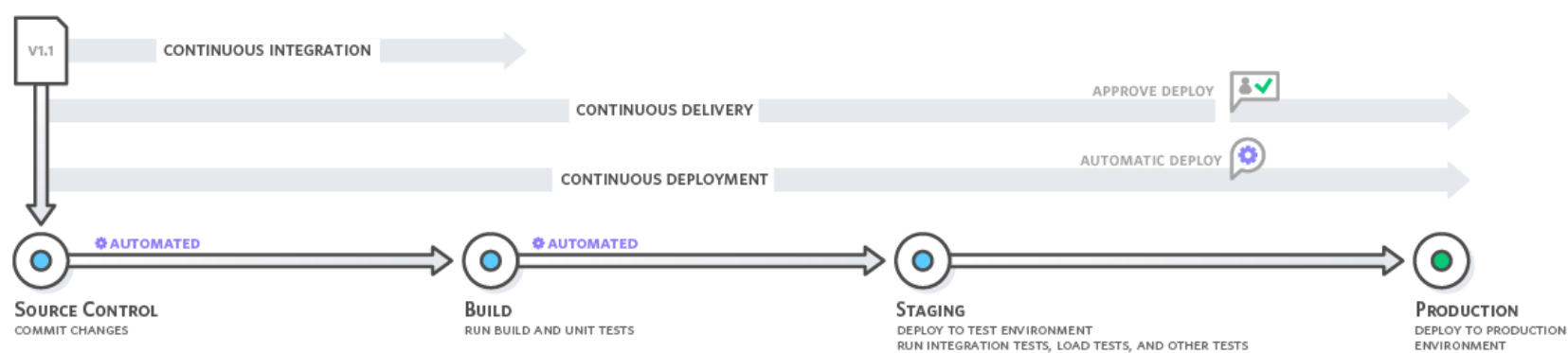
- 프로젝트 배포에 있어 너무 많은 작업이 수행되어야 한다.
  - 기본적으로 3 번 이상의 수동적인 작업이 필요

⇒ 만약 배포된 프로그램에 수정이 반영되어야 한다면? 방금 릴리즈했는데 장애가 발견되었다면? 하나의 서버가 아닌 다양한 서버에 배포가 이뤄져야 한다면?

⇒ 똑같은 작업을 계속 반복적으로 수행하여야 하기 때문에 매우 소모적

- 빠른 개발 환경에 비해 릴리즈가 늦어지는 경우가 생김
  - 업데이트가 사용자에게 늦게 전달되어 시장의 니즈를 바로 반영할 수 없음
  - 장애 상황을 즉각적으로 대응하지 못함

그렇기 때문에 우리는 지속적인 전달과 지속적인 배포가 필요하다!



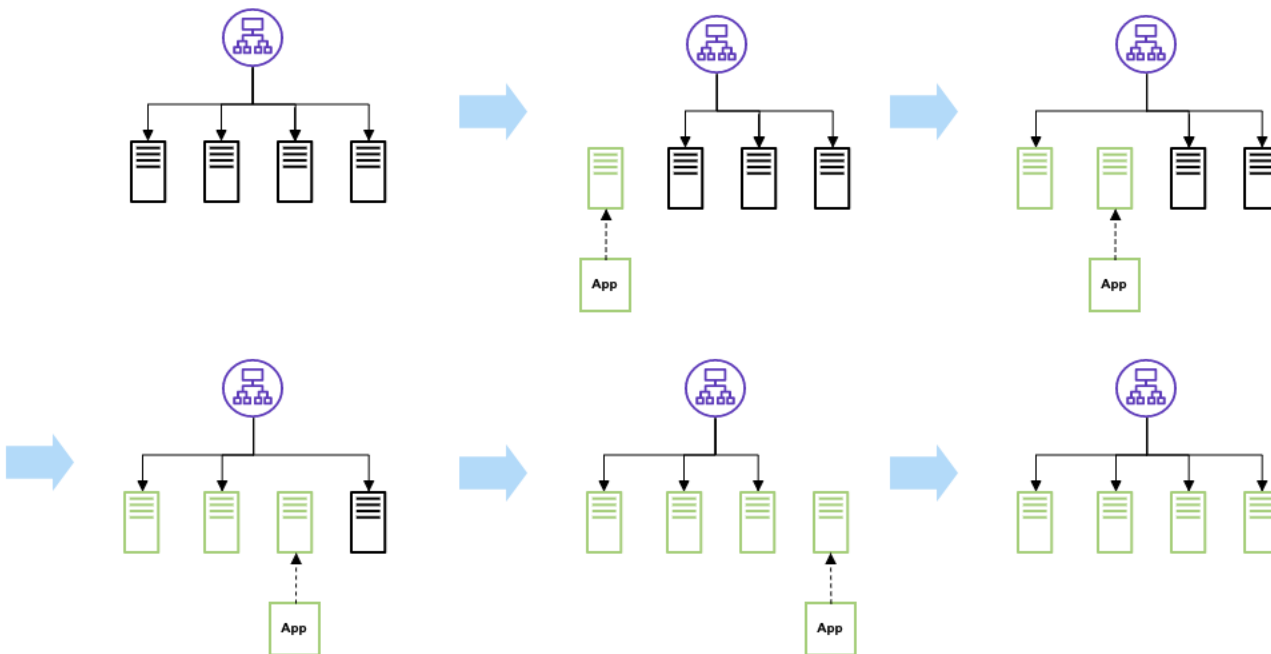
💡 CD 작업은 이 과정을 자동화하여 프로그램에 변경이 있을 때 마다 동시 다발적으로 레포지토리에 프로그램 파일을 전달하고 실행하는 작업을 수행하기 때문에 **많은 시간과 노력이 절약** 더하여, 개발자와 프로덕션에 관련된 많은 사람들이 이 과정을 CI / CD 툴로 모니터링 할 수 있기 때문에 통합 배포 단계에서 일어난 오류들을 즉각적으로 확인, 수정 가능

배포는 그냥 전달해서 수행하면 되는가?

- NOPE! 배포에도 전략이 있다.
- 최근 기술 서비스들은 대부분 작게 만들고 자주 배포하는 방식
- 변경을 빠르게 반영하는 것은 좋은 것, but 현재 잘 작동하는 것을 손 대는 것은 항상 위험
- 무작정 새로운 것을 배포하는 것은 리스크가 있기 때문에 이를 최소화 하는 배포 전략이 존재!

배포 전략

1. 롤링 배포



- 사용 중인 환경에 변경된 애플리케이션만 반영
- 각 환경에 있는 애플리케이션을 일시정지 한 후, 변경된 애플리케이션이 설치되면 새로운 앱을 실행
  - 예) 4 개의 서버가 가동되어 있는 상태고 로드밸런서가 있다고 가정할 때
    - 2개의 서버를 로드밸런서에서 등록 해제한 후, 새로운 애플리케이션을 설치
    - 설치가 완료되면 2개의 서버를 등록한 후, 나머지 2개의 서버를 등록 해제하여 설치

장점

- 기존에 사용되고 있는 인스턴스들로만 배포를 진행할 수 있기 때문에 새로운 인스턴스가 필요하지 않음

단점

- 배포 중에는 가용되는 인스턴스 수가 줄어들기 때문에 성능 부분에서 문제가 생길 수 있음
- 따라서, 서버 로드밸런싱 처리에 효율성을 따져야 한다

2. 블루/그린 배포



- 블루는 Old Release, 그린은 Latest Release가 된다.
- 기존의 서버 인스턴스를 가동하고 있는 상태에서 변경이 반영된 서버 인스턴스를 설치한 후 새로운 버전이 준비가 완료될 때 기존의 연결을 해제한 후 새로운 버전에 연결한다.

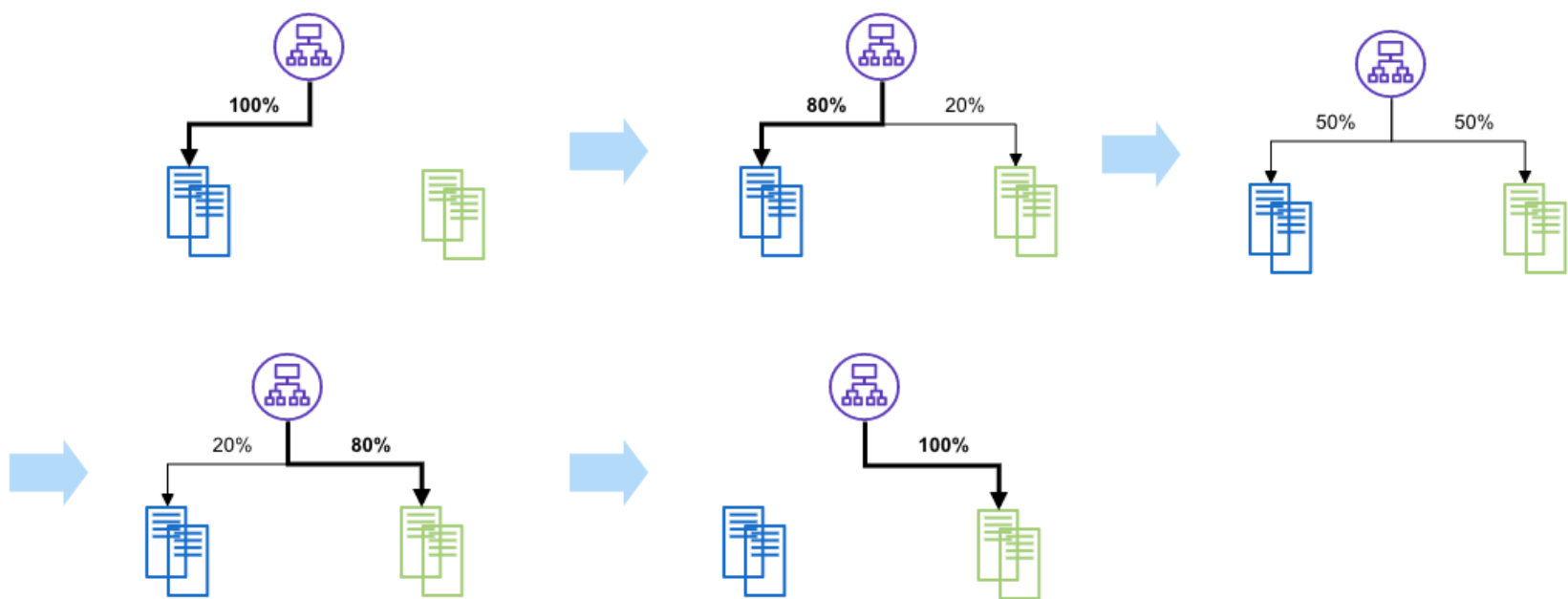
장점

- 기존 서버 인스턴스와 같은 크기의 서버를 함께 사용하기 때문에 무중단 배포가 가능하며 성능적인 면에서도 큰 차이가 없다.
- 일제히 서비스를 전환하는 것이기 때문에 롤백이 쉽다
- 운영 환경에 영향을 주지 않은 상태로 실제 서비스 환경 안에서 테스트 가능

단점

- 이전 버전과 새 버전을 동시에 실행하기 때문에 리소스가 두 배로 필요하다는 단점 존재

3. 카나리 배포



카나리란?

- 예전 광부들이 탄광에 가기 전 테스트처럼 탄광 안으로 보내는 새의 이름
- 새가 탄광안에서 나오지 못하면 유독 가스가 있다고 판단했다고 한다.



## 카나리 배포

- 새로운 버전을 하나씩 올려보면서 해당 버전이 잘 동작했을 때 기존 버전을 내리는 것
- **위험을 빠르게 감지할 수 있는 배포 전략**
- 새로운 버전을 일부 사용자들에게 배포함으로써 **사용자 피드백을 받을 수 있고**, 실제 서비스에서의 안정성을 확신할 수 있다.

## 장점

- 서비스 중단 없이 새로운 버전을 릴리즈 가능
- 실제 서비스 운용 환경에서 기술적 테스트가 가능하기 때문에 서비스 안정성을 확신할 수 있음
- 기술적인 안정성 외에 **실제 사용자의 피드백을 감지할 수 있다**

## 단점

- 블루 / 그린 배포전략 과 같이 **2배의 리소스를 가용하기 때문에 비용적인 부분의 문제가 생길 수 있다!**



배포 전략에는 정답이 있지 않다. 각 전략에 강점과 취약점이 있는 만큼 **각 서비스 환경에 알맞는 전략 선택이 중요할 것!!!**



### 서버 사이드 테스트 자동화 여정 - 1. 테스트 자동화를 시작한 계기와 그 첫 발걸음 - LINE ENGINEERING

안녕하세요. LINE 미디어 플랫폼 개발과 운영 업무를 담당하고 있는 하태호입니다. 미디어 플랫폼은 LINE 메시징 서비스 및 LINE의 다양한 패밀리 서비스에서 생성되고 유통되는 미디어 콘텐츠(이미지, 비디오, 오디오, 라이브 스트림 등)를 각 서비스 요구 사항에 맞추어서 가공하고 저장한 뒤 사용자에게 전달하는 역할을 합니다. 미디어 플랫폼은 LINE 내 수많은

 <https://engineering.linecorp.com/ko/blog/server-side-test-automation-journey-1/>

LINE Engineering

서버 사이드 테스트 자동화 여정 - 1편

**테스트 자동화**를 시작한  
계기와 첫 발걸음

### CI/CD의 자동화된 테스트 | TeamCity CI/CD 가이드 | JetBrains

자동화된 테스트는 CI/CD 파이프라인의 핵심적인 부분입니다. 테스트가 CI/CD 프로세스에 적합한 이유와 TeamCity를 사용해 테스트 피라미드를 구축하는 방법을 알아보세요.

 <https://www.jetbrains.com/ko-kr/teamcity/ci-cd-guide/automated-testing/>

### 라이더스 개발팀 모바일에서 CI/CD 도입 | 우아한형제들 기술블로그

이 글은 CI/CD를 안드로이드에 도입하게 되면서 정리한 내용입니다. 구축 및 운영하고자 하시는 분에게 경험을 공유하고자 합니다. 안녕하세요 라이더스 개발팀 장인수 입니다. 우선 라이더스 개발팀이 하는 일을 소개 합니다. 저희 라이더스 개발팀은 배달되지 않는 음식점의 음식을 민트색 헬멧을 쓴 라이더 분들이 오토바이를 이용하여 음식을 픽업

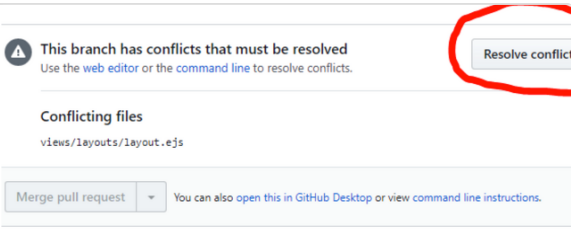
 <https://techblog.woowahan.com/2579/>

WOOWATECH  
**우아한**  **Tech**  
우아한형제들의 기술조직 이야기를 전합니다.

Git Conflict 가 났을 때 해결 방법

일반적으로 회사에서 업무를 하면 Git 을 사용하는 경우가 가장 많다. 본인이 작업한 내용을 push 하고 merge까지 하려고 하면, 다른 사람이 작업한 내용과 conflicts 가 나는 경우가 있다. 최근 Github 에서는 충돌이 났을 때, 간단한 충돌의 경우 웹 상에서 해결할 수 있는 기능이 존재한다.

🔗 <https://baek-kim-dev.site/251>



지속적 전달이란 무엇입니까? - Amazon Web Services

지속적 전달(Continuous Delivery)은 프로덕션에 릴리스하기 위한 코드 변경이 자동으로 준비되는 소프트웨어 개발 방식입니다. 현대 애플리케이션 개발의 기반인 지속적 전달은 빌드 단계 이후의 모든 코드 변경을 테스트 환경 및/또는 프로덕션 환경에 배포함으로써 지속적 통합 을 확장합니다. 적절하게 구현할 경우, 개발자는 언제나 즉시 배포할 수 있

🔗 <https://aws.amazon.com/ko/devops/continuous-delivery/>



매번 헛갈리는 CI/CD 배포 전략 정리해버리기 | DevelopersIO

안녕하세요. CX사업본부 모바일 사업부의 정하은입니다👋 4월부터 쉬지 않고 프로젝트 개발을 달려오고, 드디어 이번 한 주동안 휴식기간을 얻을 수 있게 되었는데요. 개인 공부 시간이 생긴 참에 미뤄두었던 블로그를 써보았습니다. AWS 자격증 중 하나인 DVA나 DOP에서 반드시 등장하는 배포 파트. 블루/그린 배포, 카나리 배포, 롤링 배포 등 상황

🔗 <https://dev.classmethod.jp/articles/ci-cd-deployment-strategies-kr/>

매번 헛갈리는  
CI/CD 배포 전략  
정리해버리기



Canary Release

Architecture 원문: <http://martinfowler.com/bliki/CanaryRelease.html> Canary Release는 새로운 버전의 소프트웨어를 운영 환경에 배포할 때, 전체 사용자들이 사용하도록 모든 인프라에 배포하기 전에 소규모의 사용자에게만 먼저 배포함으로써 리스크를 줄이는 기법이다. Blue Green Deployment 와 유사하게, 먼저 어떤 사용자도 들어오지 않

🔗 <https://m.blog.naver.com/muchine98/220262491992>

