
# CI / CD

| | |
|---|---|
| 주차 | 25주차 |
| 스터디 일자 | @2024/05/03 |

## CI / CD

> **CI / CD** 란, 애플리케이션 개발 단계부터 배포까지 **모든 단계를 자동화하여** 좀 더 효율적이고 빠르게 사용자에게 빈번히 배포할 수 있도록 하는 것

## CI (Continuous Integration)

> **CI** 는 **지속적인 통합** 이라는 의미로 간단히 요약하면 **빌드 / 테스트 자동화 과정** 이다.

애플리케이션의 버그 수정이나 새로운 코드 변경을 주기적으로 `빌드 및 테스트` 하면서 레포지토리에 통합( `merge` ) 하는 것을 의미하며,

여러 개발자가 동시에 애플리케이션 개발과 관련된 코드 작업을 할 경우 코드 충돌이 발생할 수 있는 문제를 방지할 수 있다.

- 지속적인 통합은 커밋할 때마다 빌드와 테스트가 자동으로 이루어져 문제가 발생하는 것을 방지한다.

### 주기적으로 통합(merge) 해야하는 이유?

여러 개발자들이 하나의 프로젝트를 진행하고 있음.

여기서 여러 개발자들이 빈번하게 merge 하지 않고 오랜 기간 동안 개발을 진행하면서 작업한 많은 코드를 한번에 merge 하게 되면 **충돌되는 코드가 많이 생길 수 있다.**

⇒ 그렇기 때문에, 가능한 작은 단위로 나누어서 주기적으로 빈번히 개발하고 계속해서 통합하는 것이 중요하다.

### CI의 장점

- 주기적인 merge로 충돌 가능성이 낮아져 개발 생산성이 증가하며 버그와 문제점을 빠르게 찾을 수 있다.
- 테스트를 통과한 코드만 레포지토리에 올라가기 때문에 코드가 안정성있고 좋은 퀄리티를 유지할 수 있다.

## CD

> CD (`Continuous Delivery`)는 지속적인 제공이라는 의미와 ,
> CD (
> `Continuous Deployment`) 지속적인 배포라는 의미가 있다.

### 지속적인 제공 (Continuous Delivery)

CI 에서 Build 되고 Test 된 후에, 배포 단계에서 release 할 준비 단계를 거치고 문제가 없는지 검증을 마친 이후에 서비스를 제공해도 되겠다는 판단이 되면

**수동적으로 프로덕션 환경으로 배포를 진행하는 것**

### 지속적인 배포 (Continuous Deployment)

CI 에서 Build 되고 Test 된 후에, 배포 단계에서 release 할 준비가 되면

**자동화를 통해 고객이 사용 가능한 프로덕션 환경까지 배포하는 것**

### CD의 장점

개발자는 배포보다는 개발에 더욱 신경을 쓸 수 있으며, 품질 저하 없이 최대한 빨리 사용자에게 새로운 기능을 제공할 수 있다.

## CI/CD Pipeline

### Pipeline 이란?

컴퓨터 과학에서 파이프 라인은 제공된 데이터 또는 코드에 대해 사전정의된 작업을 수행하는 일련의 처리 단계

파이프 라인의 사용 목적은 반복적인 프로세스를 자동화하여 시간을 절약하고 정밀도를 높히는 것으로,

파이프라인의 이러한 장점은 CI/CD 인프라와의 호환성와 효율성을 높여준다.

### CI / CD 파이프 라인을 구성하는 요소

1. 빌드 ( 소프트웨어 컴파일 )
2. 테스트 ( 호환성 및 오류 검사 )
3. 릴리즈 ( 버전 제어 저장소의 어플리케이션 업데이트 )
4. 배포 ( 개발에서 프로덕션 환경으로의 변환 )
5. 규정 준수 및 유효성 검사

⇒ CI / CD 파이프 라인의 목표는 빌드, 테스트 및 제공을 수동 처리보다 **더 빠르고 안정적으로 만드는 것**

# Jenkins


> **Jenkins**는 소프트웨어 개발 시 배포 서비스와 통합 서비스를 제공하는 자동화 도구다.

Jenkins를 사용하면 소프트웨어 개발 프로세스의 자동화를 통해 개발자들이 소스 코드의 변경 사항을 빠르게 테스트하고 배포할 수 있다.

## Jenkins 기능

- **각종 배치 작업의 간략화**

  프로젝트 기간 중에 개발자들은 개발 작업 이외에, Deploy 작업과 같은 단순 작업에 시간과 노력을 들이는데, 이러한 작업들을 Jenkins를 사용함으로 해결할 수 있다.

- **코드 표준 준수여부 검사와 빌드 파이프라인 구성**

  Jenkins는 개인이 미처 하지 못한 코딩 표준 준수 여부나 정적 분석을 통한 **코드 품질 검사**를 빌드 내부에서 수행한다.

  또한, 2개 이상의 모듈로 구성되는 레이어드 아키텍처가 적용된 프로젝트에서 빌드 파이프라인 구성을 간단히 할 수 있으며, 스크립트를 통해 복잡한 제어도 가능하다.

- **자동화 테스트**

Jenkins는 버전 관리 시스템과 연동하여 코드 변경을 감지하고 자동화 테스트를 수행하기 때문에 만약 개인이 실시하지 못한 테스트가 있어도 Jenkins가 테스트를 한번 더 수행한다.

# Jenkins 특징

**편리한 설정**

- **웹 기반의 콘솔로 다양한 인증 기반과 결합**하여 권한 관리 기능을 통해 안전한 빌드 / 배포 환경을 구축할 수 있다.
- **수많은 플러그인을 사용**하여 자동화 할 수 있어 반복되는 작업을 줄일 수 있다.
- 빌드 / 배포의 결과에 대해 **통지 받을 수 있는 설정이 간편**하고 다양한 채널을 통해 빠르게 피드백을 받을 수 있다.

**안정적인 빌드 / 배포 환경**

- **소스 버전 관리 툴과 연동**하여 코드 변경 사항을 감지하고, 자동화 테스트를 포함한 빌드를 수행하여 소프트웨어 품질을 향상시킬 수 있다.
- 자동화 테스트에는 **코딩 표준 준수 여부 체크, 유닛 테스트, 통합 테스트 등을 설정**할 수 있고 테스트 결과에 대한 피드백을 받아 잠재적인 오류를 사전에 예방할 수 있다.
- 빌드 결과물을 지속적으로 배포하여 개발 프로세스 전체를 자동화할 수 있다.

**다양한 활용 및 손쉬운 확장**

- Jenkins는 많이 사용하는 오픈 소스 소프트웨어로 **문서화가 잘 되어 있다.**
- **플러그인을 직접 개발**할 수 있다.

# Jenkins 파이프라인

- **Agent Section**
  - 어떤 Jenkins가 어떤 일을 할지 정의

- **Post Section**
  - 각 스테이지가 끝난 후 후속 조치 설정
    - ex ) Success / failure / always / cleanup
- **Stage Section**
  - 어떤 일들을 처리할 것인지에 대한 Stage 정의 ( 카테고리 )
- **Declaratives**
  - Environment ( 파이프라인 및 stage scope의 환경변수 설정 )
    - Parameter ( 파이프라인 실행시에 받을 파라미터 )
    - Triggers ( 어떤 형태로 트리거 되는지 정의 )
    - When ( 언제 실행되는가 )
- **Steps**
  - 실행할 작업 정의

# Jenkins 플러그인

- Git

  git의 소스코드를 가져와 빌드하는 등 git과의 연동을 위한 플러그인
- Amazon EC2

  EC2 인스턴스와 Jenkins의 연동을 위한 플러그인
- JIRA Plugin

  이슈 추적 도구인 JIRA와 Jenkins를 연동하는 플러그인
- Kubernetes

  쿠버네티스 클러스터에서 동적인 에이전트 실행

등.. 엄청엄청 많습니다