Jenkins

CI/CD ?

`CI(Continuous Intergration)`란 **개발자들이 빠른 주기로 작업한 내용을 통합 브랜치에 통합하고 빌드하는 개발 방식**을 의미한다.

CI는 개인이 작업한 코드를 master 또는 develop 브랜치에 통합하는 과정에서 발생하는 이슈를 빠르게 발견하기 위해서 필요하다.

`CD` 는 `Continuous Delivery` 혹은 `Continuous Depolyment` 두 용어 모두의 축약어이다. Continuous Delivery는 **공유 레포지토리까지 자동으로 Release하는 것**을 의미하며, Continuous Depolyment은 **프로덕션 레벨까지 자동으로 Deploy**하는것을 의미한다.

쉽게 말해서 **개발자가 애플리케이션에 변경 사항을 작성한 후 몇 분 이내에 애플리케이션을 자동으로 실행할 수 있는 것**을 의미한다.

Jenkins ?

> 💡 CI와 CD를 제공하는 툴, CI/CD의 자동화를 지원하는 툴

🚢 Jenkins 설정

1. ec2 ssh 접속

```
ssh -i 키명.pem ubuntu@i9a603.p.ssafy.io
```

2. ec2에 도커 설치(https://everydayyy.tistory.com/121)

```
sudo apt-get upgrade
sudo apt-get update
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
sudo apt update
sudo apt install docker-ce
docker --version
```

3. jenkins 도커 이미지 다운

```
docker pull jenkins/jenkins:lts
```

4. 도커 컨테이너 실행(밑에거 썼음- 나중에 docker not found 오류를 잡기위해)

```
docker run -d -t -p 8080:8080 -v /jenkins:/var/jenkins -v /home/ubuntu/.ssh:/root/.ssh -v /var/run/docker.sock:/var/run/docker.sock --
docker run \
  --name jenkins \
```

```
  -p 8080:8080 -p 50000:50000 \
  -v /home/jenkins:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /usr/bin/docker:/usr/bin/docker \
  -u root \
  -d \
  jenkins/jenkins:lts
```

5. 젠킨스 첫 어드민 키 확인

```
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

6. 젠킨스 플러그인 설치
    a. gitlab
    b. ssh agent

7. gradle 설정(https://gksdudrb922.tistory.com/236)

- gitlab credential 추가
- 설정 - 스프링 gradle 버전 추가
- item 만들기
- 웹훅 연동

▼ **... ec2 프리티어에서 빌드시, 메모리 부족으로 인한 무한 로딩 이슈**

- 싸피에서 주는 ec2를 사용
- swapping 사용하여 메모리 늘리기

AWS EC2 free에서 발생한 메모리 문제 (jenkins + build 배포) - ec2 초기설정

✔️ EC2 Ubuntu 20.04 LTS스토리지 30port 여러개메모리 1GB (free 요금으로 ec2를 이용할 경우 메모리 크기가 1GB이다.) local에서 만든 프로젝트를 가상 서버에 배포하기 위해 ec2 환경을 생성했다.다만, 매번 배포 파일을 만들기 위해

https://velog.io/@chang626/AWS-EC2-free에서-발생한-메모리-문제-jenkins-build-배포

Jenkins

```
BUILD SUCCESSFUL in 1m 14s
7 actionable tasks: 6 executed, 1 up-to-date
Build step 'Invoke Gradle script' changed build result to SUCCESS
Finished: SUCCESS
```

워훙 ~ 빌드 성공루~

8. ssh 설정

- 젠킨스 ssh 서버 등록

Jeinkins/Docker 명령어로 FreeStyle publish-over-ssh 배포하기

두가지 인스턴스가 필요하다.하나는 실제 배포되어서 나오는 worker-instance,나머지느 하나는 jenkins가 설치된 jenkins-instace이다.worker-instance에는 도커를 설치한다.우분투 버전에 도커 설치는 다음 명령어를 따라야 한다.이젠,

https://velog.io/@mooh2jj/젠킨스-자동배포하기

- 빌드후 조치 설정(send build 어쩌고..)

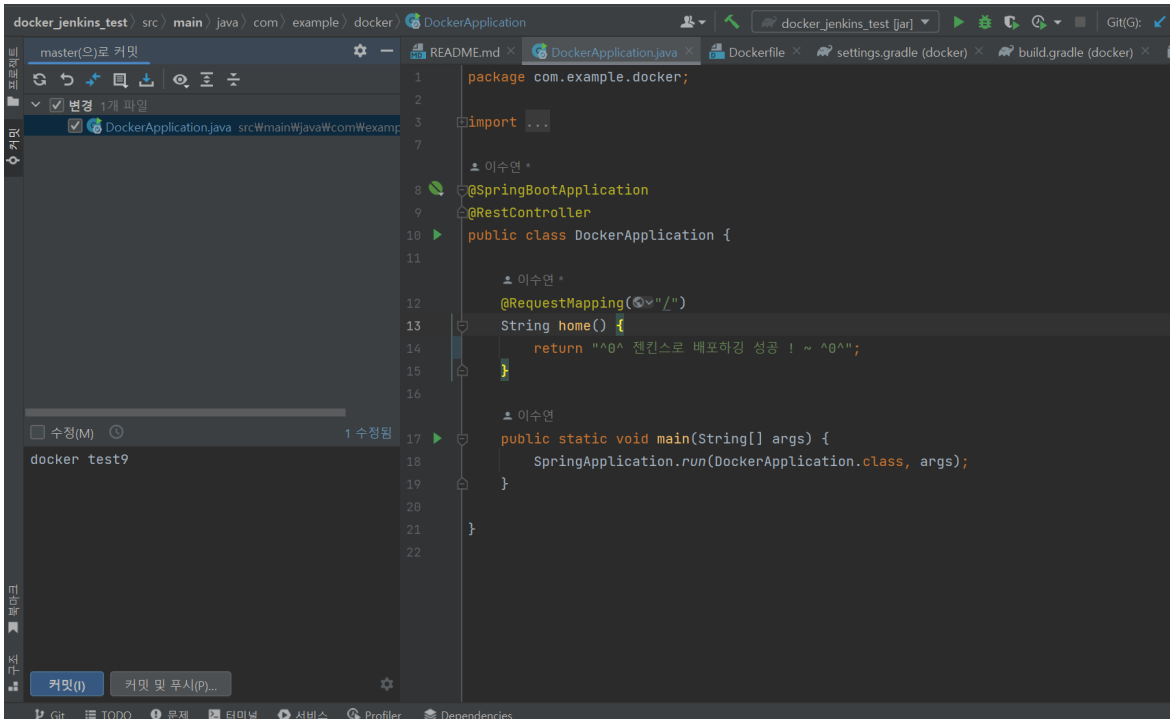
Jeinkins/Docker 명령어로 FreeStyle publish-over-ssh 배포하기 ver 0.2

Spring Boot로 작성한 프로젝트를 Jenkins와 Docker를 활용하여 이미지를 생성하고 그 이미지를 기반으로 AWS EC2에 자동 배포하는 것을 실습

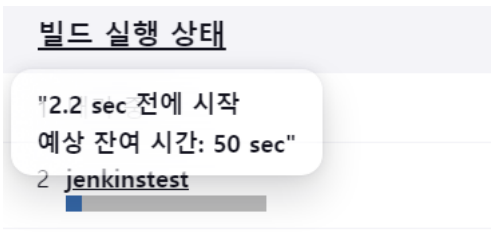
https://velog.io/@mooh2jj/docker-jenkins-github-배포

▼ 결과(잘 되는데 빌드끝나고 잠시 중단 후 배포되는 이슈 - nginx)

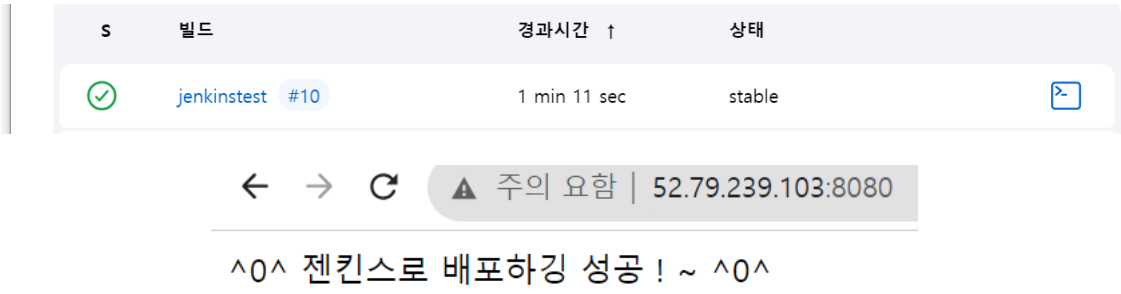
1. 인텔리제이에서 커밋 & 푸시한다.



2. 젠킨스에서 빌드가 시작된다



3. 젠킨스에서 빌드가 성공하면, 웹 브라우저에서 바뀐것을 확인할 수 있다



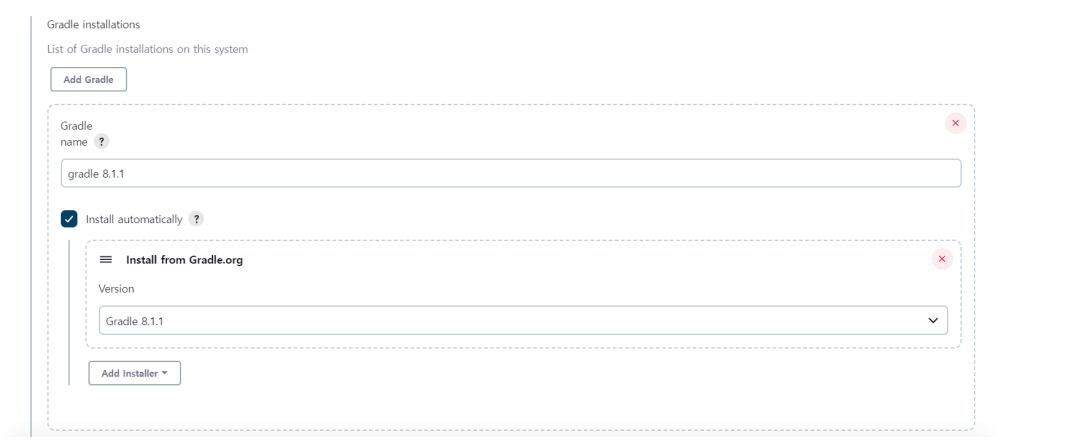
^0^ 젠킨스로 배포하깅 성공 ! ~ ^0^

Pipeline ver.

- 하나의 스크립트를 통해 전체 흐름을 알 수 있기에 더욱 관리가 편하다.
- GUI를 통한 진행과정을 피드백을 받을 수 있다는 장점

젠킨스관리- Tools- gradle 생성, nodejs 생성

- Gradle



- Node js

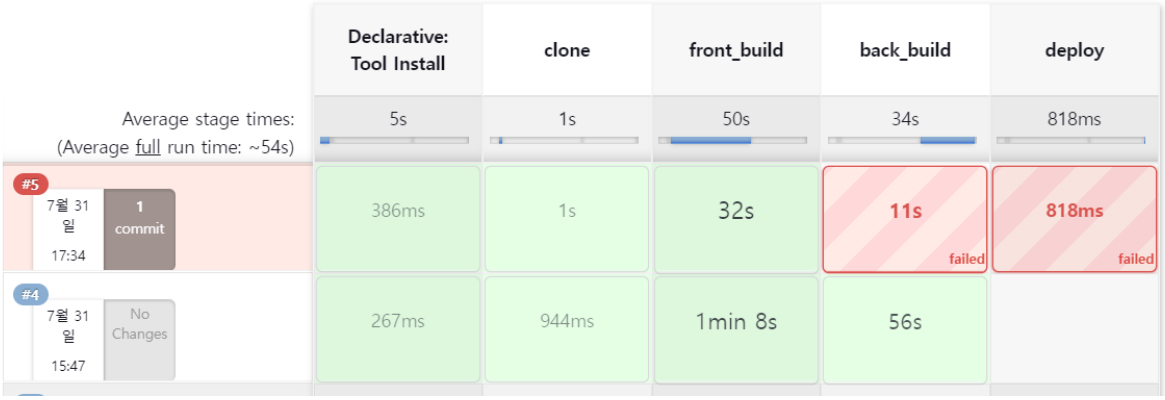
웹훅 생성 - 위와 동일

Pipeline script

```
pipeline {
    agent any

    tools {
        nodejs "nodejs 18.16.1"
        gradle 'gradle 8.1.1'

    }

    stages {
        stage('clone'){
            steps{
                git credentialsId: 'e55ec033-71da-466e-be3b-5d67b55d99e3', url: 'https://lab.ssafy.com/sooyeon990828/jenkinstest.git'
            }
        }
        stage('front_build'){
            steps{
                dir('Frontend'){
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }
        stage('back_build'){
            steps{
                dir('Backend'){
                    dir('Java'){
                        sh 'gradle clean build'
                    }
                }
            }
        }
        stage('deploy'){
            steps{
                sh 'sudo docker-compose up -d --build'
            }
        }
    }

}
```

build 까지 잘된당 ^0^ docker 파일 부분은 좀 다시 만져야 할듯

Stage View



default.conf

```
worker_processes auto;

events { worker_connections 1024; }

http {

  include mime.types;
  sendfile on;

  upstream backend {
    server spring-app:8080;
  }

  upstream frontend {
    server react-app:3000;
  }

  server {
    listen 80;
    server_tokens off;

    location / {
      proxy_pass         http://frontend;
      proxy_redirect     off;
      proxy_set_header   Host $host;
      proxy_set_header   X-Real-IP $remote_addr;
      proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header   X-Forwarded-Host $server_name;
    }

    location /api {
      proxy_pass         http://backend;
      proxy_redirect     off;
      proxy_set_header   Host $host;
      proxy_set_header   X-Real-IP $remote_addr;
      proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header   X-Forwarded-Host $server_name;
    }

  }
}
```

docker-compose.yml

```
version: "3"
services:
  nginx:
    build:
      context: ./Frontend
    ports:
      - 80:80
    depends_on:
```

```
      - spring-app
      - react-app

  spring-app:
    build:
      context: ./Backend
    expose:
      - "8080"

  react-app:
    build:
      context: ./Frontend
    expose:
      - "3000"
```

nginx - Dockerfile

```
FROM nginx:latest

COPY nginx.conf /etc/nginx/nginx.conf

CMD ["nginx", "-g", "daemon off;"]
```

spring-Dockerfile

```
FROM openjdk:11
ARG JAR_FILE=./build/libs/docker-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```