포팅매뉴얼

개발환경

서버 인스턴스 사양

- CPU 정보 : Intel Core i7-9750H CPU @ 2.60GHz 2.59GHz
- 코어개수 : 6
- RAM and Disk :

프로세서 Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

기본 속도:	2.59GHz
소켓:	1
코어:	6
논리 프로세서:	12
가상화:	사용
L1 캐시:	384KB
L2 캐시:	1.5MB
L3 캐시:	12.0MB

```
ubuntu@ip-172-26-1-121:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       311G  121G  191G  39% /
```

설치된 RAM 16.0GB

형상관리

- GitLab

OS

- Window 10
- Ubuntu 20.04.6 LTS

협업 툴

- MatterMost
- Discord

UI/UX

- Figma

기타 편의 툴

- Postman
- Terminus
- Git Window Ver.

이슈 관리

- Jira (이슈 관리)

Back-end

Java	11
Springboot	2.7.14
gradle	8.1.1
Lombok	1.18.24

Front-end

Kotlin	SDK 33
	min SDK 24

Infra

Jenkins	2.416
Docker	24.0.5

DB

	8.0.32

IDE

IntelliJ	2023.1.3 (Ultimate Edition)
Android Studio	Giraffe

배포 과정

- EC2 서버는 구축된 상태

1. SSAFY EC2(서버) 접속

1-1. WSL을 사용하여 EC2에 SSH 연결

- SSH 접속
- 최초 접속 시 권한 요구하면 'yes' 입력

```
# sudo ssh -i [pem키 위치] [접속 계정]@[접속할 도메인]
```

```
$ sudo ssh -i J9B107T.pem ubuntu@j9b107.p.ssafy.io
```

- EC2에 편하게 접속하는 방법 (TIP)
  - EC2 정보가 담긴 config파일을 만들어 번거롭게 pem와 도메인 경로를 쓰지 않고 접속할 수 있다.
  - ssh 전용 폴더 생성

```
  mkdir ~/.ssh
cd ~/.ssh // ssh 폴더 생성 및 이동
cp [로컬 pem 키 위치] ~/.ssh // pem 키 옮기기
vi config  // config 파일 생성
```

  - config 내용 추가

```
Host ssafy
        HostName [서버 ip 주소]
        User ubuntu
        IdentityFile ~/.ssh/[pem키 파일 명].pem
```

  - ssafy 계정에 접속

2. EC2 초기 설정

```
$ sudo apt update                 //시스템의 패키지 정보를 업데이트
$ sudo apt upgrade                //시스템의 패키지 정보들중 업그레이드
$ sudo apt install build-essential  // "build-essential" 패키지를 설치하여서 소프트웨어 개발을 위해 필요한 기본적인 도구와 라이브러리를 포함
//안정성 및 배포에 문제가 생길수있으니 미리 해두는것임.
```

3. 한국으로 시간 설정 (안해도 됨)

```
$ sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime

ln: 링크(link)를 생성하는 명령
-sf: 옵션으로, 심볼릭 링크(symlink)를 생성하고 이미 존재하는 경우에 덮어쓰는 것을 의미
/usr/share/zoneinfo/Asia/Seoul: 심볼릭 링크를 만들 대상 파일로, 이 경우 서울 시간대의 정보가 들어 있는 파일을 가리킴
/etc/localtime: 생성될 심볼릭 링크의 경로로, 시스템의 로컬 시간대 설정을 나타냄

쉽게 말해 링크는 원본 파일의 복사본, 심볼릭 링크는 원본 파일이나 디렉토리를 가리키는 별도의 파일
# 시간 확인
$ date
```

4.EC2 환경 설정

1.Docker 설치

1.기본 설정, 사전 설치

```
$ sudo apt update  //해놨으면 안해도됨~~
$ sudo apt install apt-transport-https ca-certificates curl software-properties-commo //쉽게말하자면 HTTPS를 통한 패키지 저장소 접속!
apt-transport-https 는 HTTPS 프로토콜을 통해 패키지 저장소를 접근하기 위해 필요한 패키지
ca-certificates: 인증서 관련 패키지로, SSL/TLS 연결 시 사용되는 인증서 정보를 포함
curl: 커맨드 라인에서 URL을 통해 데이터를 전송하는데 사용되는 도구
software-properties-common: 소프트웨어 저장소 관련 공통 도구 및 라이브러리 패키지
```

- HTTPS를 통한 패키지 저장소를 접속하기위함
- SSL 인증서 관련 문제를 해결한다.

- 파일 다운로드, API 호출에 사용될수있다
- 저장소 추가, 제거 ,업데이트 등 CRUD에 용이하다.

2. 자동 설치 스크립트 활용

- 리눅스 배포판 종류를 자동으로 인식하여 Docker 패키지를 설치해주는 스크립트를 제공

```
$ sudo wget -qO- https://get.docker.com/ | sh //결국 https://get.docker.com 에 있는 스크립트를 다운받겠다는 의미다.
wget은 웹에서 파일을 받겠다는 의미다.
-qO-은 다운로드된 내용을 화면에 표시하지 않고 직접 출력하라는 의미
|: 파이프(pipe) 기호로, 한 명령의 출력을 다른 명령의 입력으로 전달하는 역할
sh: 다운로드한 스크립트를 실행하는 명령
```

- 결국 Docker를 설치하기위해서 간편하게 제공되는 공식 스크립트를 이용한 것이다.

3. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```
$ sudo systemctl start docker  //이 명령어는 Docker 서비스를 시작하는 명령 start는 Docker 서비스를 수동으로 이용
$ sudo systemctl enable docker //이 명령어는 시스템이 부팅될 때 Docker 서비스가 자동으로 시작되도록 설정, enable 명령은 서비스를 부팅 시 자동으로 시작되도록 등록
```

- 결국은 두 명령어를 순서대로 실행하면 Docker 서비스가 시작되고, 시스템이 부팅될 때마다 Docker 서비스도 함께 자동으로 시작되어 컨테이너 기반의 애플리케이션을 운영할수있다.
  - 젠킨스를 할때 자꾸 Docker가 자동으로 실행되는 것 때문에 오류가 자주났던것같은데, `sudo systemctl enable docker` 이 오류 일수도있겠다는 생각이 든다.

4. Docker 그룹에 현재 계정 추가

```
$ sudo usermod -aG docker ${USER}
$ sudo systemctl restart docker
```

- sudo를 사용하지 않고 docker를 사용할 수 있다.
- docker 그룹은 root 권한과 동일하므로 꼭 필요한 계정만 포함
- 현재 계정에서 로그아웃한 뒤 다시 로그인

5. Docker 설치 확인

```
$ docker -v
```

5. MySQL설치 (본 프로젝트는 MySQL로 진행하였음)

1. Docker MySQL DB 이미지 다운로드 받기

```
$ docker pull mysql
```

2. Docker에 MySQL DB 컨테이너 만들고 실행하기

```
$ docker run --name mysql -d -p 3306:3306 -v --restart=always -e MYSQL_ROOT_PASSWORD=root mysql
```

▼ 옵션 설명

- -v : 마운트 설정, hotst 의 /var/lib/mysql 과 mariadb의 /var/lib/mysql의 파일들을 동기화
- -name: 만들어서 사용할 컨테이너의 이름을 정의
- d: 컨테이너를 백그라운드에서 실행
- p: 호스트와 컨테이너 간의 포트를 연결 (host-port:container-port) // 호스트에서 3306 포트 연결 시 컨테이너 3306 포트로 포워딩
- -restart=always: 도커가 실행되는 경우 항상 컨테이너를 실행
- e: 기타 환경설정(Enviorment)
- MYSQL_ROOT_PASSWORD=root // mariadb의 root 사용자 초기 비밀번호를 설정
- mysql: 컨테이너를 만들 때 사용할 이미지 이름

3. MySQL DB에 database를 추가하고 user 권한 설정

- Docker - MySQL DB컨테이너 접속하기

```
docker exec -it mysql /bin/bash
```

- **MySQL** - 루트 계정으로 데이터베이스 접속하기

```
mysql -u root -p
```

비밀번호는 "root"

MySQL 사용자 추가하기

```
예시) create user 'user_name'@'XXX.XXX.XXX.XXX' identified by 'user_password';

create user 'ssafy601'@'%' identified by 'ssafy601';
```

MySQL - 사용자 권한 부여하기

```
예시) grant all privileges on db_name.* to 'user_name'@'XXX.XXX.XXX.XXX';
flush privileges;

grant all privileges on *.* to 'ssafy601'@'%';
flush privileges;
```

MySQL - 데이터 베이스 만들기

```
예시) create database [db_name];

create database ssafy601;
```

6. EC2에 Jenkins 설치 및 배포 환경 구성

apt-get 업데이트

```
apt-get update
```

JDK설치

JDK 8 이상의 원하는 버전을 설치한다.

```
sudo apt-get install openjdk-11-jdk
```

Jenkins 저장소 키 다운

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -
```

sources.list.d 에 jenkins.list 추가

```
echo deb http://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list
```

Key 등록

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys FCEF32E745F2C3D5
```

apt-get 재 업데이트

```
sudo apt-get update
```

Jenkins 설치

```
sudo apt-get install jenkins
```

Jenkins 서버 포트 번호 변경

생략가능(실제로 생략함)

```
sudo vi /etc/default/jenkins
```

문서가 열리면 HTTP_PORT를 원하는 포트번호로 지정한다.

Jenkins 서비스 재기동

```
sudo service jenkins restart
```

Jenkins 서비스 상태 확인

```
sudo systemctl status jenkins
```

```
ubuntu@jenkins:~$ sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
     Loaded: loaded (/etc/init.d/jenkins; generated)
     Active: active (exited) since Sat 2022-01-08 19:20:01 KST; 17h ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 0 (limit: 1147)
     Memory: 0B
     CGroup: /system.slice/jenkins.service

Jan 08 19:19:58 jenkins systemd[1]: Starting LSB: Start Jenkins at boot time...
Jan 08 19:19:59 jenkins jenkins[439]: Correct java version found
Jan 08 19:19:59 jenkins jenkins[439]:  * Starting Jenkins Automation Server jenkins
Jan 08 19:20:00 jenkins su[692]: (to jenkins) root on none
Jan 08 19:20:00 jenkins su[692]: pam_unix(su-l:session): session opened for user jenkins by (uid=0)
Jan 08 19:20:00 jenkins su[692]: pam_unix(su-l:session): session closed for user jenkins
Jan 08 19:20:01 jenkins jenkins[439]:    ...done.
Jan 08 19:20:01 jenkins systemd[1]: Started LSB: Start Jenkins at boot time.
```

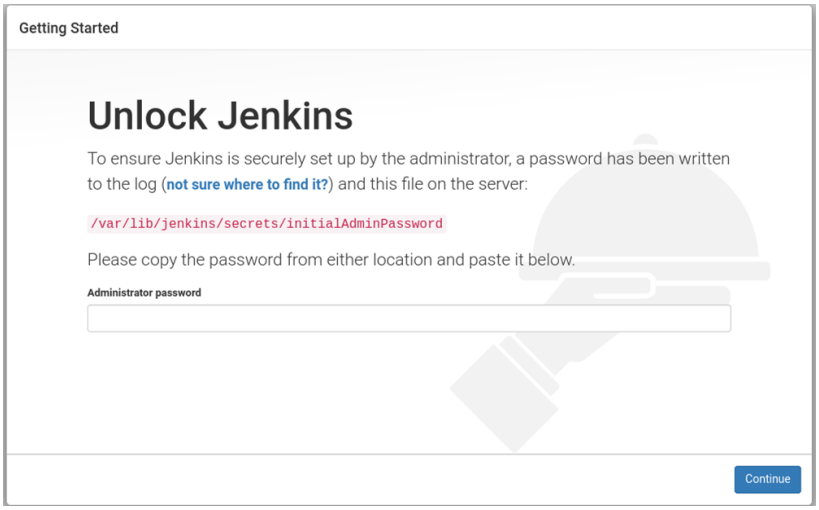
위와 같이 'active' 상태가 되면 성공이다.

Jenkins 초기 비밀번호 확인

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

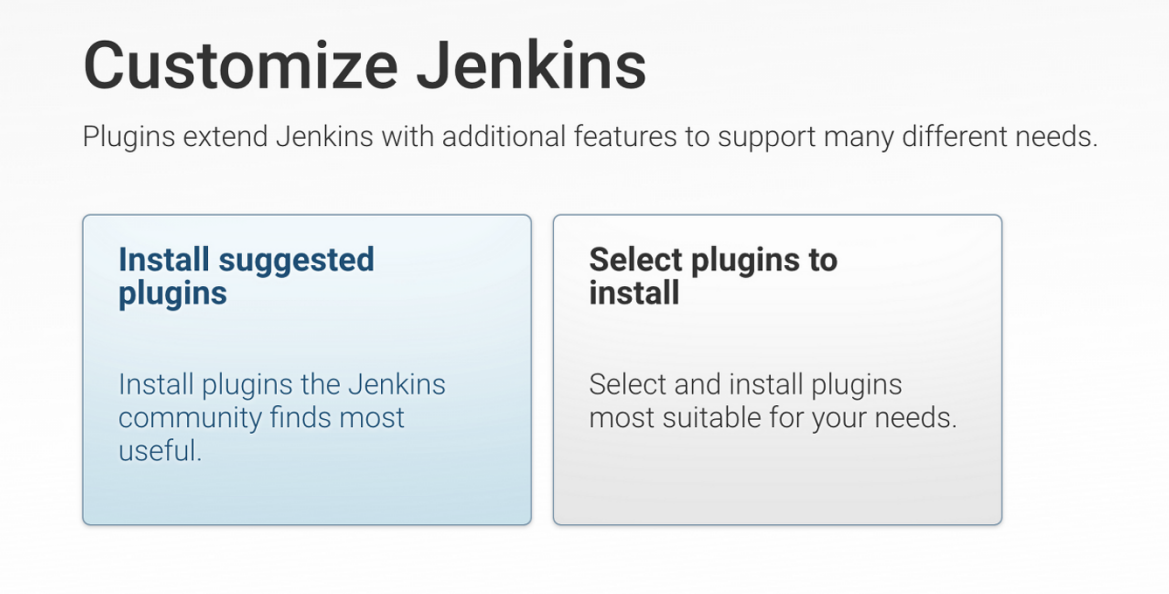
Jenkins 사이트로 이동 후 11번에서 확인한 비밀번호 입력

주소창에 **http://[서버URL]:[포트]** URL을 입력해 젠킨스에 접속한다.



11번에서 확인한 비밀번호를 입력한다.

플러그인 설치



- 정상적으로 입력했다면 플러그인 설치가 나오는데, 우리는 Install suggested plugins를 선택합니다.

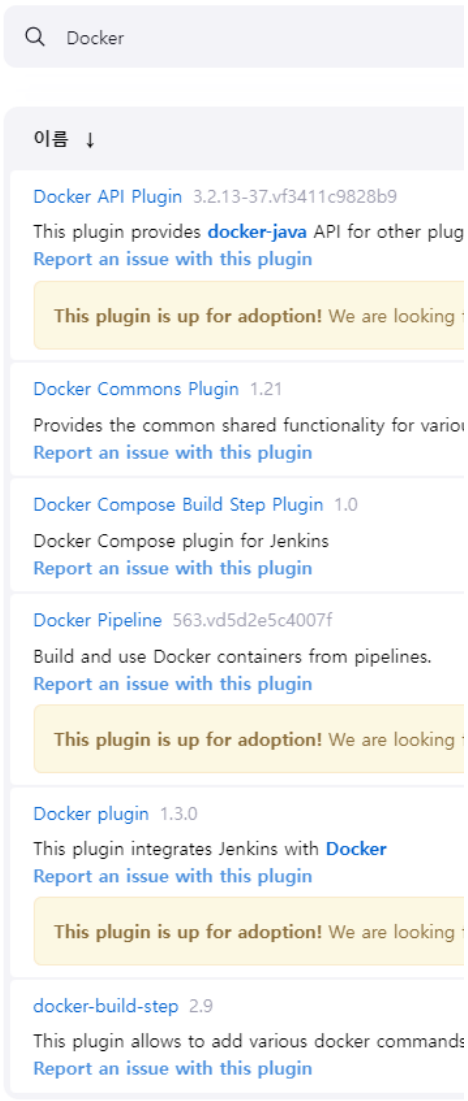
- 설치가 완료되면, 어드민 계정 생성창이 나오고, 본인이 사용하실 정보들을 입력해줍시다.
- 앞으로 이 url로 젠킨스에 접속하시면 됩니다.

Jenkins 플러그인 설정

- Gitlab, Docker 플러그인을 받습니다. (헷갈리다면 비슷한거라도 플러그인 다운받으세요, 너무 다른거 말고)

Gitlab

Docker



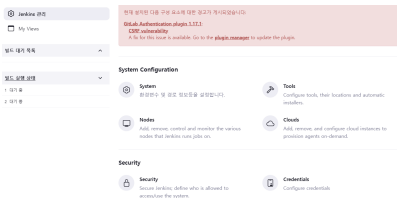
- 여기까지 오셨다면, 젠킨스 설치 및 초기 세팅 완료!

CI/CD (빌드 및 배포) 초기세팅 (먼저 Plugins에서 GitLab이랑 WebHook을다운받는다.)

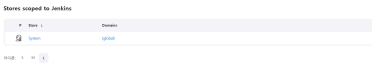
먼저 Jenkins 관리 → Credentials 를 누른다. → 먼저 만드는 이유는?

1. `Credentials를 먼저 만들어서 Jenkins 관리에서 관리하는 것은 보안을 강화하고 민감한 정보를 안전하게 저장하며, 효율적인 관리와 재사용성을 가능하게 하는 중요한 단계`
2. `즉 Credentials를 중앙화하여 관리함으로써 여러 프로젝트 또는 빌드에서 동일한 인증 정보를 사용할 수 있습니다. 변경이 필요한 경우, 한 곳에서 수정하면 모든 사용처에 즉시 적용됨.`

1. 먼저 Jenkins 관리 → Credentials 를 누른다.
2. Add Credentials를 누른다
3. Add Credentials를 누른다

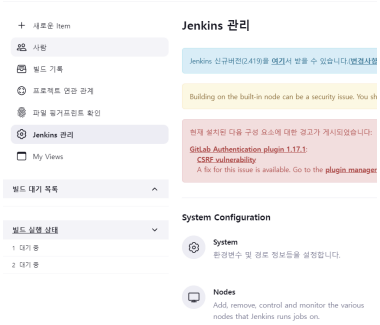


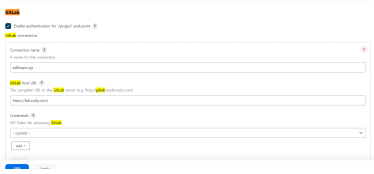
1. Kind는 Username with password 방식
2. UserName은 메일 주소 입력
3. lab.ssafy.com 계정 비밀번호를 입력한다.



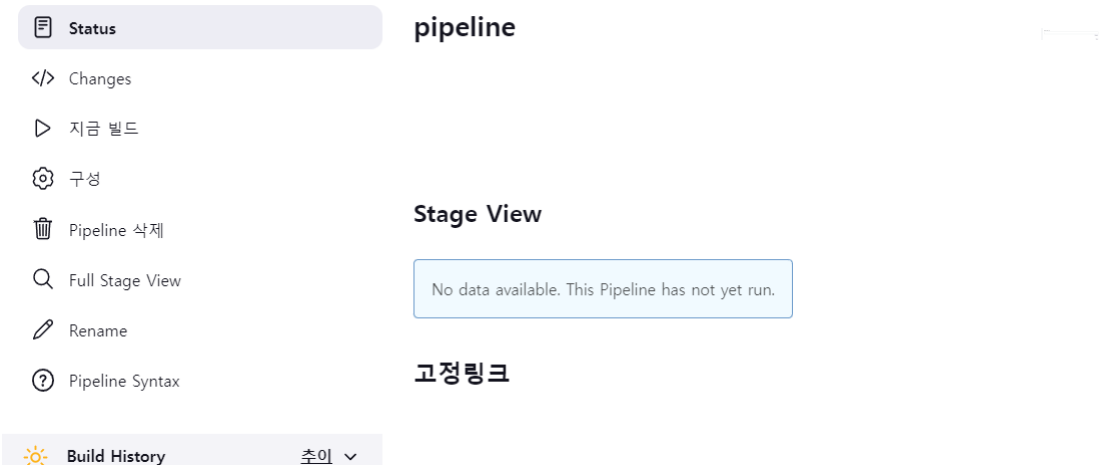
2. GitLab Connection을 하기위해서 Jenkins 관리의 System을 누른다.

3. Connection은 자유
4. GitLab Host URL에서 lab.ssafy.com 입력





1. Pipeline에서 구성을 선택한다.
2. Build when a change is pushed to Gitlab~ 체크
3. WebHook 등록하기위한 Secret Token 생성
4. 이걸로 WebHook으로 자동 감지하게 만들어야함.



Webhook 설정

- Jenkins 트리거 체크

- Jenkins 에서 `빌드 유발` → `Build when …` → `고급` → 하단에 `Secret token Generate` → 토큰 발급 완료!
- Gitlab Webhook에서 해당 토큰을 등록합니다.
  1. lab.ssafy.com Gitlab 프로젝트 접속
  2. 좌측 Settings → Webhook 접속
  3. Jenkins Project URL 입력 후, Secret Token 입력 그리고 Trigger는 원하는 Event에 대해서만 설정
  4. 생성후 Test를 눌렀을 때 200 응답이 return되면 성공

7. CI/CD (빌드 및 배포) 세팅

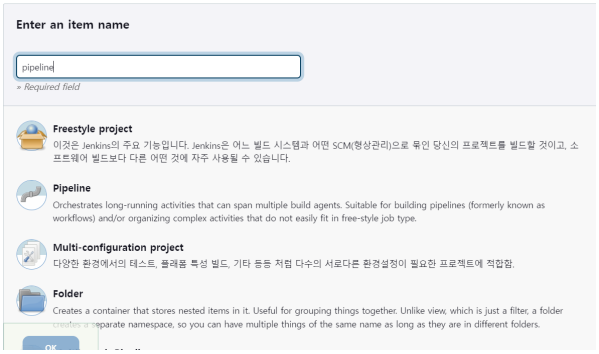
- 먼저, 대쉬보드의 파이프라인 을 클릭합니다.
- FreeStyle Project는 요즘은 많이 안쓴다고 들음… 파이프라인을 많이 쓰는이유는

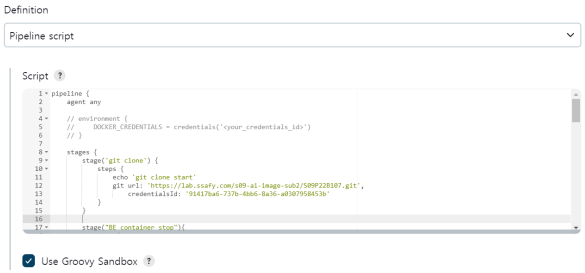
요즘은 코드 기반의 자동화를 선호하며, 파이프라인이 더 널리 사용되고 있습니다.
파이프라인을 사용하면 코드와 인프라 구성을 함께 관리하며, 배포 파이프라인을 수정하거나 공유하기가 더 간편합니다. 라고합니다.

1. 파이프라인 선택

2. SCM 하지않고 pipeline 바로 작성

(파이프라인이 복잡하지 않고 백엔드만 배포하면 되는 상황이어서 해당 방식 채택)





배포 시작

빌드 순서는 JenkinsFile→ Dockerfile

- 젠킨스파일은 젠킨스에만 존재한다.
- 백엔드만 배포된다.
- 도커파일은 프로젝트에 존재한다.

JenkinsFile (stages[큰 묶음] → stage[진짜 실행되는 작은 묶음])

```
pipeline {
    agent any

    // environment {
    //     DOCKER_CREDENTIALS = credentials('<your_credentials_id>')
    // }

    stages {
        stage('git clone') {
            steps {
                echo 'git clone start'
                git url: 'https://lab.ssafy.com/s09-ai-image-sub2/S09P22B107.git',
                    credentialsId: '91417ba6-737b-4bb6-8a36-a0307958453b'
            }
        }

        stage("BE container stop"){
            when {
                expression {
                    return sh(script: 'docker ps -a --format "{{.Names}}" | grep -wq "dct-back"', returnStatus: true) == 0
                }
            }
            steps{
                echo 'BE container stop start'

                sh 'docker stop dct-back'
            }
        }
        stage("BE container remove"){
            when {
                expression {
                    return sh(script: 'docker ps -a --format "{{.Names}}" | grep -wq "dct-back"', returnStatus: true) == 0
                }
            }
            steps{
                echo 'BE container remove start'

                sh 'docker rm dct-back'
```

```
            }
        }
        stage('BE image remove'){
            when {
                expression {
                    return sh(script: 'docker images --format "{{.Repository}}" | grep -wq "dct-back"', returnStatus: true) == 0
                }
            }
            steps{
                echo 'BE image remove'

                sh 'docker rmi dct-back'

                // 안쓰는이미지 -> <none> 태그 이미지 찾아서 삭제함
                sh '''
                    result=$(docker images -f "dangling=true" -q)
                    if [ -n "$result" ]
                    then
                        docker rmi -f $(docker images -f "dangling=true" -q)
                    else
                        echo "No such container images"
                    fi
                '''

            }
        }
        stage('BE new image build'){
            steps{
                echo 'BE new image build'

                // deleteDir() // Clean the workspace

                dir('./BE/DrumComesTrue'){
                //     pwd()
                //     sh 'pwd'

                    sh 'docker build -t dct-back:latest .'
                }

                // sh 'docker build -t dct-back:latest -f Dockerfile.Dockerfile .'
            }
        }
        stage('BE run new image'){
            steps{
                echo 'BE run new image'
                pwd()

                // dir('BE/DrumComesTrue'){
                sh 'docker container run -d -p 8000:8000 --name=dct-back dct-back'
                // }
            }
        }
    }
}
```

DockerFile - BackEnd

```
### multi-stage build ###

### build stage ###
FROM openjdk:11 as builder
ENV WORKSPACE /spring-docker
WORKDIR $WORKSPACE

# copy build & code
COPY build.gradle settings.gradle gradlew .
COPY gradle gradle
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew clean bootJar
#RUN ./gradlew bootJar

### create image stage ###
```

```
FROM openjdk:11

# copy from build stage
ENV WORKSPACE /spring-docker
WORKDIR $WORKSPACE

COPY --from=builder $WORKSPACE/build/libs/*.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

빌더 단계 (as builder):

- FROM openjdk:11 as builder: OpenJDK 11 이미지를 사용하여 빌더 단계를 시작
  필요한 파일과 디렉토리를 복사하고 권한을 설정하여 애플리케이션을 빌드
- ENV WORKSPACE /spring-docker
  WORKDIR $WORKSPACE
  - : 도커 컨테이너 내에서 작업환경을 spring-docker로 변경
    굳이 하지 않아도 되는 작업이지만 분류를 위해 해봄
- COPY build.gradle settings.gradle gradlew . : 빌드 관련 기능을 spring-docker에 복사
  COPY gradle gradle : gradle은 spring-docker/gradle 에 복사 (그냥 gradle 디렉토리는 존재해서 이렇게 spring-docker/gradle 처럼 다른 디렉토리에 넣어주어야 함)
  COPY src src : 빌드에 필요한 소스코드 복사
  RUN chmod +x ./gradlew : gradlew실행 권한 추가
  RUN ./gradlew clean bootJar : Gradle을 사용하여 Spring Boot 애플리케이션의 JAR 파일을 빌드. 이전에 있던 빌드파일 지워버림

실행 단계 (FROM openjdk:11):

- FROM openjdk:11: 런타임 단계에서는 또 다른 OpenJDK 11 이미지를 사용
- COPY --from=builder build/libs/*.jar app.jar: 빌더 단계에서 생성된 JAR 파일을 복사하여 컨테이너 내의 app.jar로 지정된 위치에 배치
- EXPOSE 8080: 컨테이너가 노출하는 포트 번호를 지정.
- ENTRYPOINT ["java", "-jar", "app.jar"] : 도커파일의 실행이 완료되는 시점에 (컨테이너가 run 될 때) app.jar실행

이 Dockerfile은 멀티스테이지 빌드를 활용하여 애플리케이션을 빌드하고 실행하기 위해 두 개의 단계를 사용하고 있습니다. 빌더 단계에서는 애플리케이션을 빌드하고,
실행 단계에서는 빌더 단계에서 생성된 JAR 파일을 컨테이너 내에서 실행합니다. 이렇게 하면 최종 이미지의 크기가 줄어들며, 빌드 과정과 런타임 환경을 분리하여 관리할 수 있습니다.