



목차

1. 개발환경
2. 배포서버 환경 구축(CI/CD)
3. 외부 서비스 정보
4. 프로젝트에 활용되는 프로퍼티 파일



1. 개발 환경

형상관리

- Gitlab

Communication

- Notion
- Mattermost

AR

- Unity
- Unity Ar Foundation

이슈 관리

- Jira

CI/CD

- Jenkins 2.387.1
- Docker 23.0.5

개발 방법

- TDD

OS

- Window 10

UI/UX

- Figma

IDE

- IntelliJ 2022.3.1
- Android Studio : Electric eel

DataBase

- MySQL 8.0.33

Server

- AWS EC2 Ubuntu 20.04.6 LTS
- Tomcat 9.0.71
- Nginx 1.15.12
- S3

기타 편의 툴

- PostMan
- Swagger
- MobaXterm

Front-end 기술 스택

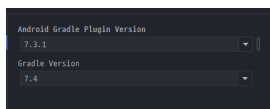
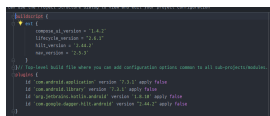
- Kotlin :
- Kotlin Flow
- Gradle : 7.4
- Android
Jetpack
Compose
- Android
Jetpack
- Retrofit
- Dagger-Hilt
- OAuth2.0
- Navigation
- AR
- NFC
- Beacon
- Coil 2.3.0
- OkHttp3

Back-end 기술 스택

- Kotlin
- Gradle 7.6.1
- Spring boot 2.7.1
- Spring Data JPA
- Qu
- Spring Security
- Junit5
- OAuth2.0

AR 기술 스택

- Unity





2. 배포서버 환경 구성(CI/CD)

1. 서버 접속하기

기본 서버 세팅

[우분투 서버의 서버 시간을 한국 표준시로 변경](#)

Server 시간 동기화

[미러 서버를 카카오 서버로 변경](#)

[패키지 목록 업데이트 및 패키지 업데이트](#)

[Swap 영역 할당](#)

[EC2 초기 설정](#)

[Docker 설치](#)

[Docker Compose](#)

[Docker 로 Jenkins 설치](#)

3. Jenkins 컨테이너 실행

[-v /jenkins:/var/jenkins_home * 볼륨 옵션이 중요](#)

[Jenkins OS 재시작 후 자동 실행 설정](#)

[Jenkins 접속](#)

[Jenkins, Gitlab 연동](#)

[GitLab 계정 등록 \(Username with password\)](#)

[GitLab 계정 등록 \(Api Token\)](#)

[Gitlab 커넥션 추가](#)

[GitLabWebHook 설정](#)

[Gitlab Webhook 지정](#)

[Docker Hub 토큰 생성](#)

[Docker Hub 레포지토리 생성](#)

[아이템 추가](#)

[특정 브랜치에 변경 이벤트가 있을 때 마다 소스 코드 복제 하기](#)

[SSH 인증키](#)

[개념](#)

[Git lab과 젠킨스에 등록한다.](#)

[GitLab 에 SSH 공개 키 등록](#)

[Jenkins에 SSH 개인 키 등록](#)

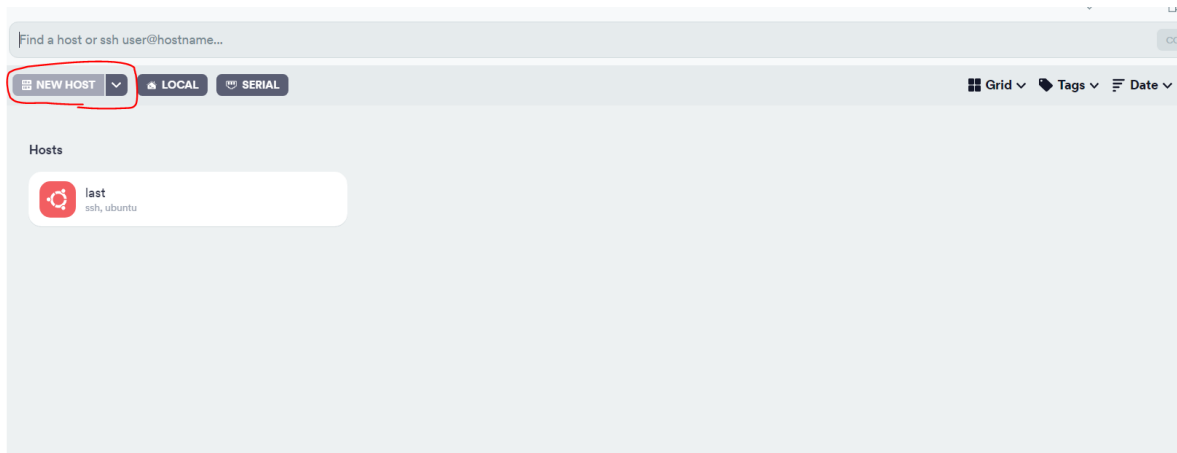
▼ 1. 서버 접속하기

Termius는 SSH, SFTP 같은 다양한 프로토콜을 지원하는 모바일 및 데스크톱 SSH 클라이언트입니다. Termius를 설치합니다.

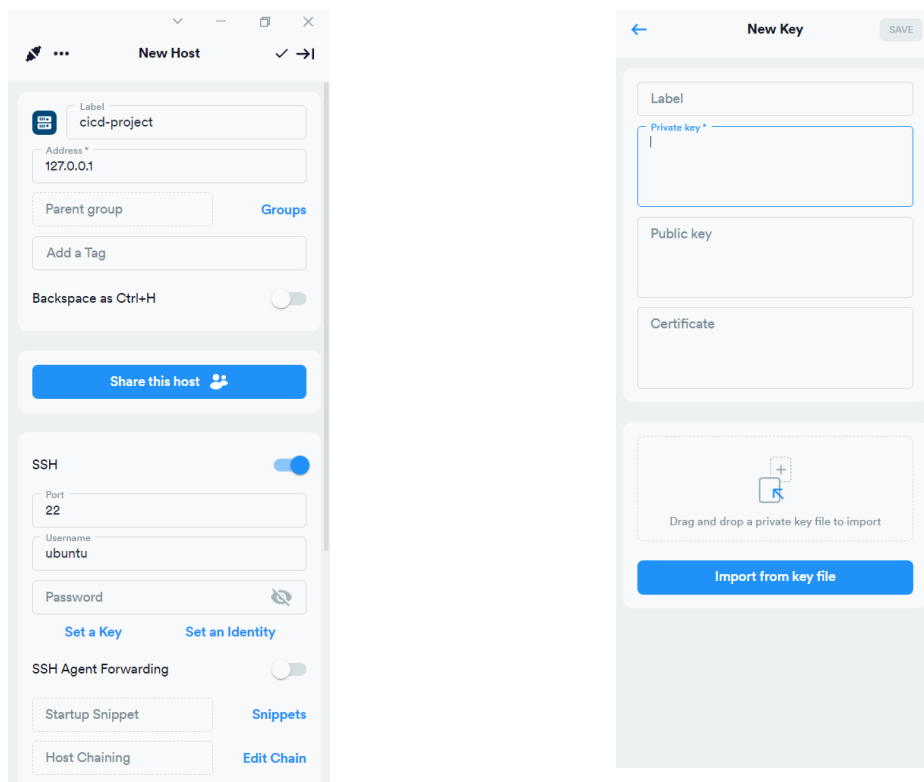
Termius - SSH platform for Mobile and Desktop

Termius helps to organize the work of multiple DevOps and engineering teams. It reduces the admin work for managing users. Enterprise compliance. SOC2 II report.

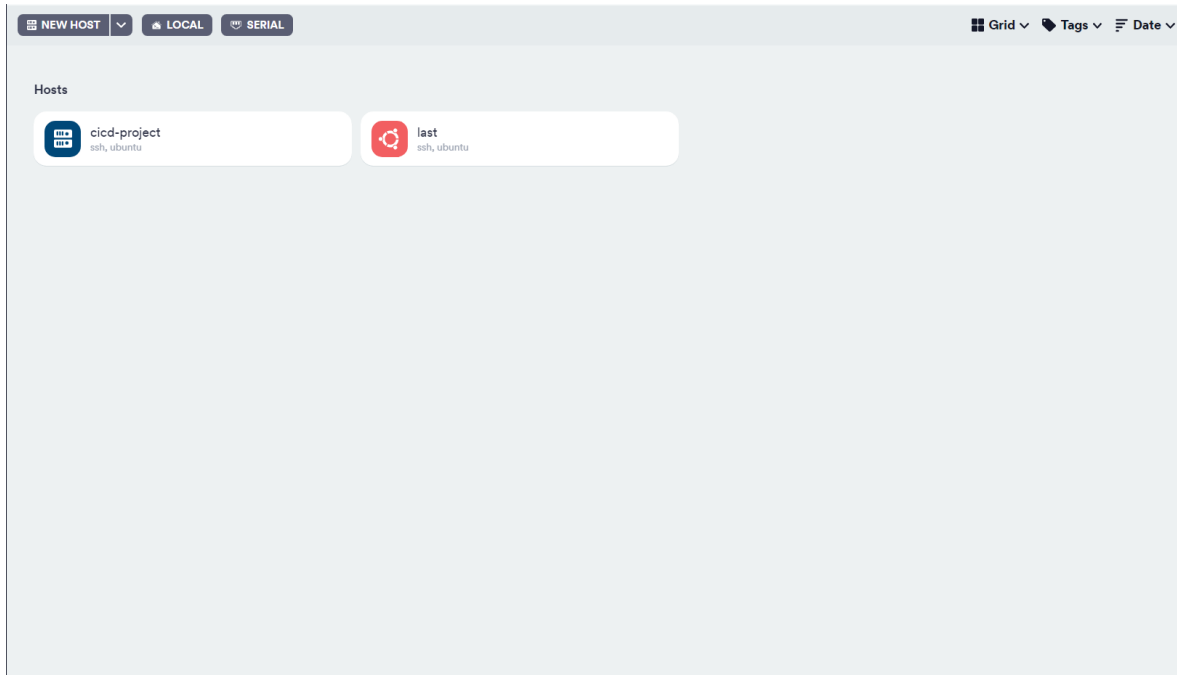
 <https://termius.com/>



terminus를 설치하면 위와 같은 화면이 생성되는데, 상단에 있는 new host를 클릭합니다.



그러면 위와 같은 탭이 우측에 생성됩니다. Address에 서버의 ip주소를 적어줍니다. Port의 경우 기본적으로 SSH는 22번 포트를 사용합니다. 그리고 AWS에서 ubuntu 인스턴스를 생성하면 Username은 ubuntu로 설정되어 있습니다. password는 pem키를 사용할 것인데, password 아래에 있는 **Set a Key**를 클릭하고 **New Key**를 선택하면 오른쪽에 있는 화면이 나옵니다. 여기서 pem파일을 제일 하단의 영역에 드래그한 후 저장해줍니다.



이렇게 새로운 서버가 생성된 것을 확인할 수 있고, 더블 클릭을 통해서 원격 서버에 쉽게 접근할 수 있습니다.

기본 서버 세팅

우분투 서버의 서버 시간을 한국 표준시로 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

Server 시간 동기화

- rdate 서비스 설치

```
sudo apt-get -y install rdate
```

- rdate 서비스 설치
 - 10초정도 기다려도 Sync가 끝나지 않으면 **Ctrl+C**로 강제종료

```
sudo rdate -s time.google.com
```

- 변경된 시간 확인

```
date
```

미러 서버를 카카오 서버로 변경

```
sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/g' /etc/apt/sources.list
```

AWS 의 기본 미러 서버는 해외에 있으니
(해외에 있기 때문에 국내에선 미러시 느리다)

미러 서비스가 필요한 이유

만약 무슨 일이 생겨 APT 패키지를 제공하는 서버가 내려간다면?

사실 그래서 존재하는 것이 미러 (Mirror) 서버다. 원본 서버가 제공하는 정보들을 복제해 다른 경로를 통해 다시 제공하는 것. 꼭 APT가 아니더라도 음악 미러, 동영상 미러 사
참고: <https://kyccfeel.github.io/2019/07/22/Docker%EB%A1%9C-%EC%89%BD%EA%B2%8C-%EC%98%AC%EB%A6%AC%EB%8A%94-%EB%82%98%EB%A7%8C%EC%9D%98-A>

국내 미러 서버 카이스트, 부경대, 카카오 미러 서버 중 카카오 서버로 변경

카카오 미러는 국내 미러들에 비해 속도가 굉장히 빠르고 서버 안정성이 높다는 장점이 있다.

국내 주요 미러(Mirror) 사이트 목록 :: System Administrator's Note (tistory.com).

패키지 목록 업데이트 및 패키지 업데이트

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

```
ubuntu@ip-172-26-14-14:~$ sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/g' /etc/apt/sources.list
ubuntu@ip-172-26-14-14:~$ sudo apt-get -y update && sudo apt-get -y upgrade
Get:1 http://mirror.kakao.com/ubuntu focal InRelease [265 kB]
Get:2 http://mirror.kakao.com/ubuntu focal-updates InRelease [114 kB]
Hit:3 https://download.docker.com/linux/ubuntu focal InRelease
Get:4 http://mirror.kakao.com/ubuntu focal-backports InRelease [108 kB]
Get:5 http://mirror.kakao.com/ubuntu focal/main amd64 Packages [970 kB]
Get:6 http://mirror.kakao.com/ubuntu focal/main Translation-en [506 kB]
Get:7 http://mirror.kakao.com/ubuntu focal/main amd64 c-n-f Metadata [29.5 kB]
Get:8 http://mirror.kakao.com/ubuntu focal/restricted amd64 Packages [22.0 kB]
Get:9 http://mirror.kakao.com/ubuntu focal/restricted Translation-en [6212 B]
Get:10 http://mirror.kakao.com/ubuntu focal/restricted amd64 c-n-f Metadata [392 B]
Get:11 http://mirror.kakao.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:12 http://mirror.kakao.com/ubuntu focal/universe Translation-en [5124 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:14 http://mirror.kakao.com/ubuntu focal/universe amd64 c-n-f Metadata [265 kB]
Get:15 http://mirror.kakao.com/ubuntu focal/multiverse amd64 Packages [144 kB]
Get:16 http://mirror.kakao.com/ubuntu focal/multiverse Translation-en [104 kB]
Get:17 http://mirror.kakao.com/ubuntu focal/multiverse amd64 c-n-f Metadata [9136 B]
Get:18 http://mirror.kakao.com/ubuntu focal-updates/main amd64 Packages [2500 kB]
Get:19 http://mirror.kakao.com/ubuntu focal-updates/main Translation-en [424 kB]
Get:20 http://mirror.kakao.com/ubuntu focal-updates/main amd64 c-n-f Metadata [16.4 kB]
Get:21 http://mirror.kakao.com/ubuntu focal-updates/restricted amd64 Packages [1782 kB]
Get:22 http://mirror.kakao.com/ubuntu focal-updates/restricted Translation-en [250 kB]
Get:23 http://mirror.kakao.com/ubuntu focal-updates/restricted amd64 c-n-f Metadata [636 B]
Get:24 http://mirror.kakao.com/ubuntu focal-updates/universe amd64 Packages [1052 kB]
```

- 패키지 목록 업데이트 도중 다음과 같은 화면이 나오면 **ENTER** 키를 누른다

```
What do you want to do about modified configuration file sshd_config?
```

Package configuration

Configuring openssh-server

A new version (/tmp/fileYuckpN) of configuration file /etc/ssh/sshd_config is available, but the version installed currently has been locally modified.

What do you want to do about modified configuration file sshd_config?

install the package maintainer's version
keep the local version currently installed
show the differences between the versions
show a side-by-side difference between the versions
show a 3-way difference between available versions
do a 3-way merge between available versions
start a new shell to examine the situation

<Ok>

Swap 영역 할당

스왑 파일(swap file) 메모리 할당을 많이 하는 것은 컴퓨터의 전체 성능에 영향을 미칠 수 있습니다. 스왑 파일은 컴퓨터의 물리적인 RAM이 부족할 때 디스크 공간을 일시적인

장점:

더 많은 메모리를 사용할 수 있습니다. RAM이 부족할 때, 스왑 파일을 사용하여 프로그램이 더 많은 메모리를 사용할 수 있게 해줍니다.
프로그램 충돌이나 메모리 관련 오류가 줄어듭니다. 충분한 스왑 파일 공간을 확보함으로써, 시스템이 메모리 부족으로 인한 오류를 경험할 가능성이 줄어듭니다.

단점:

디스크 속도가 느리기 때문에, RAM보다 스왑 파일을 사용하는 것이 성능이 느려질 수 있습니다. 따라서, 스왑 파일을 많이 사용할수록 전체 시스템 성능이 저하될 가능성이 있습니다.
디스크의 수명이 단축될 수 있습니다. 스왑 파일을 많이 사용하면 디스크에 더 많은 읽기 및 쓰기 작업이 발생하므로, 디스크의 수명이 단축될 수 있습니다.
스왑 파일 메모리를 적절하게 할당하는 것이 중요합니다. 일반적으로, 물리적인 RAM의 1.5배에서 2배까지의 스왑 파일 공간을 설정하는 것이 권장됩니다. 하지만, 이는 사용자의 컴

• 용량 확인

```
df -h
```

• swap 영역 할당 (4GB)

```
sudo fallocate -l 4G /swapfile
```

• swapfile 생성

```
sudo mkswap /swapfile
```


- swapfile 활성화

```
sudo swapon /swapfile
```

- System 재부팅 되어도 swap 유지 될 수 있도록 설정

```
sudo echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

- [주의] 명령어가 먹히지 않으면 다음과 같이 수행

- 에디터 열기

```
vi /etc/fstab
```

- 파일 맨 마지막에 명령어 입력 후 저장

```
/swapfile none swap sw 0 0
```

- swap 영역이 할당 되었는지 확인

```
ubuntu@ip-172-26-14-14:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           15Gi        3.6Gi        6.7Gi        0.0Ki        5.3Gi        11Gi
Swap:          4.0Gi          0B        4.0Gi
```

EC2 초기 설정

```
sudo apt update # 우분투 패키지 목록 업데이트
```

```
sudo apt upgrade # 설치된 패키지를 최신 버전으로 업그레이드, 시스템에서 실행되는 소프트웨어 최신화
```

```
sudo apt install build-essential # 기본 개발 도구 설치
```

Docker 설치

1. 기본 설정, 사전 설치

```
sudo apt update
```

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

apt-transport-https: 이 패키지는 HTTPS를 통해 패키지를 전송할 수 있게 해주는 APT 전송 패키지입니다.

일부 패키지 저장소는 HTTPS를 통해 패키지를 제공하므로 이 패키지가 필요합니다.

ca-certificates: 이 패키지는 일반적인 인증 기관(CA)에서 발급한 인증서들의 모음을 제공합니다.

이 인증서들은 시스템이 HTTPS로 패키지를 다운로드할 때 인증서를 확인하고 신뢰할 수 있는 소스에서 패키지가 제공되었는지 확인하는 데 사용됩니다.

curl: 이 패키지는 커맨드 라인에서 URL 문법을 사용하여 데이터를 전송하는 도구입니다.

여러 프로토콜을 지원하며, 패키지 다운로드 및 스크립트 실행 등 다양한 목적으로 사용됩니다.

software-properties-common: 이 패키지는 소프트웨어 패키지 저장소 및 관련 설정을 쉽게 관리할 수 있는 도구를 제공합니다.

이 도구를 사용하면 서드파티 패키지 저장소를 추가하거나 제거할 수 있습니다.

이러한 패키지들은 특히 서드파티 패키지 저장소를 추가하거나, HTTPS를 통해 패키지를 다운로드할 때 필요한 도구 및 라이브러리를 제공합니다.

2. 자동 설치 스크립트 활용

```
sudo wget -qO- https://get.docker.com/ | sh
```

3. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```
sudo systemctl start docker
sudo systemctl enable docker
```

4. Docker 그룹에 현재 계정 추가

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

# sudo를 사용하지 않고 docker 를 사용할 수 있게 해준다.
```

5. Docker 설치 확인

```
docker -v
```

Docker Compose

- what is Docker Compose
 - 여러 개의 컨테이너가 한 애플리케이션으로 동작할 때, 테스트하기 위해 컨테이너를 하나씩 생성해야 한다
 - 여러개의 컨테이너로 구성된 애플리케이션을 구축하기 위해 run 명령어를 여러번 사용해도 되지만, 번거롭다
 - Docker Compose를 이용하면 *.yaml 파일을 이용하여 여러개의 컨테이너 실행을 한 번에 관리하여 하나의 프로젝트처럼 다룰 수 있는 환경을 제공한다

Docker 로 Jenkins 설치

1. Jenkins 이미지 파일 pull(its 버전)

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

2. pull 받은 이미지 확인

```
docker images
```

3. Jenkins 컨테이너 실행

- Jenkins 컨테이너를 명령어를 통해 서비스를 띄운다
 - sudo : 관리자 권한으로 명령어를 실행한다
 - -d : 컨테이너를 **데몬**으로 띄운다
 - -p 8080:8080 : 컨테이너 외부와 내부 포트에 대해 **포워딩** 한다
 - 왼쪽 : Host Port
 - 오른쪽 : Container Port
 - -v /jenkins:/var/jenkins_home : 도커 컨테이너의 데이터는 컨테이너가 종료되면 사라지기 때문에, **볼륨 마운트** 옵션을 이용하여 Jenkins 컨테이너의 /var/jenkins_home 디렉토리를 Host OS의 /jenkins와 연결하여 데이터를 유지한다
 - --name jenkins : 도커 컨테이너의 이름을 설정하는 옵션

- `-u root` : 컨테이너가 실행될 리눅스 사용자 계정 지정 (root)
- `/var/run/docker.sock:/var/run/docker.sock` (정리 필요. sock 으로 통신가능)

```
docker run -d -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul -p 9090:8080 -p 50000:50000 -v /jenkins:/var/jenkins_home -v /home/
//참고 예제
sudo docker run -d --env JENKINS_OPTS="--httpPort=9090" -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul -p 9090:9090 -v /jenkins:/v
```

-v /jenkins:/var/jenkins_home * 볼륨 옵션이 중요

만약 컨테이너 인메모리에 있는 것을 os 저장 시켜서 혹시 컨테이너가 지워지더라도 위의 명령어를 다시 쓰면 원래 컨테이너가 복귀된다
예를 들면 디비나 nginx 들도 해당 방법을 통해 os 저장 시키는 게 좋다. 아니면 다 날라갈 위험 있음

4. Container check

```
docker ps
```

```
ubuntu@ip-172-26-14-14:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  NAMES
2093af08b365   jenkins/jenkins:lts                "/usr/bin/tini -- /u..."  jenkins
tcp, 0.0.0.0:50000->50000/tcp, :::50000->50000/tcp
```

Jenkins OS 재시작 후 자동 실행 설정

- docker-jenkins.service 등록

```
sudo vi /etc/systemd/system/docker-jenkins.service
```

```
# vi /etc/systemd/system/docker-jenkins.service
[Unit]
Description=docker-jenkins
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start jenkins
ExecStop=/usr/bin/docker stop jenkins

[Install]
WantedBy=multi-user.target
```

- docker 서비스 활성화

```
sudo systemctl enable docker
```

- docker 서비스 활성화

```
sudo systemctl start docker
```

- docker-jenkins 서비스 활성화

```
sudo systemctl enable docker-jenkins.service
```

- docker-jenkins 서비스 시작

```
sudo systemctl start docker-jenkins.service
```

작업이 끝났는지 확인 위해서

```
sudo reboot
```

```
// 다시 접속해서  
docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS
560f17936d42	jenkins/jenkins:lts	jenkins	"/usr/bin/tini -- /u..."	2 hours ago	Up 20 seconds

Jenkins 접속

1. k8d201.p.ssafy.io:8080 접속

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

2. 암호 찾기

```
docker logs jenkins
```

```
Jenkins initial setup is required. An admin user has been created and a password generated.  
Please use the following password to proceed to installation:
```

```
2560aa5b2f3a49c4a749300e6efd6f27
```

```
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
```

3. 비밀 번호 활용해 로그인

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

.....

3. Install suggested plugins 로 플러그인 설치

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

4. 회원 가입

Create First Admin User

계정명

암호

암호 확인

이름

Jekins, Gitlab 연동

1. ~~ssh 키 생성 팸키가 있으면 안해도 된다~~

```
ssh-keygen
```

```

ubuntu@ip-172-26-14-14:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
/home/ubuntu/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:xW+1J9QcQoVtkTo6aDbilH0S9coQH0YfPJ9TfSUmGEs ubuntu@ip-172-26-14-14
The key's randomart image is:
+---[RSA 3072]-----+
|    o    E+..==o|
|    . B ..+.ooo*.|
|    + B +.o . = o|
|    = + o . = . |
|    + S . + + . |
|    . = = +   o |
|    + = . .     |
|    . o         |
|    o.          |
+-----[SHA256]-----+

```

이미 설치해서 override 했다

```

+---[RSA 4096]-----+
|    ..    |
|    o o +E.. |
|    * B.=... |
|    . B.@ =o o|
|    . +S*+X.o..o|
|    . o=+o+ . * |
|    ... .o= |
|    .o .. . |
|    ..o.    |
+-----[SHA256]-----+

```

Jenkins Container를 생성할 때 "/home/ubuntu/.ssh:/root/.ssh"로 .ssh 디렉토리를 마운트 해놓았기 때문에 Container 밖에서 ssh 키를 생성하면 Jenkins Container와 연결된다.

EC2에 접속하면 기본 유저가 ubuntu이기 때문에 /home/ubuntu/.ssh에 id_rsa와 id_rsa.pub이 생성된다.

2. Jenkins 플러그인 설치

```

# ssh 커맨드 입력에 사용
SSH Agent

# docker 이미지 생성에 사용
Docker
Docker Commons
Docker Pipeline
Docker API

# 웹훅을 통해 브랜치 merge request 이벤트 발생시 Jenkins 자동 빌드에 사용
Generic Webhook Trigger

# 타사 레포지토리 이용시 사용 (GitLab, Github 등)
GitLab
GitLab API
GitLab Authentication
Github Authentication

# Node.js 빌드시 사용
NodeJS

```

Dashboard > Jenkins 관리 > Plugin Manager

- Updates
- Available plugins**
- Installed plugins
- Advanced settings
- Download progress

Plugins

Search available plugins

Install	Name ↓	Released
<input type="checkbox"/>	Oracle Java SE Development Kit Installer 66.vd8fa_64ee91b_d Allows the Oracle Java SE Development Kit (JDK) to be installed via download from Oracle's website.	5 days 10 hr ago
<input type="checkbox"/>	Command Agent Launcher 90.v669d7ccb_7c31 Agent Management Allows agents to be launched using a specified command.	7 mo 10 days ago
<input type="checkbox"/>	JSch dependency 0.1.55.61.va_e9ee26616e7 Library plugins (for use by other plugins) Miscellaneous Jenkins plugin that brings the JSch library as a plugin dependency, and provides an SSHAuthenticatorFactory for using JSch with the ssh-credentials plugin.	8 mo 25 days ago

이곳에서 플러그인 들을 검색 해서 설치한다.

Download progress

준비

- Checking internet connectivity
- Checking update center connectivity
- Success

SSH Agent	⚠ Downloaded Successfully. Will be activated during the next boot
Authentication Tokens API	⚠ Downloaded Successfully. Will be activated during the next boot
Docker Commons	⚠ Downloaded Successfully. Will be activated during the next boot
Docker API	⚠ Downloaded Successfully. Will be activated during the next boot
Docker	⚠ Downloaded Successfully. Will be activated during the next boot
Docker Commons	✅ 성공
Docker Pipeline	⚠ Downloaded Successfully. Will be activated during the next boot
Docker API	✅ 성공
Generic Webhook Trigger	⚠ Downloaded Successfully. Will be activated during the next boot
Jersey 2 API	⚠ Downloaded Successfully. Will be activated during the next boot
GitLab	⋮ 대기중
GitLab API	⋮ 대기중
GitLab Authentication	⋮ 대기중
GitHub Authentication	⋮ 대기중
Config File Provider	⋮ 대기중
NodeJS	⋮ 대기중

→ [메인 페이지로 돌아가기](#)
(설치된 플러그인을 바로 사용하실 수 있습니다.)

→ ☐ 설치가 끝나고 실행중인 작업이 없으면 Jenkins 재시작.

restart 로 설치 하면 실행 중인 컨테이너가 꺼진다.

```
ubuntu@ip-172-26-14-14:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
2093af08b365   jenkins/jenkins:lts   "/usr/bin/tini -- /u..."   2 hours ago   Exited (5)   37 seconds ago   jenkins
```

```
ubuntu@ip-172-26-14-14:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
2093af08b365   jenkins/jenkins:lts   "/usr/bin/tini -- /u..."   2 hours ago   Exited (5)   37 seconds ago   jenkins
```



```
docker restart ${CONTAINER ID} # docker ps -a 로 컨테이너 id 를 확인하고 도커를 재시작한다.
```



Welcome to Jenkins!

☐ 로그인 상태 유지

로그인

그리고 다시 로그인 하면 된다.

GitLab 계정 등록 (Username with password)

- Jenkins 관리 - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add credentials 클릭
- Add Credentials 클릭
- 정보 입력 후 **Create** 클릭
 - Kind : Username with password 선택
 - Username : Gitlab 계정 아이디 입력
 - Password : Gitlab 계정 비밀번호 입력 (**토큰 발행시, API 토큰 입력**)
 - ID : Credential에 대한 별칭

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
----	------	------	-------------

New credentials

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

g2c1010

☐ Treat username as secret ?

Password ?

.....

ID ?

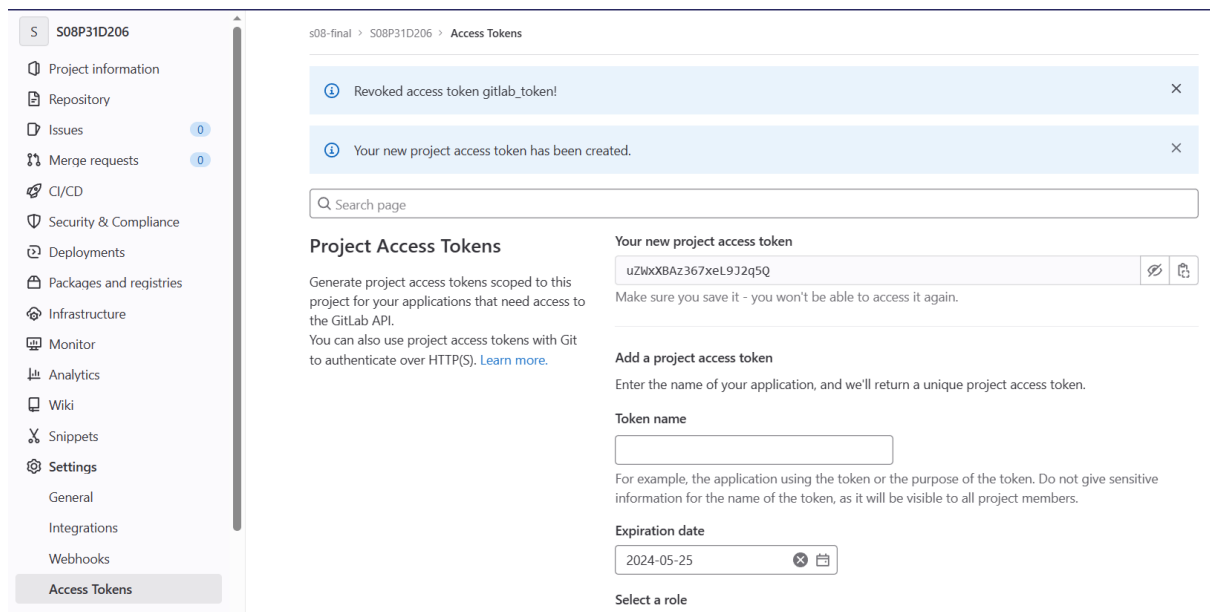
gitlab-rigizer|

Description ?

Create

GitLab 계정 등록 (Api Token)

- Git lab에 가서 토큰을 생성한다. (토큰은 최초 생성시 만 볼수 있으므로 잘 기록해야한다)
- uZWxXBAz367xeL9J2q5Q







- Jenkins 관리 - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add credentials 클릭
- Add Credentials 클릭
- 정보 입력 후 **Create** 클릭
 - Kind : Gitlab API token 선택
 - API tokens : Gitlab 계정 토큰 입력
 - ID : Credential에 대한 별칭

Global credentials (unrestricted)

[+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 gitlab-rigizer	g2c1010/*****	Username with password	
 rigizer-gitlab	GitLab API token	GitLab API token	

최종적으로 이렇게 만들어 진다.

Gitlab 커넥션 추가

- Jenkins 관리 - System Configuration - System 클릭
- 해당 페이지에서 Gitlab을 찾는 다
- Gitlab의 **Enable authentication for '/project' end-point** 체크
 - Connection name : Gitlab 커넥션 이름 지정
 - Gitlab host URL : Gitlab 시스템의 Host 주소 입력
 - Credentials : 조금 전 등록한 **Jenkins Credential (API Token)**을 선택

- 이후, **Test Connection**을 눌러 Success가 뜨면 **저장** 클릭
 - 아니면 입력한 정보를 다시 확인

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

Gitlab connection name required.

Gitlab host URL

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

Gitlab host URL required.

Credentials

API Token for accessing Gitlab

- none -

Add

API Token for Gitlab access required

저장

Apply

성공 시

GitLab connections

Connection name

A name for the connection

rigizer

Gitlab host URL

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

https://lab.ssafy.com

Credentials

API Token for accessing Gitlab

GitLab API token

Add ▾

고급 ▾


Success

Test Connection

실패시

Test Connection

Jenkins 2.387.2



Oops!

A problem occurred while processing the request.

GitLabWebHook 설정

Dashboard > admin > My Views > All > jenkins > Configuration

☐ Cancel pending merge request builds on update

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

e4cfcea7d8dd7bc251245a97e4b7ecc5

Generate

Clear

Generate 를 눌러 Secret token을 발급 받는 다.

e4cfcea7d8dd7bc251245a97e4b7ecc5

Gitlab Webhook 지정

- Gitlab에 특정 브랜치에 merge request가 된 경우 Webhook을 통해 빌드 및 서비스 재배포 이벤트 발동
- Gitlab의 배포할 서비스의 Repository 접속
- Settings - Webhooks 클릭
 - URL : Jenkins의 Item URL 입력 (양식 : `http://[Jenkins Host]:[Jenkins Port]/project/[파이프라인 아이템명]`)
 - Secret token : Jenkins의 Gitlab trigger 고급 설정 중 Secret token Generate 버튼을 이용해 만든 토큰 입력
 - Trigger : Push events 체크, merge request가 되면 Jenkins 이벤트가 발동하게 할 브랜치 입력



Q Search page

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

http://k8d206.p.ssafy.io:7777/project/jenkins

URL must be percent-encoded if it contains one or more special characters.

Secret token

.....

Used to validate received payloads. Sent with the request in the **X-Gitlab-Token** HTTP header.

Trigger

☒ Push events

develop/be

Push to the repository.

SSL verification

☒ Enable SSL verification

Add webhook

Add webhook 을 누르면 생성된다.

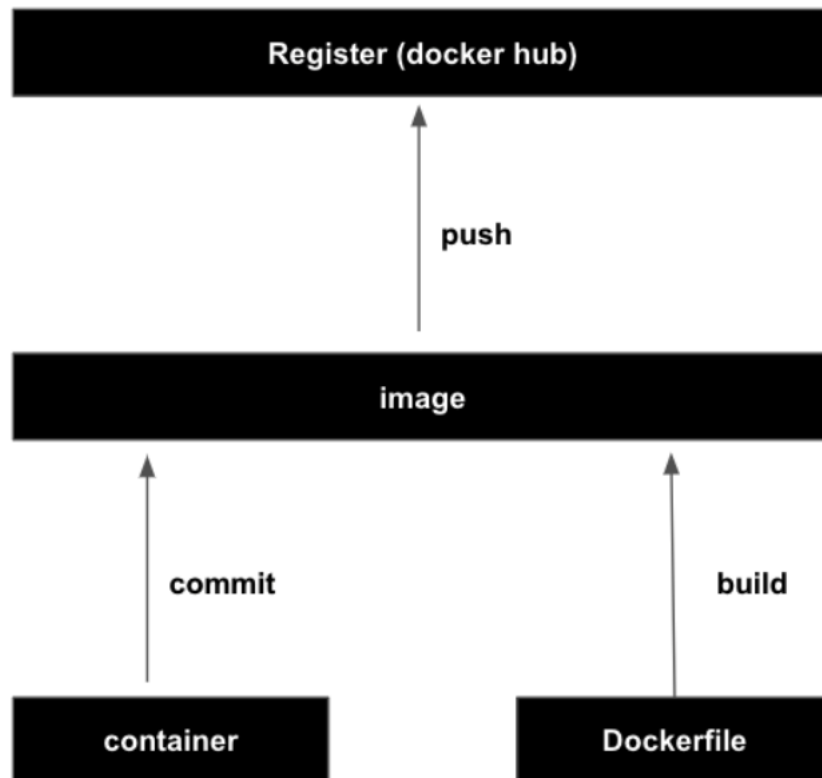
Project Hooks (1)

http://k8d206.p.ssafy.io:7777/project/jenkins

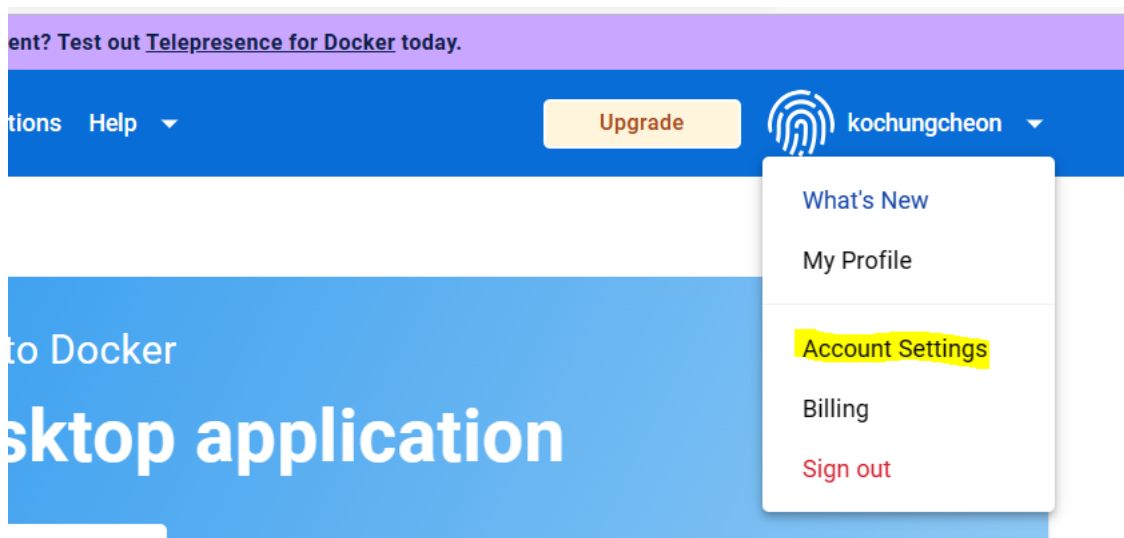
Push Events SSL Verification: enabled

Test Edit Delete

Docker Hub 토큰 생성



Docker Hub



Account Settings

- Security 선택 후 New Access Token 클릭



kochungcheon

User

Joined April 14, 2023

General

Security

Default Privacy

Notifications

Convert Account

Access Tokens

It looks like you have not created any access tokens.
Docker Hub lets you create tokens to authenticate access. Treat personal access tokens as alternatives to your password. [Learn more](#)

New Access Token

- Access Token 이름 지정
 - Access 권한 지정
 - 이후, **Generate** 버튼 클릭

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *

rigizer-token

Access permissions

Read, Write, Delete



Read, Write, Delete tokens allow you to manage your repositories.

- 생성된 Access Token 복사 및 저장
 - 이후, **Copy and Close** 클릭
 - **[주의] Access Token은 단 한 번만 공개되니 반드시 저장바람.** 실수로 지웠다면 삭제 후 재발급

토큰 : dckr_pat_j999NNHlejVMG11dK3S0v7ncwE

Copy Access Token

When logging in from your Docker CLI client, use this token as a password. [Learn more](#)

ACCESS TOKEN DESCRIPTION

rigizer-token

ACCESS PERMISSIONS

Read, Write, Delete

To use the access token from your Docker CLI client:

1. Run `docker login -u kochungcheon`
2. At the password prompt, enter the personal access token.

dckr_pat_MdwQipVDhu0V1bcvEcb88VC4Kxc

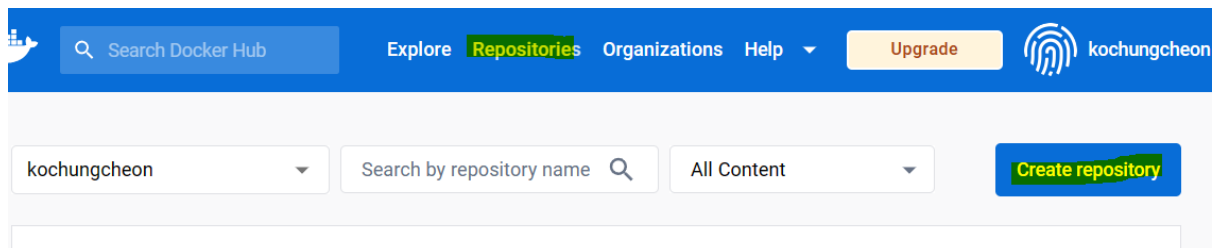


WARNING: This access token will only be displayed once. It will not be stored and cannot be retrieved. Please be sure to save it now.

Copy and Close

Docker Hub 레포지토리 생성

- Repositories - Create repository



- Repository
 - Visibility 지정 (Public 지정, Private는 계정단 한 개만 가능)
 - 이후, **Create** 클릭


Create repository

Namespace	Repository Name *
kochungcheon	d206-backend


Description

Visibility

Using 0 of 1 private repositories. [Get more](#)

☒ Public 

Appears in Docker Hub search results


☐ Private 

Only visible to you

Cancel


Create

- 생성된 Repository 확인

 kochungcheon / d206-backend

Description

This repository does not have a description 

 Last pushed: a few seconds ago

Docker commands

[Public View](#)

To push a new tag to this repository,


```
docker push kochungcheon/d206-backend:tagname
```

Tags

This repository is empty. When it's not empty, you'll see a list of the most recent tags here.

Automated Builds

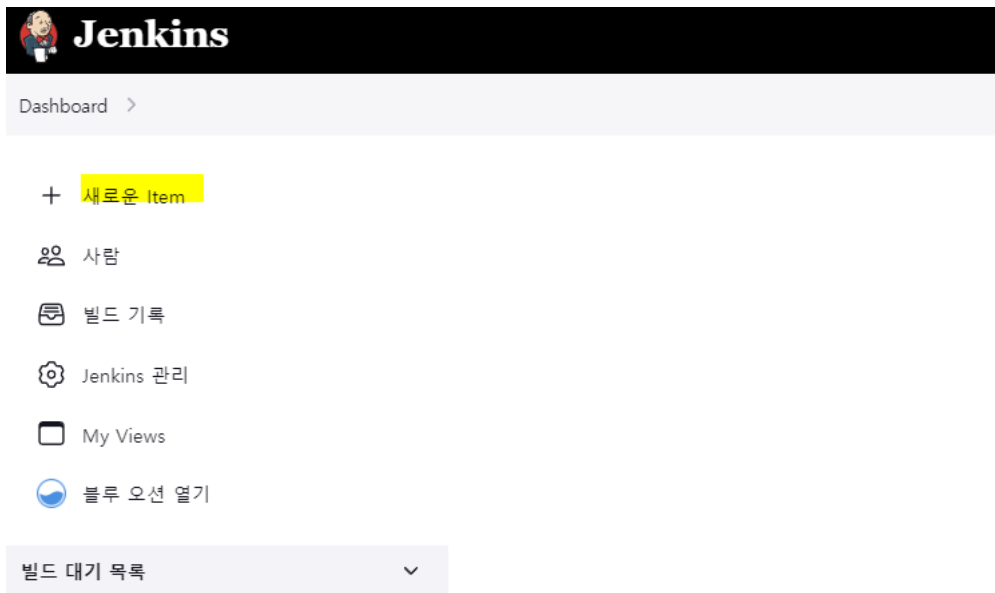
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#) .

Upgrade

아이템 추가

- 새로운 Item 클릭



- 아이템 이름 지정
- Pipeline 클릭

파이프라인 (Pipeline):

파이프라인은 일련의 과정이나 단계가 순차적으로 진행되는 구조를 말합니다. 일반적으로 특정 프로세스를 완료하기 위해 필요한 여러 단계가 연결되어 있습니다.

장점:

순차적인 구조로 인해 각 단계의 출력이 다음 단계의 입력으로 사용되므로 전체 프로세스가 원활하게 진행됩니다.

각 단계를 독립적으로 관리할 수 있어 효율적인 프로세스 관리가 가능합니다.

오류 발생 시 특정 단계를 수정하거나 최적화할 수 있어 유지 보수가 용이합니다.

단점:

파이프라인의 각 단계가 서로 의존적이므로, 한 단계에서 문제가 발생하면 전체 프로세스에 영향을 줄 수 있습니다.

각 단계가 순차적으로 실행되기 때문에 병렬 처리가 어려워 일부 경우에는 시간이 오래 걸릴 수 있습니다.

- OK 클릭

Enter an item name

» This field cannot be empty, please enter a valid name

- Freestyle project**
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로...
- Maven project**
Maven 프로젝트를 빌드합니다. Jenkins은 POM 파일의 이점을 가지고 있고 급격히 설정을...
- Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for build type.
- Multi-configuration project**
다양한 환경에서의 테스트, 플래폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이...
- Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unl name as long as they are in different folders.
- Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

- Configure - General - Gitlab Connection 선택

Configure

General Enabled

설명

[Plain text] [미리보기](#)

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☐ GitHub project

GitLab Connection

rigizer

☐ Use alternative credential

☐ Pipeline speed/durability override ?

☐ Preserve stashes from completed builds ?

☐ Throttle builds ?

☐ 오래된 빌드 삭제 ?

☐ 이 빌드는 매개변수가 있습니다 ?

저장 Apply

- Build Triggers의 **Build when a change is pushed to GitLab** 체크
 - 기본 체크되어 있는 것은 건드리지 않음

Configure

General

Advanced Project Options

Pipeline

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k8d206.p.ssafy.io:8080/project>
 - Enabled GitLab triggers
 - ☒ Push Events
 - ☐ Push Events in case of branch delete
 - ☒ Opened Merge Request Events
 - ☐ Build only if new commits were pushed to Merge Request ?
 - ☐ Accepted Merge Request Events
 - ☐ Closed Merge Request Events
 - Rebuild open Merge Requests
 - Never
 - ☒ Approved Merge Requests (EE-only)
 - ☒ Comments

• Pipeline script 작성

Configure

General

Advanced Project Options

Pipeline

Advanced Project Options

고급

Pipeline

Definition

Pipeline script

Script

1

try sample Pipeline...

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

저장

Apply

• Pipeline script 작성 예시

```
pipeline {
  agent any

  environment {
    imageName = "rigizer/d209-backend"
    registryCredential = 'docker-rigizer'
    dockerImage = ''

    releaseServerAccount = 'ubuntu'
    releaseServerUri = 'j8d209.p.ssafy.io'
    releasePort = '443'
  }
}
```

```

stages {
    stage('Git Clone') {
        steps {
            git credentialsId: 'rigizer', url: 'https://lab.ssafy.com/s08-blockchain-contract-sub2/S08P22D209'
        }
    }

    stage('Jar Build') {
        steps {
            dir ('backend') {
                sh 'chmod +x ./gradlew'
                sh './gradlew clean bootJar'
            }
        }
    }

    stage('Docker Build') {
        steps {
            dir('backend') {
                script {
                    docker.withRegistry('', registryCredential) {
                        sh "docker buildx create --use --name mybuilder"
                        sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ."
                    }
                }
            }
        }
    }

    stage('Remote SSH Before') {
        steps {
            sshagent(credentials: ['ubuntu-d209']) {
                sh '''
                if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker ps -aq --filter ancestor
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker stop $(docker ps -aq --filter ance
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rm -f $(docker ps -aq --filter anc
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rmi rigizer/d209-backend:latest'
                fi
                '''
            }
        }
    }

    stage('Remote SSH After') {
        steps {
            sshagent(credentials: ['ubuntu-d209']) {
                sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'"
                sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker run -i -e TZ=Asia/Seoul -"
            }
        }
    }

    stage('Send Message') {
        steps {
            sh '''
            curl -d '{"text":"Release Complete"}' -H "Content-Type: application/json" -X POST https://meeting.ssafy.com/hooks/actw
            '''
        }
    }
}
}

```

```

pipeline {
    agent any // 이 파이프라인이 어떤 빌드 에이전트에서도 실행될 수 있음을 나타냅니다.
    environment {
        imageName="kochungcheon/d206-backend" // 1
        registryCredential = 'docker-kochungcheon' // 2
        dockerImage = '' // 도커 이미지 변수를 선언, 실제 사용되지 않아 빈문자열로 초기화

        releaseServerAccount = 'ubuntu' // 배포할 원격 서버의 사용자 계정을 지정합니다.
        releaseServerUri = 'k8d206.p.ssafy.io' // 배포할 원격 서버의 URI
        releasePort='443'
    }

    stages {
        stage('Git Clone') { // 파이프라인의 첫 번째 단계로, 이 단계에서는 Git 저장소로부터 소스 코드를 복제합니다.
            steps { // 3
                checkout scmGit(branches: [[name: '*/develop_be']], extensions: [submodule(parentCredentials: true, reference: '', trackingSub
                ) // 이 단계가 완료되면, Jenkins 작업이 실행되는 에이전트에 소스 코드가 복제됩니다. 이렇게 하면 후속 단계에서 빌드, 테스트 및 배포 작업을 수행할 수 있
            }
        }
    }
}

```

```

stage('Jar Build') { // 파이프라인의 두 번째 단계로, 이 단계에서는 프로젝트를 빌드하고 실행 가능한 JAR 파일을 생성합니다.
    steps {
        dir ('backend') { // 작업 디렉토리를 'backend' 폴더로 변경합니다. 이 예제에서는 프로젝트의 백엔드 코드가 'backend' 폴더에 위치해 있기 때문
            sh 'chmod +x ./gradlew' // gradlew 파일에 실행 권한을 부여합니다. Gradle Wrapper를 사용하여 빌드 작업을 수행하기 위해 필요한 작업입
            sh './gradlew clean bootJar' // Gradle Wrapper를 사용하여 빌드 작업을 수행합니다. clean 명령은 이전 빌드 결과물을 삭제하고, bootJ
        }
    }
} // 이 단계가 완료되면 실행 가능한 JAR 파일이 생성되며, 이 파일은 후속 단계에서 Docker 이미지를 빌드하는 데 사용됩니다.
stage('Docker Build') { // 파이프라인의 세 번째 단계로, 이 단계에서는 Docker 이미지를 빌드하고 레지스트리에 푸시합니다.
    steps {
        dir('backend') {
            script { // Docker 명령을 실행하기 위해 Groovy 스크립트를 사용
                docker.withRegistry('', registryCredential) { // 이 명령은 Jenkins에서 미리 설정한 Docker 레지스트리 인증 정보를 사용하여
                    sh "docker buildx create --use --name mybuilder" // Docker BuildKit 기반의 빌더 인스턴스를 생성하고 사용합니다. 빌더
                    sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ." // Docker 이미지를
                                                                // -t $imageName
                }
            }
        }
    }
}
stage('Remote SSH Before') { // 파이프라인의 네 번째 단계로, 이 단계에서는 원격 서버에 SSH를 통해 접속하여 이전에 실행 중인 동일한 Docker 이미지를 중지하
    steps {
        sshagent(credentials: ['kochungcheon-ssh']) { // 이 명령은 Jenkins에서 미리 설정한 SSH 인증 정보를 사용하여 원격 서버에 접속합니다. 여기서
            // 이 블록 내에서는 쉘 스크립트를 작성할 수 있습니다. 이 예제에서는 원격 서버에 SSH 접속하여 명령어를 실행하는 스크립트를 작성합니다.
            // 5
            sh '''
            if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker ps -aq --filter ancestor=koc
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker stop $(docker ps -aq --filter ancestor
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rm -f $(docker ps -aq --filter ancesto
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rmi kochungcheon/d206-backend:latest'
            fi
            '''
        }
    }
}
stage('Remote SSH After') { // 원격 서버에 접속해서 새로운 이미지를 받아 컨테이너를 실행한다.
    steps {
        sshagent(credentials: ['kochungcheon-ssh']) { // Jenkins에서 미리 설정한 SSH 인증 정보를 사용하여 원격 서버에 접속
            // 6
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'"
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker run -i -e TZ=Asia/Seoul -v /h
        }
    }
}
}

```

```

pipeline {
    agent any
    environment {
        imageName = "kochungcheon/d206-backend"
        registryCredential = 'docker-kochungcheon'
        dockerImage = ''
        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'k8d206.p.ssafy.io'
        releasePort = '8000'
    }
    stages {
        stage('Git Clone') {
            steps {
                checkout scmGit(branches: [[name: '/develop_be']], extensions: [submodule(parentCredentials: true, reference: '', tra
            ]
        }
        stage('Jar Build') {
            steps {
                dir('backend') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                }
            }
        }
        stage('Docker Build') {
            steps {
                dir('backend') {
                    script {

```



```

        docker.withRegistry('', registryCredential) {
            sh "docker buildx create --use --name mybuilder"
            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ."
        }
    }
}

stage('Remote SSH Before') {
    steps {
        sshagent(credentials: ['kochungcheon-ssh']) {
            sh '''
            if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker ps -aq --filter ancestor
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker stop $(docker ps -aq --filter ance
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rm -f $(docker ps -aq --filter anc
            ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rmi kochungcheon/d206-backend:late
            fi
            '''
        }
    }
}

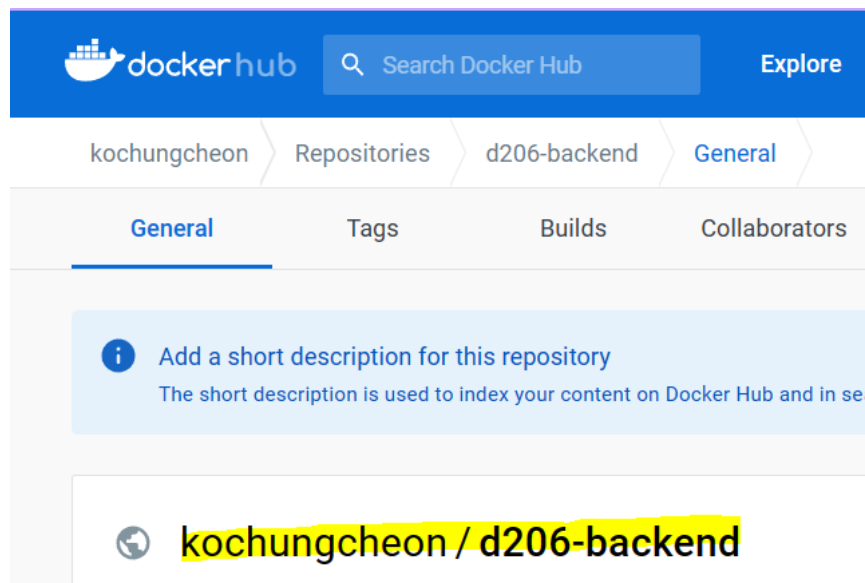
stage('Remote SSH After') {
    steps {
        sshagent(credentials: ['kochungcheon-ssh']) {
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'"
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker run -i -e TZ=Asia/Seoul -"
        }
    }
}
}
}
}

```

파이프라인은 다음과 같은 단계를 거쳐 실행됩니다:

1. 'Git Clone': Git 저장소로부터 소스 코드를 복제합니다.
2. 'Jar Build': 프로젝트를 빌드하고 실행 가능한 JAR 파일을 생성합니다.
3. 'Docker Build': Docker 이미지를 빌드하고 레지스트리에 푸시합니다.
4. 'Remote SSH Before': 원격 서버에 SSH를 통해 접속하여 이전에 실행 중인 동일한 Docker 이미지를 중지하고 제거합니다.
5. 'Remote SSH After': 원격 서버에서 새로운 이미지를 받아 컨테이너를 실행합니다.

// 1



//2

젠킨스에 docker-hub credential을 추가해준다.

Dashboard > Jenkins 관리 > Credentials > System > Global credentials (unrestricted) >

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

Docker hub id 를 입력하세요

☐ Treat username as secret ?

Password ?

ID ?

Jenkins 에서 변수로 사용할 id

! Unacceptable characters


Description ?

Create

//3







특정 브랜치에 변경 이벤트가 있을 때 마다 소스 코드 복제 하기

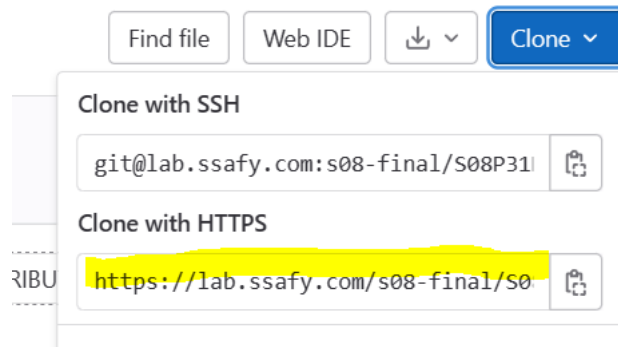
(해당 과정을 통해 특정 브랜치 변경 마다 빌드를 할 수 있습니다.)

 **Jenkins**

Dashboard > Jenkins 관리 > Credentials

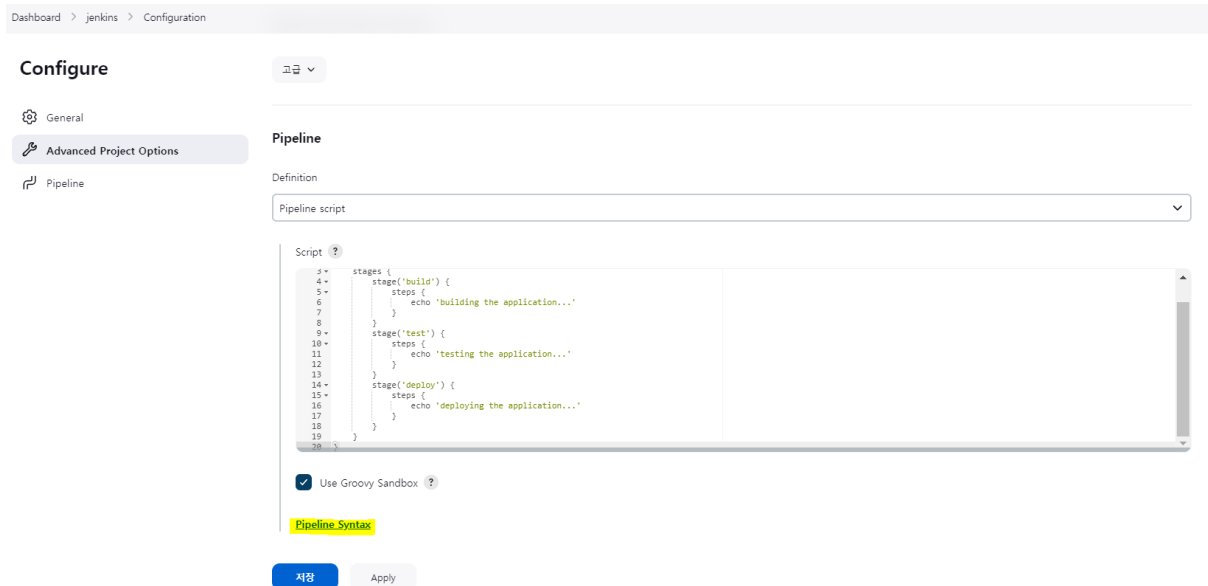
Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	gitlab-rigizer	g2c1010/*****
		System	(global)	rigizer-gitlab	GitLab API token
		System	(global)	docker-kochugcheon	kochugcheon/*****

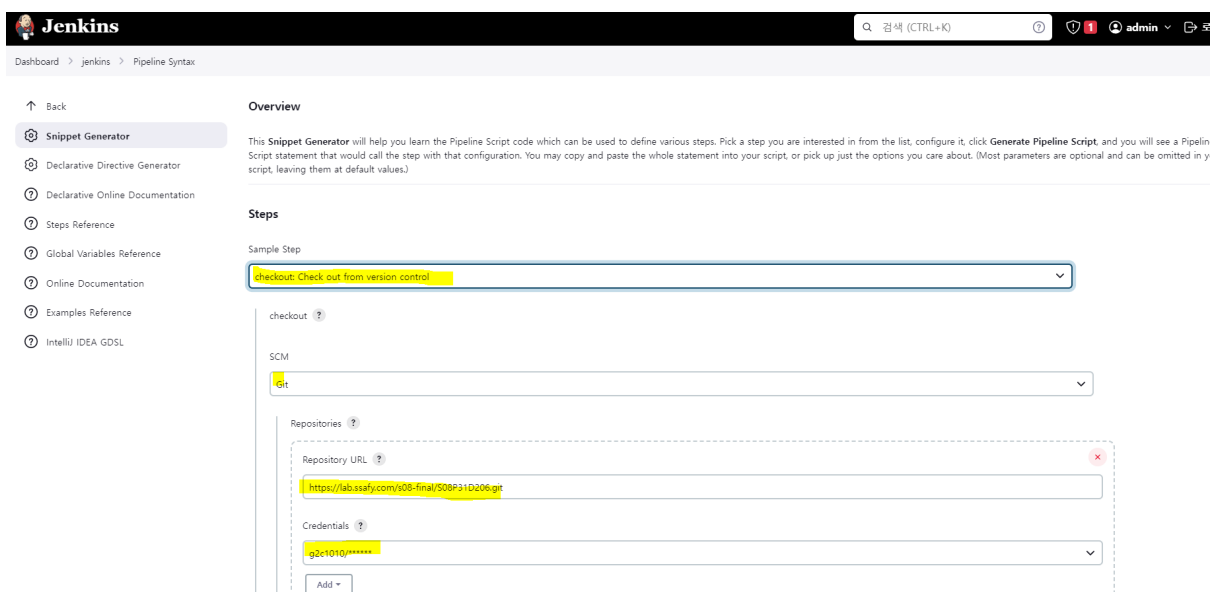


파이프 라인 코드를 만드는 법을 알아 보겠습니다

Dashboard > jenkins > configuration 하단 Pipeline으로 옵니다.



Pipeline Syntax 를 클릭 합니다. (파이프라인 스크립트 생성에 도움을 줍니다.)



깃 주소와 적합한 credential 을 넣어 줍니다.

Branches to build ?

Branch Specifier (blank for 'any') ?

*/*develop_be 1

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Advanced sub-modules behaviours

☐ Disable submodules processing ?

☐ Recursively update submodules ?

☒ Update tracking submodules to tip of branch ? 2

☒ Use credentials from default remote of parent repository ? 3

Generate Pipeline Script

```
checkout scmGit[branches: [[name: "/*develop_be]], extensions: [submodule(parentCredentials: true, reference: "", trackingSubmodules: true)], userRemoteConfigs: [[credentialsId: "gitlab-rigizer", url: "https://lab.ssfy.com/s08-final/S08P31D206.git"]]]
```

1 은 변경 사항이 발생할 때 마다 빌드할 브랜치 명입니다.

2 는 update tracking 하겠다는 것입니다.

3 은 Use credentials from the default remote of the parent project.

☒ Include in polling? ?

☒ Include in changelog? ?

Generate Pipeline Script

```
checkout scmGit[branches: [[name: "/*develop_be]], extensions: [submodule(parentCredentials: true, reference: "", trackingSubmodules: true)], userRemoteConfigs: [[credentialsId: "gitlab-rigizer", url: "https://lab.ssfy.com/s08-final/S08P31D206.git"]]]
```

파란색 버튼을 클릭하면 스크립트가 생성 됩니다.

이 코드를 사용하면 원하는 브랜치 및 서브모듈을 포함하여 Git 저장소를 복제할 수 있습니다.

//4

SSH 인증키

개념

서버를 작성할 때 기존 SSH 키를 선택하거나 SSH 키를 생성해야 합니다. SSH 키는 공개 키 암호화를 통해 사용자나 디바이스를 식별하기 위해 서버에서 사용합니다. SSH 키는 영숫자 조합으로 구성되며 지정되는 디바이스에 대해 고유합니다.

두 키는 아래와 같은 방식으로 작동합니다.

1. 퍼블릭 키는 공개되어 있으므로 누구나 볼 수 있습니다. 이 키를 사용하여 데이터를 암호화할 수 있습니다. 이렇게 암호화된 데이터는 오직 해당 퍼블릭 키와 쌍을 이루는 프라이빗 키를 가진 사람만 복호화할 수 있습니다.
2. 프라이빗 키는 개인이 보관하며, 누구에게도 공개되지 않아야 합니다. 이 키를 사용하여 데이터를 복호화하거나, 디지털 서명을 생성할 수 있습니다. 이렇게 생성된 서명은 해당 프라이빗 키와 쌍을 이루는 퍼블릭 키를 가진 사람만 확인할 수 있습니다.

GitLab에 퍼블릭 키를 등록하면, GitLab은 해당 키를 사용하여 사용자를 인증합니다. 즉, 해당 퍼블릭 키와 쌍을 이루는 프라이빗 키를 가진 사람만 접근이 가능합니다. 이를 통해 사용자가 안전하게 GitLab에 접근할 수 있습니다.

Jenkins에 프라이빗 키를 등록하는 이유는, Jenkins가 GitLab과 통신하거나 원격 서버에 배포 작업을 수행할 때 사용자 인증을 거치기 위함입니다. Jenkins는 등록된 프라이빗 키를 사용하여 암호화된 데이터를 복호화하거나 서명을 생성함으로써, 해당 사용자로서 인증을 진행합니다.

Git lab과 젠킨스에 등록한다.

GitLab에 등록된 공개 키와 연결된 개인 키를 사용하여 Jenkins가 프라이빗 저장소에 접근하기 위해서!

GitLab 에 SSH 공개 키 등록

공개키 출력

```
cat ~/.ssh/id_rsa.pub
```

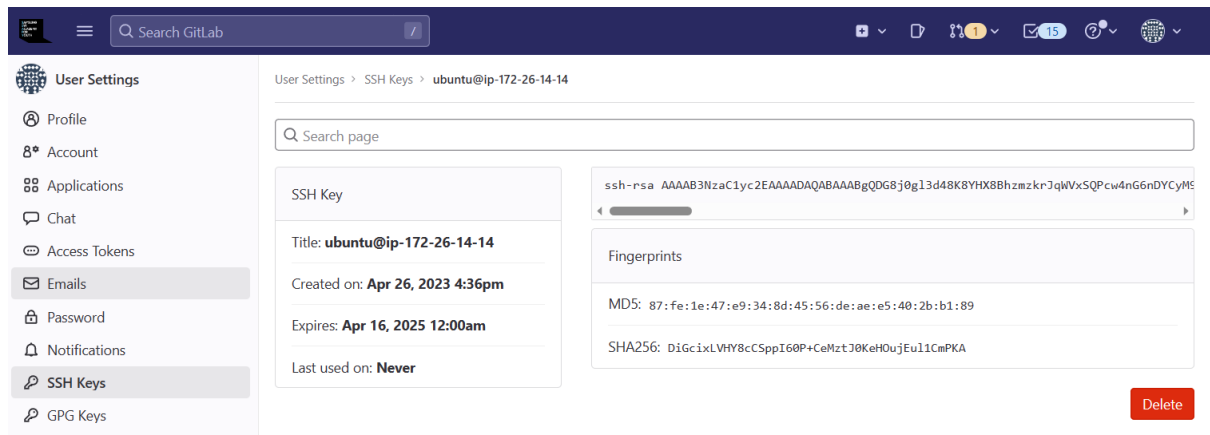
```
ubuntu@ip-172-26-14-14:~$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDG8j0gl3d48K8YHX8BhzmzkrJqWVxSQPcw4nG6nDYCyM99dPdCnyKVu8+r/u0
4zEUKPcMLVtGfp3fu2CtqY4EL60Sjm5bey9ZaXu59SvQg3XxYc6FrJbJkdhFX7iHquvNg84NdDJbtIZEyvS4uckaEYqbjFyj7gE
PVOjNEB/aKhnhPZqILqUwvc0QxiofiGAtk5+ckgUom9Wl0P46S6AiWnX4a/JTVsu/bxwHIzbAtox4GO0tUuBAnVmBkWhcDwr776
XfH4TPMrfu9EtYQU1LI060pyxQ2wiOpM6BP2R+lg1uqDXaITNH83/HFKS0ivnkkwVAA+Kr5kfrY3kvyRi8ByGiFvmTVIF2d1uD7
TyaWWIqkg36h3br6L5w3Sv3vstDRqKggqVc+6krUlhRXP1GmNvJ3aC8ZMCmlhmXZEL5NTh3Zum5NIjQigQ7/UvATziPheONessB
8doqME/bDPAZj/i3iiqoo8lQ+jMuSqKwA9M46zUN9eFdk7HBT3e0C9i0= ubuntu@ip-172-26-14-14
```

퍼블릭키가 출력된다. 이걸 복사 한다.

이제 GitLab으로 간다.

프로필 클릭 > Edit profile 클릭 > SSH Keys 클릭 > 복사한 퍼블릭 키를 3에 복사 > Add Key 클릭

성공적으로 등록 됐다면 아래와 같은 화면을 확인 할 수 있다.



Jenkins에 SSH 개인 키 등록

개인 키 출력

```
cat ~/.ssh/id_rsa
```

커멘드 창에 출력된 개인 키 복사

Jenkins 에 credentials 추가

kind= SSH Username with private key

New credentials

Kind

SSH Username with private key



Scope ?

Global (Jenkins, nodes, items, all child items, etc)



ID ?

credentials-ssh

An internal unique ID by which these credentials are identified from jobs and other configuration. Normally left blank, in which case an ID will be generated, which is fine for jobs created using visual forms. Useful to specify explicitly when using credentials from scripted configuration.

(from [SSH Credentials Plugin](#))

Description ?

Username

kochungcheon-ssh

☐ Treat username as secret ?

Private Key

☒ Enter directly

Key

Enter New Secret Below

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktbjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAABlwAAAAadzC2gtcn
NhAAAAAwEAAQAAAYEAxvI9IJd3ePCvGB1/AYc5s5KvaI1cLkD3M0Jxupw2AsiPfXT3Qb8i
```

Passphrase









Create

Create 를 하면 아래와 같이 만들어진 것을 확인 할 수 있다

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 gitlab-rigizer	g2c1010/*****	Username with password	
 rigizer-gitlab	GitLab API token	GitLab API token	
 docker-kochugcheon	kochugcheon/*****	Username with password	
 credentials-ssh	kochungcheon-ssh	SSH Username with private key	

//5


```

Bash
pipeline {
  agent any // 이 파이프라인이 어떤 빌드 에이전트에서도 실행될 수 있음
  environment {
    imageName="kochungcheon/d206-backend" // 1
    registryCredential = 'docker-kochugcheon' // 2
    dockerImage = '' // 도커 이미지 변수를 선언, 실제 사용되지 않아 빈
  }

  releaseServerAccount = 'ubuntu' // 배포할 원격 서버의 사용자 계정
  releaseServerUri = 'k8d206.p.ssafy.io' // 배포할 원격 서버의 URI
  releasePort='443'
}

```

처음 파이프 라인 작성을 할 때 ImageName 을 설정 해주었다.

그리고 stage('Docker Build') 단계에서 우리는 Docker 이미지를 빌드하고 레지스트리에 푸시했다.

```

stage('Docker Build') { // 파이프라인의 세 번째 단계로, 이 단계에서는 Docker 이미지를 빌드하고 레지스트리에
  steps {
    dir('backend') {
      script { // Docker 명령을 실행하기 위해 Groovy 스크립트를 사용
        docker.withRegistry('', registryCredential) { // 이 명령은 Jenkins에서 미리 설정한 Docker 레지스트리
          sh "docker buildx create --use --name mybuilder" // Docker BuildKit 기반의 빌더
          sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ." // Docker 이미지를 빌드하고 레지스트리에 푸시
        }
      }
    }
  }
}

```

ivy 스크립트를 사용

```

stage('Docker Build') { // 이 명령은 Jenkins에서 미리 설정한 Docker 레지스트리 인증 정보를 사용하여 레지스트리에 로그인
  steps {
    dir('backend') {
      script { // 이 명령은 Jenkins에서 미리 설정한 Docker 레지스트리 인증 정보를 사용하여 레지스트리에 로그인
        docker.withRegistry('', registryCredential) { // 이 명령은 Jenkins에서 미리 설정한 Docker 레지스트리
          sh "docker buildx create --use --name mybuilder" // Docker BuildKit 기반의 빌더 인스턴스를 생성하고 사용합니다. 빌더 인스턴스의 이름은 mybuilder입니다.
          sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ." // Docker 이미지를 빌드하고 레지스트리에 푸시
        }
      }
    }
  }
}

```

Remote SSH Before 단계에서는 이전에 실행되던 동일한 이미지를 기반으로 하는 Docker 컨테이너가 중지되고 제거되며, 이전 이미지도 삭제됩니다.

```

stage('Remote SSH Before') { // 파이프라인의 네 번째 단계로, 이 단계에서는 원격 서버에 SSH
  steps {
    sshagent(credentials: ['kochungcheon-ssh']) { // 이 명령은 Jenkins에서 미리 설정한 SSH 인증 정보를 사용하여 원격 서버에 SSH
      // 이 블록 내에서는 쉘 스크립트를 작성할 수 있습니다. 이 예제에서는 원격 서버에 SSH
      // 5
      sh '''
        if test ""`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest'`; then
          ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker stop $(docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest)'
          ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rm -f $(docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest)'
          ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'docker rmi kochungcheon/d206-backend:latest'
        fi
      '''
    }
  }
}

```

원격 서버에 SSH를 통해 접속하여 이전에 실행 중인 동일한 Docker 이미지를 중지하고 제거합니다.

여기서 미리 설정한 SSH 인증 정보를 사용하여 원격 서버에 접속합니다. 여기서 'kochungcheon-ssh'는 Jenkins에서 미리 설정한 SSH 인증 정보입니다.

원격 서버에 SSH 접속하여 명령어를 실행하는 스크립트를 작성합니다.

```

@releaseServerUri 'docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest'; then
  releaseServerUri 'docker stop $(docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest)'
  releaseServerUri 'docker rm -f $(docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest)'
  releaseServerUri 'docker rmi kochungcheon/d206-backend:latest'

```

셸 스크립트의 내용:

- 먼저, 원격 서버에서 실행 중인 동일한 Docker 이미지가 있는지 확인합니다. `docker ps -aq --filter ancestor=kochungcheon/d206-backend:latest` 명령을 사용하여, kochungcheon/d206-backend:latest 이미지를 기반으로 하는 모든 Docker 컨테이너의 ID를 가져옵니다.
- `if` 문을 사용하여 결과가 있는 경우, 즉 동일한 이미지를 기반으로 하는 컨테이너가 존재하는 경우 아래 작업을 수행합니다.
 - `docker stop` 명령을 사용하여 실행 중인 컨테이너를 중지합니다.
 - `docker rm -f` 명령을 사용하여 해당 컨테이너를 강제로 제거합니다.
 - `docker rmi kochungcheon/d209-backend:latest` 명령을 사용하여 이전에 사용한 kochungcheon/d209-backend:latest 이미지를 원격 서버에서 삭제합니다.

위 작업을 통해 이전에 실행되던 동일한 이미지를 기반으로 하는 Docker 컨테이너가 중지되고 제거되며, 이전 이미지도 삭제됩니다. 이렇게 함으로써 새로운 이미지를 사용하여 Docker 컨테이너를 실행할 준비가 되는 것입니다.

// 6

원격 서버에서 새로운 이미지를 기반으로 하는 Docker 컨테이너가 실행됩니다. 이 단계를 완료하면 새로운 이미지를 사용한 배포가 완료됩니다.

'Remote SSH After' 단계는 다음과 같은 작업을 수행합니다:

1. 원격 서버에 SSH 접속합니다.
2. 원격 서버에서 최신 이미지를 받아옵니다.
3. 받은 이미지를 사용하여 Docker 컨테이너를 실행합니다.

셸 스크립트의 내용:

- 먼저, 원격 서버에서 최신 이미지를 가져옵니다: `ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'` 명령을 사용하여 원격 서버에서 이미지를 받아옵니다.
- 다음으로, 받은 이미지를 사용하여 Docker 컨테이너를 실행합니다:
 - `ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker run -i -e TZ=Asia/Seoul -v /home/ubuntu/images:/home/ubuntu/images -p $releasePort:$releasePort --log-driver=syslog --log-opt syslog-address=udp://logs6.papertrailapp.com:45773 --log-opt tag=SpringBoot -d $imageName:latest'`
 - 이 명령은 원격 서버에서 `docker run` 명령을 사용하여 새로운 이미지를 기반으로 하는 컨테이너를 실행합니다.
 - 옵션들의 설명:
 - `i`: 컨테이너를 상호작용 모드로 실행합니다.
 - `e TZ=Asia/Seoul`: 컨테이너의 환경 변수로 시간대를 설정합니다.
 - `v /home/ubuntu/images:/home/ubuntu/images`: 호스트의 `/home/ubuntu/images` 디렉터리와 컨테이너의 `/home/ubuntu/images` 디렉터리를 연결합니다. 이렇게 하면 컨테이너가 호스트의 파일 시스템에 접근할 수 있습니다.
 - `p $releasePort:$releasePort`: 호스트와 컨테이너의 포트를 연결합니다.
 - `--log-driver=syslog --log-opt syslog-address=udp://logs6.papertrailapp.com:45773 --log-opt tag=SpringBoot`: 로그를 syslog로 전달하고, 로그 서비스 Papertrail로 로그를 전송하는 설정입니다. 로그 태그는 SpringBoot로 설정되어 있습니다.
 - `d`: 컨테이너를 백그라운드에서 실행합니다.
 - `$imageName:latest`: 실행할 Docker 이미지의 이름입니다.



4. 프로젝트에 활용되는 프로퍼티 파일

1. backend

- application.yml

```
spring:
  jpa:
    hibernate:
      ddl-auto: create
      show-sql: true
    properties:
      hibernate:
        format_sql: true
      defer-datasource-initialization: true
  sql:
    init:
      mode: always
  profiles:

    active: aws, swagger, oauth, s3 # develop_be merge : aws 변경, feature/be/개인branch push : local, local-ko

server:
  port: 8000
  servlet:
    context-path: /api

security:
  jwt:
    secret: cs31313dk3gkblcxo1oslaslx421kddkcmvi3412wkkklfcxxiqwj131313dkdkskskvbi1zcHJpbmctYm9vdC1zZW51cm10eS1qd3QtZHV0b3JpYWwK
```

- application-aws.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k8d206.p.ssafy.io:3306/pathfinder?serverTimezone=Asia/Seoul
    username: root
    password: ssafy
```

- application-local.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/pathfinder?serverTimezone=Asia/Seoul
    username: root
    password: doIT@1234
```

- application-local-ko.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/pathfinder?serverTimezone=Asia/Seoul
    username: root
    password: ssafy0103!
```

- application-oauth.yml

```

spring:
  security:
    oauth2:
      client:
        registration:
          naver:
            clientId: es4u8dtxm8g9EwY71G8B
            clientSecret: 0e1T9Eyqec
            redirectUri: http://localhost:3000/callback # 수정 해야함
            authorization-grant-type: authorization_code
            scope: email
            client-name: Naver
        provider:
          naver:
            authorization_uri: https://nid.naver.com/oauth2.0/authorize
            token_uri: https://nid.naver.com/oauth2.0/token
            user-info-uri: https://openapi.naver.com/v1/nid/me
            user_name_attribute: response

```

- application-s3.yml

```

cloud:
  aws:
    s3:
      bucket: d206-buket
    credentials:
      access-key: AKIATDJTP4PXPSKFZFBH
      secret-key: h8AS9pfqbx5WE6zW/GI3qNysWTDYNRb1P+GE4zY
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false

```

- application-swagger.yml

```

spring:
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

```

2. build.gradle.kts

```

import org.jetbrains.kotlin.gradle.tasks.KotlinCompile
import org.jetbrains.kotlin.kapt3.base.Kapt.kapt

plugins {
    id("org.springframework.boot") version "2.7.10"
    id("io.spring.dependency-management") version "1.0.15.RELEASE"
    kotlin("jvm") version "1.6.21"
    kotlin("plugin.spring") version "1.6.21"
    kotlin("plugin.jpa") version "1.6.21"
    kotlin("kapt") version "1.6.21"
}

group = "ssafy.autonomous.passfind"
version = "0.0.1-SNAPSHOT"
java.sourceCompatibility = JavaVersion.VERSION_11

repositories {
    mavenCentral()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-validation")
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("org.jetbrains.kotlin:kotlin-reflect")
}

```

```

// KotlinLogging
implementation("io.github.microutils:kotlin-logging:1.4.3")

// querydsl
implementation("com.querydsl:querydsl-jpa:5.0.0")
kapt("com.querydsl:querydsl-apt:5.0.0:jpa")
kapt("org.springframework.boot:spring-boot-configuration-processor")

// Swagerer
implementation("io.springfox:springfox-boot-starter:3.0.0")

// naver social login
implementation("org.springframework.boot:spring-boot-starter-oauth2-client")
implementation("org.springframework.boot:spring-boot-starter-security")

// jwt
implementation("io.jsonwebtoken:jjwt-api:0.11.2")
runtimeOnly("io.jsonwebtoken:jjwt-impl:0.11.2")
runtimeOnly("io.jsonwebtoken:jjwt-jackson:0.11.2")

// GSON
implementation("com.google.code.gson:gson:2.10.1")
implementation("com.googlecode.json-simple:json-simple:1.1.1")

developmentOnly("org.springframework.boot:spring-boot-devtools")
runtimeOnly("com.mysql:mysql-connector-j")
testImplementation("org.springframework.boot:spring-boot-starter-test")

// testImplementation("org.springframework.security:spring-security-test")

// S3
implementation("org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE")
}

tasks.withType<KotlinCompile> {
    kotlinOptions {
        freeCompilerArgs = listOf("-Xjsr305=strict")
        jvmTarget = "11"
    }
}

tasks.withType<Test> {
    useJUnitPlatform()
}

```

3. Android

- build.gradle (:project)

```

buildscript {
    ext {
        compose_ui_version = '1.4.2'
        lifecycle_version = "2.6.1"
        hilt_version = '2.44.2'
        nav_version = '2.5.3'
    }
}

// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.8.10' apply false
    id 'com.google.dagger.hilt.android' version "2.44.2" apply false
}

```

- build.gradle (:app)

```

plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-kapt'
    id 'kotlin-parcelize'
}

```

```

        id 'com.google.dagger.hilt.android'
    }

    Properties properties = new Properties()
    properties.load(project.rootProject.file('local.properties').newDataInputStream())

    android {
        namespace 'com.dijkstra.pathfinder'
        compileSdk 33

        defaultConfig {
            applicationId "com.dijkstra.pathfinder"
            minSdk 26
            targetSdk 33
            versionCode 1
            versionName "1.0"

            buildConfigField "String", "NAVER_CLIENT_ID", properties['NAVER_CLIENT_ID']
            buildConfigField "String", "NAVER_CLIENT_SECRET", properties['NAVER_CLIENT_SECRET']

            testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
            vectorDrawables {
                useSupportLibrary true
            }
        }

        buildTypes {
            release {
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
            }
        }
        compileOptions {
            sourceCompatibility JavaVersion.VERSION_1_8
            targetCompatibility JavaVersion.VERSION_1_8
        }
        kotlinOptions {
            jvmTarget = '1.8'
        }
        buildFeatures {
            compose true
        }
        composeOptions {
            kotlinCompilerExtensionVersion '1.4.4'
        }
        packagingOptions {
            resources {
                excludes += '/META-INF/{AL2.0,LGPL2.1}'
            }
        }
        kapt {
            correctErrorTypes = true
        }
        viewBinding {
            enabled = true
        }
    }

    dependencies {
        // AR
        implementation files('libs/arcore_client.aar')
        implementation files('libs/ARPresto.aar')
        implementation files('libs/UnityARCore.aar')
        implementation files('libs/unityandroidpermissions.aar')
        implementation files('libs/path_finder.aar')

        implementation 'androidx.core:core-ktx:1.10.0'
        implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.6.1'
        implementation 'androidx.activity:activity-compose:1.7.1'
        implementation "androidx.compose.ui:ui:$compose_ui_version"
        implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
        implementation "androidx.compose.material:material:$compose_ui_version"
        testImplementation 'junit:junit:4.13.2'
        androidTestImplementation 'androidx.test.ext:junit:1.1.5'
        androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
        androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_ui_version"
        debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
        debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"

        // Material Icons
        implementation "androidx.compose.material:material-icons-extended:$compose_ui_version"

        // Material 3
        implementation "androidx.compose.material3:material3:1.1.0-rc01"
        implementation "androidx.compose.material3:material3-window-size-class:1.1.0-rc01"
    }
}

```

```

// Navigation
implementation "androidx.navigation:navigation-compose:$nav_version"

// Coroutines
implementation 'org.jetbrains.kotlin:kotlin-coroutines-core:1.6.4'
implementation 'org.jetbrains.kotlin:kotlin-coroutines-android:1.6.4'
implementation "org.jetbrains.kotlin:kotlin-coroutines-play-services:1.6.4"

// Coroutine Lifecycle Scopes
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1"

//Dagger - Hilt
implementation "com.google.dagger:hilt-android:$hilt_version"
kapt "com.google.dagger:hilt-android-compiler:$hilt_version"
implementation 'androidx.hilt:hilt-navigation-fragment:1.0.0'
implementation "androidx.hilt:hilt-navigation-compose:1.0.0"
kapt "androidx.hilt:hilt-compiler:1.0.0"

//Retrofit
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
//GSON converter
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

//OKHttp3
implementation 'com.squareup.okhttp3:okhttp:4.10.0'
implementation 'com.squareup.okhttp3:logging-interceptor:4.10.0'

//Coil
implementation "io.coil-kt:coil-compose:2.3.0"

//naver
implementation 'com.navercorp.nid:oauth-jdk8:5.4.0' // jdk 8

// accompanist
implementation "com.google.accompanist:accompanist-permissions:0.30.1"
implementation "com.google.accompanist:accompanist-systemuicontroller:0.30.1"

// Permissions
implementation "com.google.accompanist:accompanist-permissions:0.30.1"

//Activity
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'com.google.android.material:material:1.8.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

//framework ktx dependency 추가
// implementation "androidx.fragment:fragment-ktx:1.5.4"
implementation 'androidx.lifecycle:lifecycle-process:2.6.1'
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.4.1'
implementation 'androidx.activity:activity-ktx:1.7.1'

// AltBeacon -> 비컨 라이브러리
implementation 'org.altbeacon:android-beacon-library:2+'
// trilateration -> 비컨 삼각측량 라이브러리
implementation 'com.lemmingapex.trilateration:trilateration:1.0.2'
// Apache Commons Math -> 수학 라이브러리
implementation "org.apache.commons:commons-math3:3.6.1"

// NumberPicker
implementation "com.chargemap.compose:numberpicker:1.0.3"

}buildscript {
    ext {
        compose_ui_version = '1.4.2'
        lifecycle_version = "2.6.1"
        hilt_version = '2.44.2'
        nav_version = '2.5.3'
    }
}

// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.8.10' apply false
    id 'com.google.dagger.hilt.android' version "2.44.2" apply false
}

```

```

| | | |main
| | | |└─java

```