


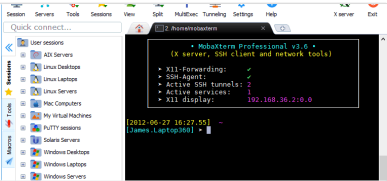
# CI/CD 포팅메뉴얼

## ▼ SSH(MobaXterm) 설치하기

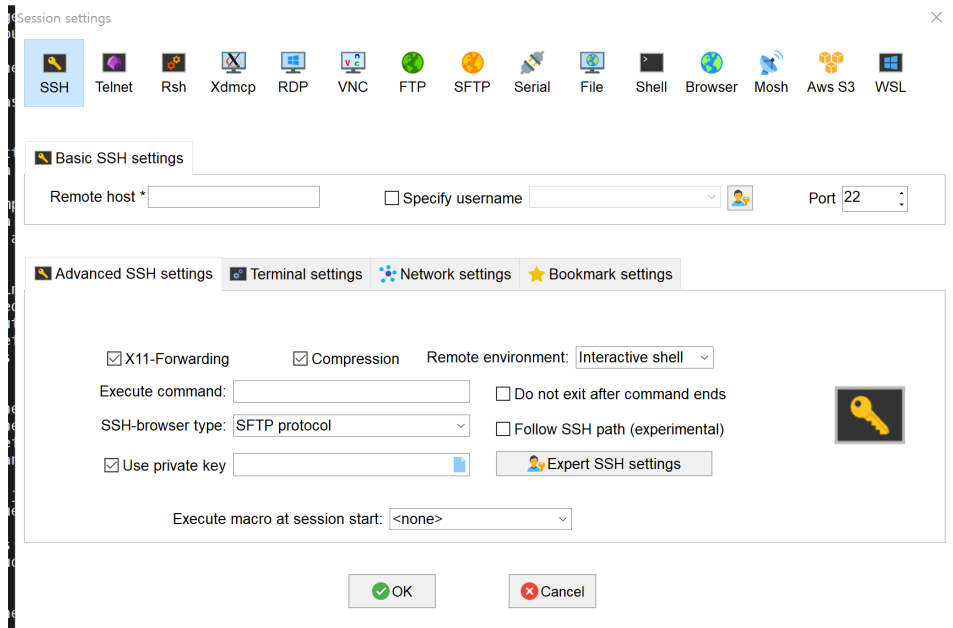
**MobaXterm**

The ultimate toolbox for remote computing - includes X server, enhanced SSH client and much more!

 <https://mobaxterm.mobatek.net/download.html>



- 설치 후 접속하기



Session settings

SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL

**Basic SSH settings**

Remote host \*  ☐ Specify username  Port

**Advanced SSH settings** Terminal settings Network settings Bookmark settings

☒ X11-Forwarding ☒ Compression Remote environment:

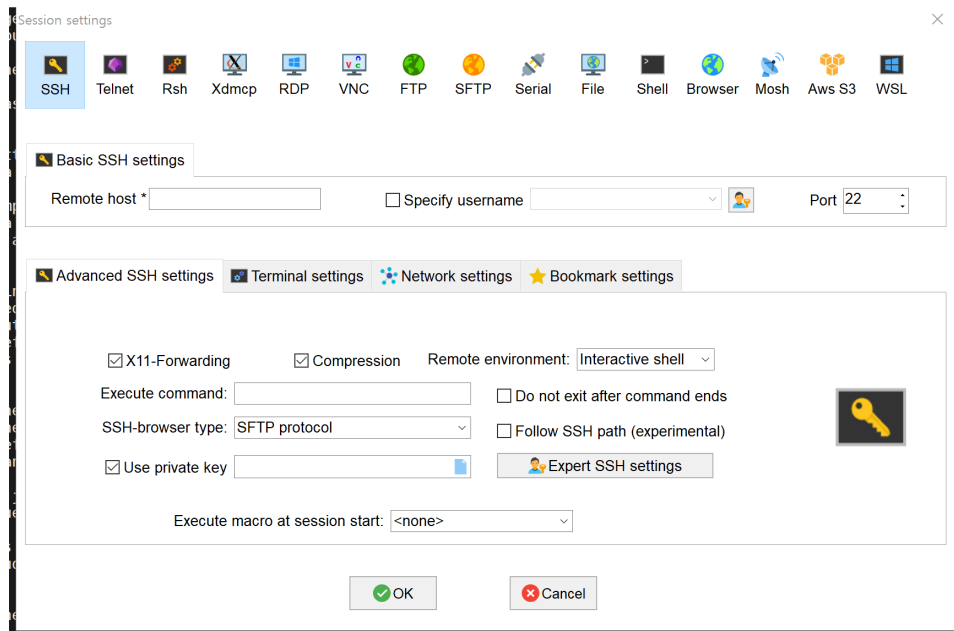
Execute command:  ☐ Do not exit after command ends

SSH-browser type:  ☐ Follow SSH path (experimental)

☒ Use private key

Execute macro at session start:

## ▼ Windows 에서 pem → ppk 로 전환



- puttygen을 다운 받아서 활용
- 주어진 pem key를 ppk로 전환
  - puttygen을 실행 시킨 후 load 를 누른다.
  - pem 파일을 선택한다.
  - save private key를 눌러서 ppk로 저장한다.

## ▼ EC2서버에 JDK 설치하기

```
sudo apt update
sudo apt install openjdk-17-jdk
```

## ▼ Docker 설치하기

- [도커 설치 안내\(공식 홈페이지\)](#).
  - 버전 : 25.0.4
1. 기존 설치되어 충돌이 발생할 수 있는 패키지 삭제

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

2. apt repository 설정

- a. Add Docker's official GPG key : 리눅스 프로그램 설치 시 무결성 검증에 사용됨
- b. Add the repository to Apt sources :

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

c. docker package 설치(최신 버전 설치 기준)

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
```

d. 설치 확인 `sudo docker run hello-world`

```
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
    // 도커 데몬은 이미지, 컨테이너, 네트워크, 볼륨과 같은 도커 객체를 관리하는 백그라운드 서비스
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

## 도커 파일 - AI

```
# Python 애플리케이션을 위한 기본 이미지를 가져옵니다.
FROM python:3.9-slim

# Python 애플리케이션에 필요한 파일을 복사합니다.
WORKDIR /app/python-logic/common/edge_lp3
COPY ./workspace/python-logic/common/edge_lp3/main.py /app/python-logic/common/edge
COPY ./workspace/python-logic/common/edge_lp3/requirements.txt /app/python-logic/cc

# Python 의존성을 설치합니다.
RUN pip install -r /app/python-logic/common/edge_lp3/requirements.txt

CMD ["python3", "main.py"]

# 베이스 이미지로 node와 python을 포함하는 이미지를 사용합니다.
FROM node:21.7.1-alpine

# Node.js 애플리케이션에 필요한 파일을 복사합니다.
WORKDIR /app/express-network
COPY ./workspace/express-network/index.js /app/express-network/index.js
COPY ./workspace/express-network/package.json /app/express-network/package.json

# Node.js 패키지를 설치합니다.
RUN npm install
```

```
# 실행할 명령을 지정합니다.  
CMD ["node", "index.js"]  
# 실행할 명령을 지정합니다.
```

## 도커 파일 - Front

```
FROM node:lts-alpine as builder  
  
WORKDIR /front  
  
ENV PATH /front/node_modules/.bin:$PATH  
  
# COPY package.json /front-edu/package.json  
COPY . .  
  
RUN npm install  
# RUN npm install typescript @types/node @types/react @types/react-dom @types/jest  
  
CMD ["npm", "start"]  
  
EXPOSE 3000
```

## 도커 파일 - Back

```
# 빌드 스테이지  
FROM amazoncorretto:17.0.7-alpine AS builder  
USER root  
WORKDIR /back  
COPY gradlew .  
COPY gradle gradle  
COPY build.gradle .  
COPY settings.gradle .  
COPY src src  
# gradlew 실행 권한 부여  
RUN chmod +x ./gradlew  
RUN ./gradlew bootJar  
  
# 실행 스테이지  
FROM openjdk:17  
WORKDIR /back  
VOLUME /tmp  
COPY --from=builder /back/build/libs/*.jar app.jar  
ENTRYPOINT ["java", "-jar", "/back/app.jar"]  
EXPOSE 8080
```

### ▼ Jenkins container 생성 및 구동

- version: 2.449
- Docker container에 마운트할 볼륨 디렉터리 생성합니다.

```
cd /home/ubuntu && mkdir jenkins-data
```

- 외부에서 접속할 포트를 열고 상태를 확인하기

```
sudo ufw allow *8080*/tcp
sudo ufw reload
sudo ufw status
```

- Docker 명령어로 jenkins container를 생성 및 구동합니다. 해당 image가 없는 경우 아래 로그와 같이 다운로드가 진행되고 이미 있는 경우 생성된 container의 ID만 출력됩니다.

```
sudo docker run -d -p 8081:8080 -v /home/ubuntu/jenkins-data:/var/jenkins_home -v /var
```

- 오류 발생 시( `dial unix /var/run/docker.sock: connect: permission denied` )

- `sudo usermod -aG docker $USER`
- `sudo chmod 666 /var/run/docker.sock`
- `sudo chown root:docker /var/run/docker.sock`

- `-d` : 백그라운드에서 실행 시키겠다는 명령어
- `-v` : 볼륨 마운트할 곳을 지정하는 명령어
- `-p` : 매핑할 포트 번호
- `-name` : 컨테이너 이름을 지정할 명령어
- 이름 뒤에 `jenkins/jenkins:latest` 는 컨테이너를 실행할 때 사용할 이미지

- 구동 상태를 보기 위해 아래 명령어로 로그를 확인합니다. 중간에 출력 되는 초기 패스워드는 별도로 기록해둡니다.

```
sudo docker logs jenkins
```

## ▼ jenkins 환경 설정 변경 - 미리링하기

- jenkins data 폴더로 이동합니다.

```
cd /home/ubuntu/jenkins-data
```

- update center에 필요한 CA 파일을 다운로드합니다.

```
mkdir update-center-rootCA
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt
```

- jenkins의 default 설정에서 특정 미리사이트로 대체하도록 아래 명령어를 실행합니다.(필수)

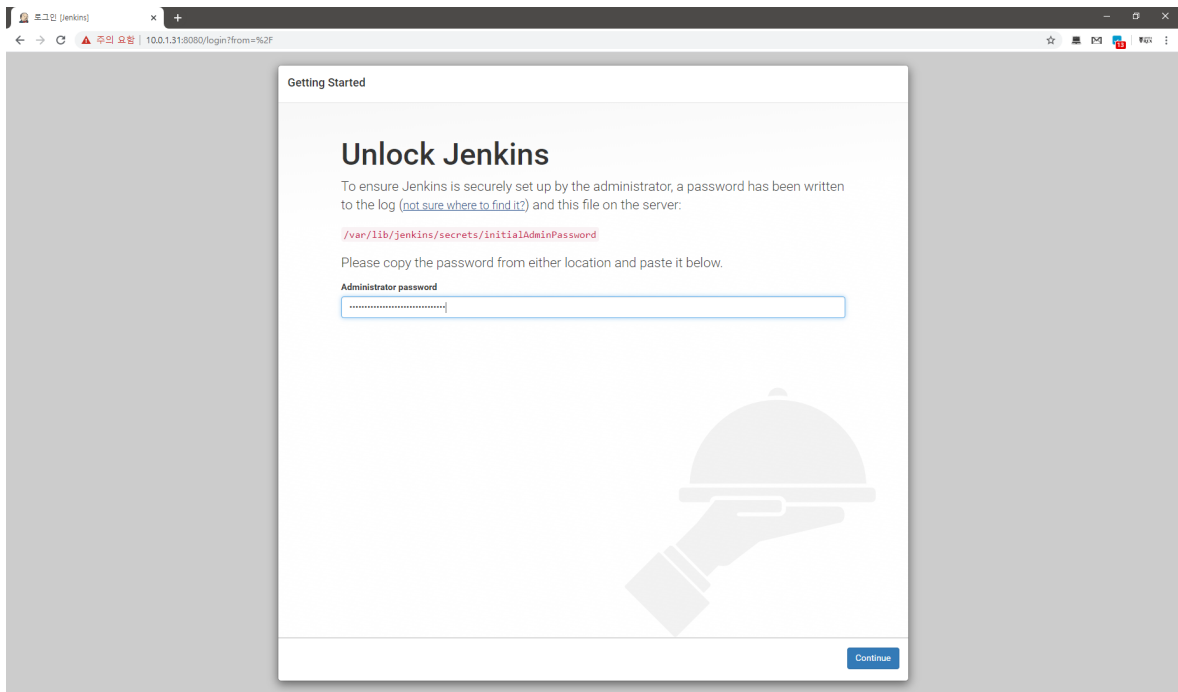
```
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com'
```

- 아래 명령어를 수행 후 `hudson.model.UpdateCenter.xml` 파일을 열어 url이 변경됐는지 확인합니다.

다음 화면에서 저장해둔 초기 패스워드 입력

## ▼ jenkins 초기 설정 진행

`http://<EC2 도메인>:8080` 으로 접속

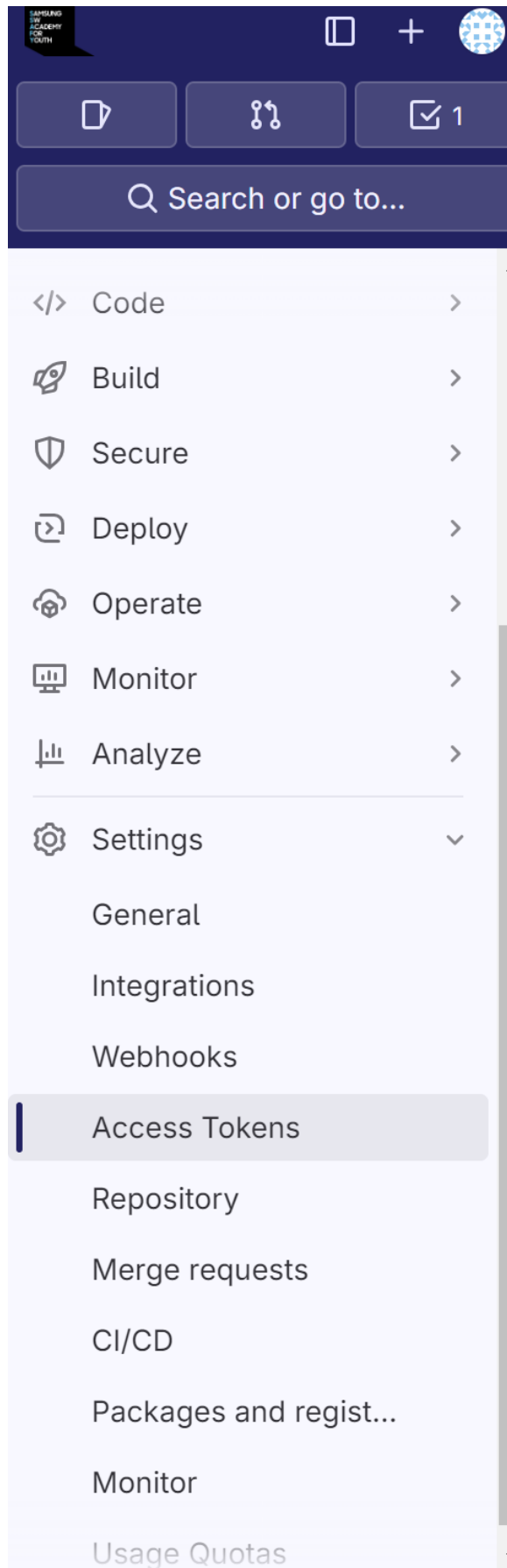


## ▼ Jenkins plugin

- gitlab
- blue-ocean
- Docker
- pipeline stage view, pipeline rest api

## ▼ jenkins Credentials 등록 - GitLab

T	P	Store ↓	Domain	ID	Name
		System	(global)	gitlab-jenkins-token	haeisuin/***** (개인 access token)
		System	(global)	Docker-hub	hyeiin/***** (docker token)



**Project Access Tokens**

**Add a project access token**

Token name

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

Expiration date

2024-05-03

Select a role

Guest

Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☒ api  
Grants complete read and write access to the scoped project API, including the container registry, the dependency proxy, and the package registry.
- ☒ read\_api  
Grants read access to the scoped project API, including the Package Registry.
- ☐ create\_runner  
Grants create access to the runners.
- ☐ k8s\_proxy  
Grants permission to perform Kubernetes API calls using the agent for Kubernetes.
- ☒ read\_repository  
Grants read access (pull) to the repository.
- ☐ write\_repository  
Grants read and write access (pull and push) to the repository.
- ☐ ai\_features  
Grants access to GitLab Duo related API endpoints.

Scope

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

Username

haetsuin

☐ Treat username as secret

Password

Concealed

Change Password

ID

gitlab-jenkins-token

Description

기타 access token

Username = 깃랩아이디 or docker-hub 아이디

**Project Access Tokens**

Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTPS. [Learn more.](#)

Your new project access token

.....

Make sure you save it - you won't be able to access it again.

Password = 생성 시 만들어진 초기 비밀번호(깃랩에서 받은 시크릿 키)

## ▼ jenkins Credentials 등록 - Docker

**docker** Explore Repositories Organizations Search Docker Hub

Account Settings / Security

**donibari**

General

Security

Default Privacy

Notifications

Convert Account

Deactivate Account

**Access Tokens**

New Access Token

Description	Source	Scope	Last Used	Created
Docker-hub	DOCKER	Read, Write, Delete	Feb 16, 2024 10:43:24	Feb 12, 2024 16:57:41

**Two-Factor Authentication**

Two-factor authentication is not enabled yet.

Two-factor authentication adds an extra layer of security to your account by requiring more than just a password to sign in. [Learn more](#)

Enable Two-Factor Authentication

방법은 깃랩과 동일

## ▼ jenkins-pipeline 설정



Dashboard > BE > Configuration

Configure

General

Advanced Project Options

Pipeline

오래된 빌드 삭제 ?

Strategy

Log Rotation

빌드 이력 유지 기간(일)

공백일 경우, [보관할 최대개수] 만큼 기록됩니다.

보관할 최대개수

if not empty, only up to this number of build records are kept

20

고급

이 빌드는 매개변수가 있습니다 ?

Build Triggers

Build after other projects are built ?

Projects to watch

No project specified

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Always trigger, even if the build is aborted

Dashboard > BE > Configuration

Configure

General

Advanced Project Options

Pipeline

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssafty.com/s10-ai-image-sub2/510P22C202.git

Credentials ?

haesuin/\*\*\*\*\* (개인 access token)

+ Add

고급

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/develop,be

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add

Script Path ?

./ForMyBaby/backend/jenkinsfile

Lightweight checkout ?

Pipeline Syntax

저장

Apply

CI/CD 포팅메뉴얼

9

## Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k10c109.p.ssafy.io:8081/project/FE> ?

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

- ☒ Set build description to build cause (eg. Merge request or Git Push) ?

- ☐ Build on successful pipeline events

Pending build name for pipeline ?

- ☐ 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 ?

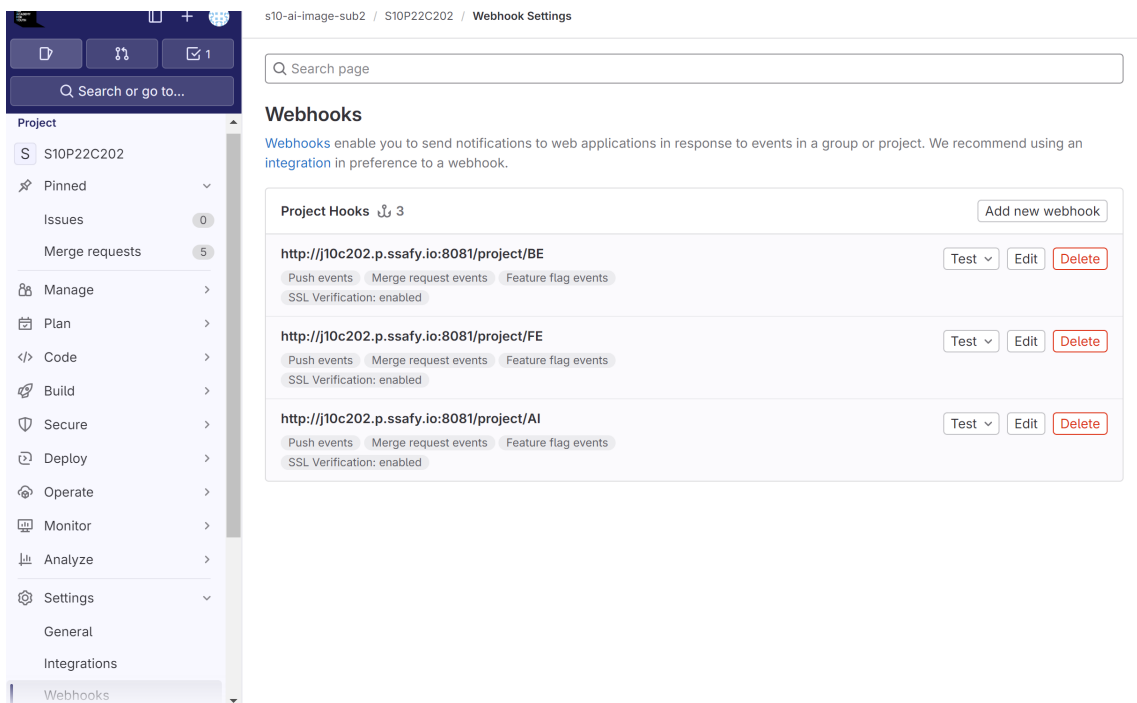
Generate

## ▼ jenkins-gitlab webhook 연결

### 1. jenkinsplugin 설치



## 2. 깃랩 웹훅 연결



## 3. 웹훅 설정

s10-ai-image-sub2 / S10P22C202 / Webhook Settings / Webhook

**Secret token**

.....

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

**Trigger**

☒ Push events

☐ All branches

☒ Wildcard pattern

develop\_be

Wildcards such as `*-stable` or `production/*` are supported.

☐ Regular expression

☐ Tag push events

A new tag is pushed to the repository.

☐ Comments

A comment is added to an issue or merge request.

☐ Confidential comments

A comment is added to a confidential issue.

☐ Issues events

An issue is created, updated, closed, or reopened.

☐ Confidential issues events

A confidential issue is created, updated, closed, or reopened.

☒ Merge request events

A merge request is created, updated, or merged.

☐ Job events

A job's status changes.

☐ Pipeline events

A pipeline's status changes.

☐ Wiki page events

A wiki page is created or updated.

☐ Deployment events

A deployment starts, finishes, fails, or is canceled.

☒ Feature flag events

A feature flag is turned on or off.

☐ Releases events

A release is created, updated, or deleted.

☐ Emoji events

An emoji is awarded or revoked. [Which emoji events trigger webhooks?](#)

**SSL verification**

☒ Enable SSL verification

[Save changes](#) [Test](#)

[Delete](#)

4.

## ▼ jenkins 환경 변수 설정하기

<https://g4daclom.tistory.com/78>

## ▼ 프로젝트 파일 세팅 확인

- 먼저 바탕화면 새폴더에 해야 할 프로젝트를 클론 시키기 전에 해야 할 git 명령어
- git init부터 초기화 하고 시작해야함 무조건!
- 방식1
  - git clone을 통해서 main 브랜치에 있는 거 다 받아오기
- 방식2
  - git remote add origin을 통해서 원격 저장소에 연결
  - 연결 후 pull 받고 싶은 브랜치만 로컬 저장소에 등록?해서
  - 그 브랜치 내용만 pull 받기

아래는 방식2를 적용하였다.

```
SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger
$ git init
Initialized empty Git repository in C:/Users/SSAFY/Desktop/trigger/.git/

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git remote add origin https://lab.ssafy.com/s10-final/S10P31C109.git

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
```

```

$ git remote -v
origin https://lab.ssafy.com/s10-final/S10P31C109.git (fetch)
origin https://lab.ssafy.com/s10-final/S10P31C109.git (push)

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git checkout -t origin/develop_be
fatal: 'origin/develop_be' is not a commit and a branch 'develop_be' cannot be created

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git branch -r

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git fetch --all
remote: Enumerating objects: 3113, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (112/112), done.
Receiving objects: 88% (2750/3113), 521.54 MiB | 5.59 MiB/s

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git fetch --all
remote: Enumerating objects: 3113, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (112/112), done.
remote: Total 3113 (delta 6), reused 0 (delta 0), pack-reused 2983
Receiving objects: 100% (3113/3113), 918.66 MiB | 5.24 MiB/s, done.
Resolving deltas: 100% (113/113), done.
From https://lab.ssafy.com/s10-final/S10P31C109
 * [new branch]      develop      -> origin/develop
 * [new branch]      develop_be   -> origin/develop_be
 * [new branch]      develop_fe   -> origin/develop_fe
 * [new branch]      develop_game -> origin/develop_game
 * [new branch]      master       -> origin/master

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (master)
$ git checkout -t origin/develop_be
Switched to a new branch 'develop_be'
branch 'develop_be' set up to track 'origin/develop_be'.

SSAFY@DESKTOP-LL832F4 MINGW64 ~/Desktop/trigger (develop_be)
$ git pull

```

위에서 보여주신 과정은 Git을 사용하여 원격 저장소에서 로컬 컴퓨터로 코드를 복제하고 관리하는 일련의 단계입니다.

#### 1. git init

이 명령어는 현재 폴더에 새로운 Git 저장소를 초기화합니다. 즉, 현재 폴더를 Git이 추적할 수 있는 저장소로 만듭니다.

#### 2. git remote add origin <URL>

원격 저장소의 주소를 로컬 저장소에 연결합니다. 여기서 'origin'은 원격 저장소의 단축 이름입니다. 이 단축

#### 3. git remote -v

현재 설정된 모든 원격 저장소의 리스트와 그들의 URL을 보여줍니다. -v 옵션은 fetch와 push 주소를 모두 표시합니다.

#### 4. git checkout -t origin/develop\_be

원격 저장소의 특정 브랜치(develop\_be)를 기반으로 새로운 로컬 브랜치를 생성하고, 그 브랜치로 전환하려고

5. `git branch -r`

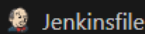
원격 저장소에 존재하는 모든 브랜치의 목록을 보여줍니다. 이 명령어로 사용 가능한 브랜치를 확인할 수 있습니다.

6. `git fetch --all`

모든 원격 저장소의 데이터를 로컬로 가져옵니다. 이 과정에서 원격 저장소의 모든 변경사항과 브랜치 정보가 로컬로 가져옵니다.

이 명령어들은 원격 저장소로부터 코드를 로컬 시스템으로 가져오고, 특정 브랜치로 작업을 시작하기 위한 환경을 만듭니다.

## ▼ 프로젝트 jenkins files 설정



Jenkinsfile 세팅 (소문자 j (X))

### ▼ 먼저 Jenkins files 파이프라인 흐름 이해하고 들어가기

파이프라인을 구성하는 각 단계는 코드의 준비부터 테스트, 빌드, 배포에 이르기까지 전체적인 워크플로우를 나타냅니다.

**Checkout:**

**목적:** 코드 저장소(SCM)에서 최신 코드를 가져오는 것입니다.

**이유:** 모든 작업을 시작하기 전에 최신 코드를 기반으로 작업하는 것이 중요하기 때문입니다. 이를 통해 최근 코드를 기반으로 작업을 할 수 있습니다.

**Build:**

**목적:** 소스 코드를 사용하여 실행 가능한 파일 또는 도커 이미지를 생성하는 단계입니다.

**이유:** 개발된 코드가 실제 실행 가능한 형태로 만들어지는 중요한 과정입니다. 도커 이미지로 빌드하는 이유는 코드를 실행 가능한 형태로 만들기 위함입니다.

**Login:**

**목적:** Docker Hub와 같은 컨테이너 레지스트리에 로그인하여 이미지를 푸시할 수 있는 권한을 확보하는 단계입니다.

**이유:** 보안된 방식으로 인증을 거쳐야 이미지를 원격 저장소에 안전하게 업로드할 수 있습니다.

**Tag and Push:**

**목적:** 빌드된 도커 이미지에 태그를 붙이고 Docker Hub로 푸시하는 작업을 수행합니다.

**이유:** 이 단계를 통해 생성된 이미지를 다른 개발자나 배포 시스템에서 접근할 수 있도록 공유합니다.

**Prune old images:**

**목적:** 시스템에 저장된 오래된 도커 이미지를 정리합니다.

**이유:** 디스크 공간을 확보하고, 시스템의 성능을 유지하기 위해 불필요한 이미지를 정기적으로 정리합니다.

**Pull:**

**목적:** 최신 도커 이미지를 Docker Hub로부터 다운로드합니다.

**이유:** 다른 환경에서도 동일한 이미지를 사용하여 일관된 결과를 보장하기 위함입니다.

**Up:**

**목적:** 도커 컨테이너를 실행시키는 과정입니다.

**이유:** 실제로 서비스를 운영하거나 테스트하기 위해 컨테이너를 실행합니다. 문제 발생 시 자동으로 재시작하는 기능을 추가할 수 있습니다. 이러한 단계를 따르는 것은 전체적인 개발 및 배포 프로세스를 자동화 하고, 에러를 최소화하며, 개발 속도

### ▼ 1. front

```
pipeline {
    agent any // 이 파이프라인은 어떤 Jenkins 에이전트에서도 실행될 수 있음을 의미합니다.

    environment {
        REPO = "s10-final/S10P31C109" // 환경 변수를 설정하여 파이프라인 전체에서 사용합니다.
    }

    stages {
        stage('Checkout') {
```

```

        steps {
            checkout scm // 소스 코드 관리(SCM) 시스템에서 최신 코드를 체크아웃합니다.
        }
    }

//
// stage('Setup Environment') {
//     steps {
//         dir("${env.WORKSPACE}/Easysign_fe-edu"){
//             script {
//                 sh "ls -al"
//                 sh "ls secure-settings -al"
//                 sh "chmod +x ./gradlew"
//                 sh "cp ./secure-settings/application.yml ./src/main/reso
//                 sh "cp ./secure-settings/application-dev.yml ./src/main/
//                 sh "ls ./src/main/resources -al"
//             }
//         }
//     }
// }
// stage('JUnit Test') {
//     steps {
//         sh "./gradlew test"
//     }
// }

stage("Build") {
    steps {
        script {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede
                sh "docker build -t ${DOCKER_USER_ID}/front front" // Dock
                sh "docker system prune --filter until=10h" // 오래된 Docker
            ]])
        }
    }
}

stage("Login") {
    steps {
        withCredentials([[ $class: 'UsernamePasswordMultiBinding', credentia
            sh """
                set +x
                echo $DOCKER_USER_PASSWORD | docker login -u $DOCKER_USER_I
                set -x
            """
        ]])
    }
}

stage("Tag and Push") {
    steps {
        script {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede

```

```

        sh "docker push ${DOCKER_USER_ID}/front" // 이미지를 Docker
    }
}
}

stage('Prune old images'){
    steps{
        script{
            sh "docker system prune --filter until=1h" // 한 시간 이내의 오래된
        }
    }
}

stage('Pull') {
    steps {
        script {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede
            sh "docker pull ${DOCKER_USER_ID}/front" // 최신 이미지를 Doc
        }
    }
}

stage('Up') {
    steps {
        script {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede
            try{
                sh "docker stop -f front || true" // 실행 중인 컨테이너를
                sh "docker rm -f front || true" // 컨테이너를 제거합니다.
                sh "docker run -d --name front -p 3001:3000 ${DOCKER_US
            } catch (Exception e){
                sh "docker restart front" // 실패할 경우 컨테이너를 재시작합
            }
        }
    }
}

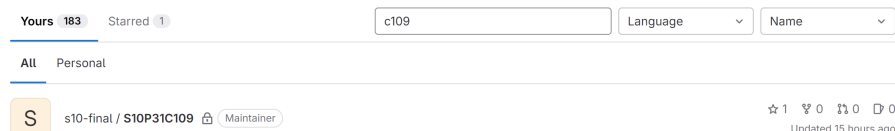
// post {
//     success {

```



```
//      script {
//          mattermostSend (color: 'good',
//              message: "FE 배포 성공      :cat_jump: :loopy_happy: :bboong: ",
//              )
//      }
//  }
//  failure {
//      script {
//          mattermostSend (color: 'danger',
//              message: "FE 배포 실패      :cry_tom: :cryingloopy: :cryingpatamon: ",
//              )
//      }
//  }
//  }
//  post {
//      always {
//          script {
//              def Author_ID = sh(script: "git show -s --pretty=%an", returnStd
//              def Author_Name = sh(script: "git show -s --pretty=%ae", returns
//              mattermostSend (color: 'good',
//                  message: "빌드 ${currentBuild.currentResult}: ${env.JOB_N
//                  endpoint: 'https://meeting.ssafy.com/hooks/q4qjarpscbf9p
//                  channel: 'C107-Jenkins'
//              )
//          }
//      }
//  }
//  }
```

## ▼ 2. back



- REPO 설정에 사용되는

```
pipeline {
    agent any
    environment {
        REPO = "s10-ai-image-sub2/S10P22C202"
        DB_URL = "${env.DB_URL}"
        DB_USER = "${env.DB_USER}"
        DB_PASSWORD = "${env.DB_PASSWORD}"
        REDIS_HOST = "${env.REDIS_HOST}"
        REDIS_PASSWORD = "${env.REDIS_PASSWORD}"
        REDIS_PORT = "${env.REDIS_PORT}"
        KAKAO_ID = "${env.KAKAO_ID}"
        KAKAO_REDIRECT_URI = "${env.KAKAO_REDIRECT_URI}"
        KAKAO_PW = "${env.KAKAO_PW}"
        S3_ACCESS_KEY = "${env.S3_ACCESS_KEY}"
        S3_SECRET_KEY = "${env.S3_SECRET_KEY}"
        S3_REGION = "${env.S3_REGION}"
    }
}
```

```

    S3_BUCKET = "${env.S3_BUCKET}"
}
stages {
    stage('Checkout') {
        steps {
            checkout scm
        }
    }
    stage('Setup Environment') {
        steps {
            dir("${env.WORKSPACE}/ForMyBaby/backend"){
                script {
                    sh "ls -al"
                    sh "echo 'SUBMODULE CHECK'"
                    sh "ls ./src/main/resources"
                    sh "chmod +x ./gradlew"
                    sh "cat ./src/main/resources/application.yml"
//                    sh "cp ./src/main/resources/application.yml"
//                    sh "chmod +x ./gradlew"
//                    sh "cat ./src/main/resources/application.yml"
                }
            }
        }
    }
    stage("Build") {
        steps {
            script {
                sh "ls -al"
                withCredentials([[ $class: 'UsernamePasswordMultiBinding', c
                    echo "도커허브 아이디: ${DOCKER_USER_ID}"
                    echo "도커허브 비밀번호: ${DOCKER_USER_PASSWORD}"
                    sh "docker build --no-cache -t ${DOCKER_USER_ID}/back F
            }
        }
    }
    stage("Login") {
        steps {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', credentia
                sh ""
                set +x
                echo $DOCKER_USER_PASSWORD | docker login -u $DOCKER_USER_I
                set -x
            ""
        }
    }
    stage("Tag and Push") {
        steps {
            script {
                withCredentials([[ $class: 'UsernamePasswordMultiBinding', cred
                    sh "docker push ${DOCKER_USER_ID}/back"
            }
        }
    }
}

```

```

    }
  }
}

stage('Prune old images'){
  steps{
    script{
      sh "docker ps"
      sh "docker system prune --filter until=10h"
    }
  }
}

stage('Pull') {
  steps {
    script {
      withCredentials([[$class: 'UsernamePasswordMultiBinding', crede
//      sh "docker stop back || true" // Ignore error ifgit co
//      sh "docker rm back || true" // Ignore error if contain
//      sh "docker rmi hyeiiin/back" //images 날리기
//      sh "docker pull ${DOCKER_USER_ID}/back"
    }
  }
}

stage('Up') {
  steps {
    script {
      withCredentials([[$class: 'UsernamePasswordMultiBinding', crede
      try {
        sh "docker stop -f back || true" // 현재 컨테이너가 돌아가고
        sh "docker rm -f back || true" // 그리고 나서 현재 컨테이너를
        sh "docker rmi back || true" // 그리고 이미지도 지운다. 안 지
        sh "docker run -d --name back -p 8082:8080 -e DB_URL=${
      } catch (Exception e) {
        sh "docker restart back || true" // Ignore error if co
      }
    }
  }
}

post {
  success {
    script {
      mattermostSend (color: 'good',
      message: "BE 배포 성공 :cat_jump: :loopy_happy: :bboong: ",
      )
    }
  }
  failure {
    script {
      mattermostSend (color: 'danger',

```

```

        message: "BE 배포 실패      :cry_tom: :cryingloopy: :cryingpatamon: "
    )
}
}
}
}
}
}

```

### ▼ 3. ai

```

pipeline {
    agent any
    environment {
        REPO = "s10-ai-image-sub2/S10P22C202"
    }
    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage("Build") {
            steps {
                script {
                    sh "pwd"
                    sh "ls -al"
                    withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede
                        sh "docker build -t ${DOCKER_USER_ID}/ai_node ForMyBaby/ai/
//                        sh "docker build -t ${DOCKER_USER_ID}/ai_py ForMyBaby/ai

                }
            }
        }

        stage("Login") {
            steps {
                withCredentials([[ $class: 'UsernamePasswordMultiBinding', credentia
                    sh """
                        set +x
                        echo $DOCKER_USER_PASSWORD | docker login -u $DOCKER_USER_I
                        set -x
                    """
            }
        }

        // stage("Tag and Push") {
        //     steps {
        //         script {
        //             withCredentials([[ $class: 'UsernamePasswordMultiBinding', c
        //                 sh "docker push ${DOCKER_USER_ID}/ai_node"
        //                 sh "docker push ${DOCKER_USER_ID}/ai_py"
        //             }
        //         }
        //     }
        // }
        // stage('Prune old images'){

```

```

//      steps{
//          script{
//          }
//      }
//  }
//  stage('Pull') {
//      steps {
//          script {
//              withCredentials([[ $class: 'UsernamePasswordMultiBinding', cr
//                  sh "docker pull ${DOCKER_USER_ID}/ai_node"
//                  sh "docker pull ${DOCKER_USER_ID}/ai_py"
//              ]])
//          }
//      }
//  }
stage('Up') {
    steps {
        script {
            withCredentials([[ $class: 'UsernamePasswordMultiBinding', crede
            try{
                sh "docker stop -f ai_node || true"
                sh "docker rm -f ai_node || true"
                sh "docker rmi ai_node || true"
                sh "docker run -d --name ai_node -p 8083:8083 ${DOCKER_
//            sh "docker stop -f ai_py || true"
//            sh "docker rm -f ai_py || true"
//            sh "docker rmi ai_py || true"
//            sh "docker run -d --name ai_py -p 8084:8083 --device
//            sh "docker run -d --name ai_py -p 8084:8083 ${DOCKER
            } catch (Exception e){
                sh "docker restart ai_node"
//            sh "docker restart ai_py"
            }
        }
    }
}

post {
    success {
        script {
            mattermostSend (color: 'good',
            message: "AI 배포 성공      :cat_jump: :loopy_happy: :bboong: ",
            )
        }
    }
    failure {
        script {
            mattermostSend (color: 'danger',
            message: "AI 배포 실패      :cry_tom: :cryingloopy: :cryingpatamon: "
            )
        }
    }
}

```

## ▼ 가비아 도메인 설정

**gabia.** 도메인 관리

김도현님 [로그아웃](#)

전체 도메인 02개

My 가비아 | 1:1 문의

홈 > DNS 정보

전체 도메인

도메인 정보 변경

**DNS 정보**

도메인 보안

예약 도메인 관리

관심 도메인

상표 보호

고객센터

내임서버

1544-4370

도메인 고객을 위한 특별한 혜택  
마이크로사이트  
무료 제공!

DNS 정보

DNS 정보는 도메인에 연결된 서비스를 확인할 수 있습니다.  
DNS 설정, 도메인 연결, 포워딩, 파킹 서비스 등의 신청 및 설정은 'DNS 관리'에서 가능합니다.

DNS 관리

번호	도메인명	DNS 정보	연결 서비스	만기일
2	easysign.shop	CNAME	DNS 설정	2025-02-06
1	trigger109.com	설정된 DNS 레코드 정보가 없습니다.		2025-04-30

엑셀 다운로드

- DNS 관리 Tool 클릭

**gabia.** DNS 관리

김도현님 [로그아웃](#)

전체 도메인 1개 (가비아 등록 도메인 + 타기관 등록 도메인)

[한국어](#)

[English](#)

My 가비아 | 1:1 문의

홈 > DNS 설정

가비아 등록 도메인

DNS 관리

DNS 권한 설정

타기관 등록 도메인

DNS 설정

easysign.shop

레코드 개수 : 3개

최근 업데이트 : 2024-02-12 21:31:44

이력 확인

엑셀 다운로드

타입	호스트	값/위치	TTL	우선 순위	서비스	상태
A	@	3.36.55.182	1800		DNS 설정	수정 삭제
CNAME	edu	easysign.shop.	600		DNS 설정	수정 삭제
CNAME	jenkins	easysign.shop.	600		DNS 설정	수정 삭제

레코드 추가

DNS 설정 목록

저장

**DNS 설정**

**레코드 수정**

타입	호스트	
A	@	IPv4
CNAME	도메인 앞 추가부분	구매한 도메인.

IPv4 주소 보는 명령어

```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```

- A 타입 : @ 모든 url 허용 ex) edu.easysign.shop, 값/위치는 서버 주소를 사용해야한다.
- CNAME : 도메인에 앞에 붙는 것 ex) edu.easysign.shop, 값 위치는 우리가 구매한 도메인 주소에.까지 붙인다.

- TTL은 "Time to Live"의 약자로, DNS(Domain Name System) 레코드가 캐시에서 저장될 수 있는 시간을 나타냅니다
- 도메인 설정해야한다.



## SSL 설정

### 3. Certbot Nginx 연결

- **sudo certbot --nginx**
  - 이메일 입력
  - 약관 동의 - **Y**
  - 이메일 수신동의
  - 도메인 입력 - **i10{팀코드}.p.ssafy.io**
  - http 입력시 리다이렉트 여부 - **2**

ex)

```
sudo certbot --nginx -d notice.trigger109.com
```

## ▼ Nginx 설정

- version : nginx/1.18.0

### 1. 패키지 업데이트 및 업그레이드

- `sudo apt update`
- `sudo apt upgrade` or `sudo add-apt-repository --remove ppa:certbot/certbot`
- `free -h` (현재 메모리 용량 확인)

### 2. 방화벽 설정

- `sudo ufw status` (방화벽 허용)
- `sudo ufw allow [포트번호]` (방화벽 허용할 포트번호 입력)

### 3. nginx 설치

- `sudo apt install nginx -y`
- `sudo systemctl status nginx` (설치 후 상태 확인)

### 4. SSL 설치

SSL 설치

## 5. Certbot 설치

```
- sudo apt-get install certbot python3-certbot-nginx  
  
- sudo certbot -nginx (certbot nginx 연결)
```

```
[  
  
1. 이메일 입력  
2. 약관 동의 : Y  
3. 이메일 수신 동의  
4. 도메인 입력 : i10c204.p.ssafy.io  
5. http 입력시 Redirect  
  
]
```

## 6. Nginx 환경 설정

```
- sudo cd /etc/nginx/sites-available/{파일명}.conf  
sudo vi /etc/nginx/sites-available/default
```

## 7. Nginx 설정 파일

```
server {  
    server_name j10c202.p.ssafy.io;  
  
    proxy_set_header X-Real-IP $remote_addr;  
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
    proxy_set_header Host $http_host;  
    proxy_set_header X-Forwarded-Proto $scheme;  
  
    location /ai {  
        proxy_pass http://localhost:8083;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection "upgrade";  
        proxy_set_header Host $host;  
        proxy_set_header Origin "";  
    }  
  
    location /v1/ {  
        proxy_pass http://localhost:8082;  
    }  
  
    location /ws {  
        proxy_pass http://localhost:3001;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection "upgrade";  
        proxy_set_header Host $host;  
        proxy_set_header Origin "";  
    }  
  
    location / {
```



```

        proxy_pass http://localhost:3001;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header Origin "";
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/j10c202.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/j10c202.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = j10c202.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name j10c202.p.ssafy.io;
    return 404; # managed by Certbot
}

```

```

server {
    // [!1.도메인 주소 변경하기]
    server_name k10c109.p.ssafy.io;

    location / {
        return 301 https://trigger109.com$request_uri;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/k10c109.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/k10c109.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = i10c202.p.ssafy.io ) {
        return 301 https://easysign.shop$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name i10c202.p.ssafy.io;
    return 404; # managed by Certbot
}

```

```

server {
    server_name easysign.shop;

    location / {
        proxy_pass http://localhost:8083;
    }

    location /api {
        proxy_pass http://localhost:8082;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/easysign.shop/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/easysign.shop/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = easysign.shop ) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name easysign.shop;
    return 404; # managed by Certbot
}

# edu.easysign.shop설정

server {
    server_name edu.easysign.shop;

    location / {
        proxy_pass http://localhost:8084;
    }

    location /api {
        proxy_pass http://localhost:8082;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/edu.easysign.shop/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/edu.easysign.shop/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = edu.easysign.shop ) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
}

```

```

listen 80;
listen [::]:80 ;
server_name edu.easysign.shop;
return 404; # managed by Certbot
}

server {
    server_name jenkins.easysign.shop;

    location / {
        proxy_pass http://localhost:8081;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/jenkins.easysign.shop/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/jenkins.easysign.shop/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = jenkins.easysign.shop) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    server_name jenkins.easysign.shop;
    listen 80;
    return 404; # managed by Certbot
}

```

```

# i10c202.p.ssafy.io 도메인에 대한 HTTPS 설정
server {
    server_name i10c202.p.ssafy.io;

    # 모든 HTTPS 요청을 easysign.shop로 리다이렉트
    location / {
        return 301 https://easysign.shop$request_uri;
    }

    # IPv6 주소에서 SSL/TLS를 사용하여 443 포트를 리스닝
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    # SSL/TLS를 사용하여 443 포트를 리스닝
    listen 443 ssl; # managed by Certbot
    # SSL 인증서 경로
    ssl_certificate /etc/letsencrypt/live/i10c202.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i10c202.p.ssafy.io/privkey.pem; # managed by Certbot
    # SSL 설정 포함
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

# i10c202.p.ssafy.io 도메인에 대한 HTTP 설정
server {

```

```

# HTTP 요청을 HTTPS로 리다이렉트
if ( $host = i10c202.p.ssafy.io ) {
    return 301 https://easysign.shop$request_uri;
} # managed by Certbot

listen 80;
listen [::]:80;
server_name i10c202.p.ssafy.io;
# 설정된 리다이렉션 외 요청은 404 처리
return 404; # managed by Certbot
}

# easysign.shop 도메인의 HTTPS 설정
server {
    server_name easysign.shop;

    # 프록시 설정
    location / {
        proxy_pass http://localhost:8083; # 웹 애플리케이션 서버로 요청 전달
    }
    location /api {
        proxy_pass http://localhost:8082; # API 서버로 요청 전달
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/easysign.shop/fullchain.pem; # managed by
    ssl_certificate_key /etc/letsencrypt/live/easysign.shop/privkey.pem; # managed
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

# easysign.shop 도메인의 HTTP 설정
server {
    # HTTP 요청을 HTTPS로 리다이렉트
    if ( $host = easysign.shop ) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name easysign.shop;
    # 설정된 리다이렉션 외 요청은 404 처리
    return 404; # managed by Certbot
}

# edu.easysign.shop 도메인의 HTTPS 설정
server {
    server_name edu.easysign.shop;

    # 프록시 설정
    location / {
        proxy_pass http://localhost:8084; # 교육용 애플리케이션 서버로 요청 전달
    }
    location /api {
        proxy_pass http://localhost:8082; # API 서버로 요청 전달
    }

```

```

    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/edu.easysign.shop/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/edu.easysign.shop/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

# edu.easysign.shop 도메인의 HTTP 설정
server {
    # HTTP 요청을 HTTPS로 리다이렉트
    if ($host = edu.easysign.shop ) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name edu.easysign.shop;
    # 설정된 리다이렉션 외 요청은 404 처리
    return 404; # managed by Certbot
}

# jenkins.easysign.shop 도메인의 HTTPS 설정
server {
    server_name jenkins.easysign.shop;

    # Jenkins 서비스로 프록시 설정
    location / {
        proxy_pass http://localhost:8081;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/jenkins.easysign.shop/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/jenkins.easysign.shop/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

# jenkins.easysign.shop 도메인의 HTTP 설정
server {
    # HTTP 요청을 HTTPS로 리다이렉트
    if ($host = jenkins.easysign.shop) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name jenkins.easysign.shop;
    # 설정된 리다이렉션 외 요청은 404 처리
    return 404; # managed by Certbot
}

```

```

server {
    server_name k10c109.p.ssafy.io; # 서버의 호스트 이름을 설정합니다.
}

```

```

    location / {
        # 301은 영구적인 리다이렉트를 의미합니다. 여기서는 모든 요청을 trigger109.com으로 리다이렉트합니다.
        return 301 https://trigger109.com$request_uri;
    }

    location /api {
        proxy_pass http://localhost:8082; # /api로 들어오는 요청을 localhost의 포트 8082로 프록시합니다.
    }

    listen 443 ssl; # 443번 포트에서 HTTPS 연결을 수신합니다.
    ssl_certificate /etc/letsencrypt/live/k10c109.p.ssafy.io/fullchain.pem; # SSL 인증서
    ssl_certificate_key /etc/letsencrypt/live/k10c109.p.ssafy.io/privkey.pem; # SSL 키
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 옵션 설정 파일 포함
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # SSL DH 파라미터 설정
}

server {
    if ($host = k10c109.p.ssafy.io) {
        return 301 https://$host$request_uri; # HTTP 요청을 HTTPS로 리다이렉트합니다.
    }

    listen 80; # 80번 포트에서 HTTP 연결을 수신합니다.
    server_name k10c109.p.ssafy.io;
    return 404; # 해당 서버의 404 오류를 처리합니다.
}

server {
    server_name main.trigger109.com; # 서버의 호스트 이름을 설정합니다.

    location / {
        proxy_pass http://localhost:3001; # 모든 요청을 localhost의 포트 3001로 프록시합니다.
    }

    listen 443 ssl; # 443번 포트에서 HTTPS 연결을 수신합니다.
    ssl_certificate /etc/letsencrypt/live/trigger109.com/fullchain.pem; # SSL 인증서
    ssl_certificate_key /etc/letsencrypt/live/trigger109.com/privkey.pem; # SSL 키
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 옵션 설정 파일 포함
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # SSL DH 파라미터 설정
}

server {
    if ($host = main.trigger109.com) {
        return 301 https://$host$request_uri; # HTTP 요청을 HTTPS로 리다이렉트합니다.
    }

    listen 80; # 80번 포트에서 HTTP 연결을 수신합니다.
    server_name main.trigger109.com;
    return 404; # 해당 서버의 404 오류를 처리합니다.
}

server {
    server_name jenkins.trigger109.com; # 서버의 호스트 이름을 설정합니다.

    location / {

```

```

        proxy_pass http://localhost:8081; # 모든 요청을 localhost의 포트 8081로 프록시함
    }

    listen 443 ssl; # 443번 포트에서 HTTPS 연결을 수신합니다.
    ssl_certificate /etc/letsencrypt/live/jenkins.trigger109.com/fullchain.pem; #
    ssl_certificate_key /etc/letsencrypt/live/jenkins.trigger109.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 옵션 설정 파일 포함
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # SSL DH 파라미터 설정
}

server {
    if ($host = jenkins.trigger109.com) {
        return 301 https://$host$request_uri; # HTTP 요청을 HTTPS로 리다이렉트합니다.
    }

    server_name jenkins.trigger109.com;
    listen 80; # 80번 포트에서 HTTP 연결을 수신합니다.
    return 404; # 해당 서버의 404 오류를 처리합니다.
}

```

## 방법 1) /etc/docker/daemon.json 추가

가장 많이 알려진 방법

```

$ sudo vi /etc/docker/daemon.json
### 해당 파일에 아래 항목 추가
{
    "iptables" : false
}
###
### 도커 재시작
$ systemctl restart docker

```

이 Nginx 서버 구성 파일에는 여러 `server` 블록들이 포함되어 있으며, 각각의 블록은 도메인 이름, 리다이렉션 정책, SSL 인증서, 프록시 설정 등 다양한 웹 서버 설정을 정의합니다. 각 부분을 차례대로 살펴보겠습니다.

### 1. `i10c202.p.ssafy.io` 도메인에 대한 설정

#### HTTPS 설정

- 도메인 이름: `i10c202.p.ssafy.io`
- 리다이렉션: HTTPS를 사용해 이 도메인으로 들어오는 모든 요청을 `https://easysign.shop` 으로 보냅니다. 즉, 사이트 방문자가 이 도메인을 사용해 접속하면 자동으로 `easysign.shop` 으로 이동하게 됩니다.
- 보안: SSL/TLS 인증서를 사용하여 데이터를 암호화하고, 이를 통해 보안 연결을 유지합니다.

#### HTTP 설정

- 도메인 이름: `i10c202.p.ssafy.io`
- 리다이렉션: HTTP로 접속한 사용자도 자동으로 HTTPS 버전의 `easysign.shop` 으로 리다이렉트됩니다.
- 에러 처리: 설정된 리다이렉션이 아닌 다른 모든 HTTP 요청은 404 에러(찾을 수 없는 페이지)로 응답합니다.

## 2. easysign.shop 도메인에 대한 설정

### HTTPS 설정

- **프록시:** 웹 서버는 사용자의 요청을 내부적으로 다른 서버나 포트로 전달합니다. 예를 들어, `/api` 경로로의 요청은 8082 포트로 전달됩니다. 이는 보통 백엔드 서비스나 API 서버로 요청을 넘기기 위해 사용됩니다.
- **보안:** SSL 인증서를 사용해 데이터를 암호화합니다.

### HTTP 설정

- **리다이렉션:** HTTP로 접속하는 사용자를 안전한 HTTPS 연결로 리다이렉트합니다.
- **에러 처리:** 설정된 리다이렉션 외의 요청은 404 에러로 처리됩니다.

## 3. edu.easysign.shop 도메인에 대한 설정

### HTTPS 설정

- **프록시:** `/` 경로의 요청은 8084 포트로, `/api` 경로의 요청은 8082 포트로 프록시됩니다. 이렇게 설정함으로써 웹 애플리케이션의 다른 부분을 다른 서비스가 처리하도록 할 수 있습니다.
- **보안:** SSL 인증서로 보안 연결을 유지합니다.

### HTTP 설정

- **리다이렉션:** HTTP로 접속하는 사용자를 HTTPS로 리다이렉트합니다.
- **에러 처리:** 설정된 리다이렉션 외의 요청은 404 에러로 처리됩니다.

## 4. jenkins.easysign.shop 도메인에 대한 설정

### HTTPS 설정

- **프록시:** 모든 요청을 로컬의 8081 포트로 전달합니다. 이는 일반적으로 Jenkins와 같은 CI/CD 도구가 작동하는 설정입니다.
- **보안:** SSL 인증서로 보안 연결을 유지합니다.

### HTTP 설정

- **리다이렉션:** HTTP로 접속하는 사용자를 HTTPS로 리다이렉트합니다.
- **에러 처리:** 설정된 리다이렉션 외의 요청은 404 에러로 처리됩니다.

이 설정 파일은 각 도메인에 대해 보안 HTTPS 연결을 사용하도록 설정하고 있으며, 필요한 경우 특정 요청을 내부 포트나 서비스로 전달하는 프록시 설정을 포함하고 있습니다. 이를 통해 외부에서 보이는 웹 서비스와 내부적으로 처리하는 서비스 간에 보안 및 효율적인 연결을 유지합니다.

## ▼ Redis

### ▼ Redis 정보

버전 : redis 7.2.4

### ▼ Redis 설치 및 설정

```
# Redis에 필요한 패키지를 먼저 설치한다.
sudo apt install lsb-release curl gpg

# Redis를 설치한다.
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings

echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://package

sudo apt-get update
```



```

sudo apt-get install redis

# Redis의 설정을 변경하기 위해 redis.conf의 권한을 수정한다.
sudo chown root:root /etc/redis/redis.conf

# Redis의 설정을 변경한다.
sudo vi /etc/redis/redis.conf
    # requirepass 비밀번호
    # bind 접근 가능한 ip
    # port 포트번호

# 다시 권한을 원래대로 돌린다.
sudo chown redis:redis /etc/redis/redis.conf

# 바뀐 설정을 적용시킨다.
sudo systemctl restart redis-server.service

# 재부팅시에도 자동으로 실행되도록 한다.
sudo systemctl enable redis-server.service

# redis cli로 확인하기
redis-cli

127.0.0.1:6379> 이 상태로 넘어오면
auth {requirepass(설정한 비밀번호를 입력)}
127.0.0.1:6379> auth mungmung109!
OK
다음처럼 관리자으로 오고
127.0.0.1:6379> set test testValue
다음처럼 키 벨류 값 으로 저장
127.0.0.1:6379> get test
키 값을 통해서 벨류 값 조회

```

## ▼ EC2 Docker로 Mysql 설치하기

- <https://lucas-owner.tistory.com/47>

### 1. 가장 최근 MySQL 버전 다운로드

```
sudo docker pull mysql
```

### 2. Mysql container 생성 및 실행

```
sudo docker run --name mysql -e MYSQL_ROOT_PASSWORD=trigger109! -d -p 10912:3306 my
```

### 3. EC2 MySQL Docker 컨테이너 접속

#### a. 차례대로 커맨드 실행

```
sudo docker exec -it mysql-container bash
```

#### b. mysql -uroot -p (root 계정 로그인) 입력 후 설정한 패스워드 입력.

```
mysql -uroot -p
```

#### 4. DB 접속 정보

##### ▼ 환경 변수

DB\_URL : jdbc:mysql://i10c202.p.ssafy.io:3307/easysign

DB\_USER : ????

DB\_PASSWORD : ????