



포팅 매뉴얼

▼ Contents

1. Version

[Front-end](#)

[Back-end](#)

2. 외부 API

[Front-end](#)

[Back-end](#)

3. Deploy

[도커 설치](#)

[compose 디렉토리 만들고 docker-compose.yml 작성](#)

[백그라운드로 실행](#)

[젠킨스 구성](#)

[젠킨스 비밀번호 확인](#)

[젠킨스 접속](#)

[플러그인 설치](#)

[환경 설정](#)

[크리덴셜 설정 \(접근 자격 설정\)](#)

[파이프 라인 설정](#)

[backend pipeline](#)

[Web hook 설정](#)

[Nginx](#)

1. Version

Front-end

- Node.js: 18.14.2
- Npm: 9.6.0
- React: ^18.2.0
- Typescript: ^4.9.5
- react-router-dom: ^6.9.0
- recoil: ^0.7.7
- tailwind-css: ^3.2.7

Back-end

- Visual Studio Code: 1.74.2
- Python: 3.9.13
- Django: 3.2.13
- MariaDB: 10.3.23
- Redis: 5.0.7
- Docker: 23.0.1
- Jenkins: 2.387.1
- nginx: 1.18.0

2. 외부 API

Front-end

- Clarifai
- Kakao 로그인

Back-end

- Kakao 로그인
- OpenAI
- AWS S3
- AWS CloudFront

- Microsoft Azure
- Papago

▼ Front-end 설정 파일

▼ Back-end 설정 파일

```
# Django Project
SECRET_KEY={settings.py에 선언된 SECRET_KEY}

# AWS S3
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_REGION=
AWS_STORAGE_BUCKET_NAME=

# CloudFront
CLOUD_FRONT_DOMAIN={CloudFront 배포 도메인 이름}

# OpenAI
CHAT_GPT_API_KEY=

# Papago
PAPAGO_CLIENT_ID=
PAPAGO_CLIENT_SECRET=
PAPAGO_URL=

# 이메일 인증에서 사용할 발신 계정
EMAIL_HOST_USER=
EMAIL_HOST_PASSWORD=

# Redis
CLIENT_ID=
REDIS_KEY=

# Kakao 로그인
KAKAO_REST_API_KEY=
SOCIAL_LOGIN_PASSWORD={임의의 비밀번호 직접 설정}

# Azure
SPEECH_KEY=
SPEECH_REGION=
```

3. Deploy

도커 설치

```
sudo apt-get update
```

```
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg
```

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
echo \
"deb [arch=$(dpkg --print-architecture)] signed-by=/etc/apt/keyrings/docker.gpg https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
```

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

잘 설치되었는지 확인

```
sudo docker run hello-world
```

compose 디렉토리 만들고 docker-compose.yml 작성

```
mkdir compose
cd compose
sudo apt install docker-compose

vi docker-compose.yml
```

```
version: "3"
services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: ubuntu_jenkins
    user: root
    volumes:
      - /var/jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 7777:8080
```

젠킨스 포트를 7777로 연결한다.

백그라운드로 실행

```
docker-compose up -d
```

젠킨스 구성

젠킨스 비밀번호 확인

```
sudo su
cat /var/jenkins_home/secrets/initialAdminPassword
```

젠킨스 접속

http://{도메인 주소 혹은 아이피 주소}:7777/

비밀번호 입력후 회원가입

플러그인 설치

메뉴 Jenkins관리 → 플러그인 관리 → Available plugins 에서 검색해서 설치

Docker Pipeline

NodeJS Plugin

GitLab

Generic Webhook Trigger Plugin

SSH Agent Plugin

환경 설정

Dashboard > Jenkins 관리 > Global Tool Configuration

NodeJS installations ^ Edited

NodeJS installations

List of NodeJS installations on this system

Add NodeJS

NodeJS Name

██████████

☒ Install automatically ?

Install from nodejs.org

Version

NodeJS 18.14.2

For the underlying architecture, if available, force the installation of 32bit architecture

☐ Force 32bit architecture

Global npm packages to install

Name : 임의로 설정







Version : 사용하는 NodeJs 버전 선택하여 저장

크리덴셜 설정 (접근 자격 설정)

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

Global credentials (unrestricted) + Add Credentials

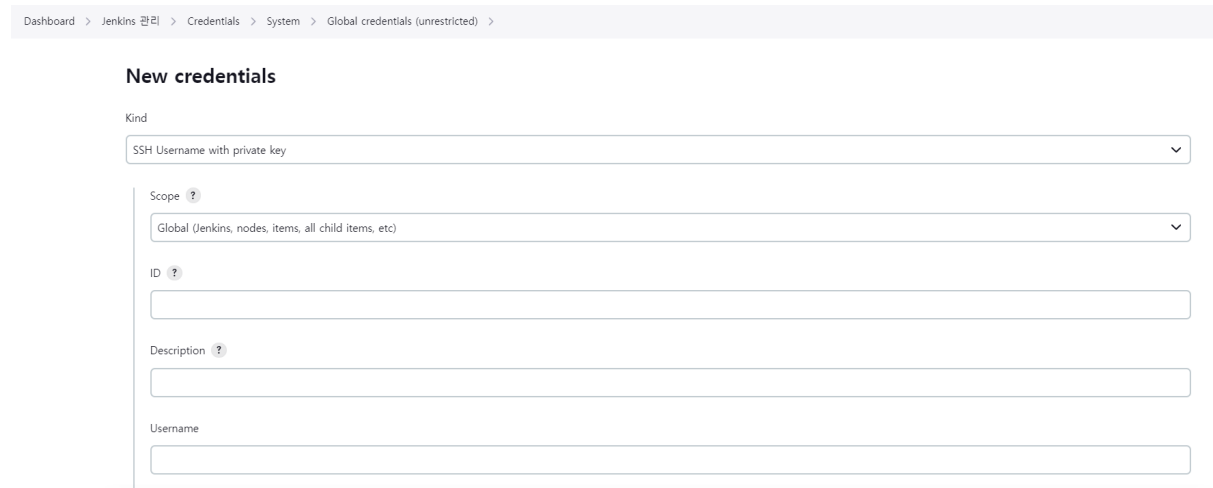
Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 ec2_ssh_key	ubuntu (ESK)	SSH Username with private key	ESK 
 gitlab_token	johjaewan/***** (GLT)	Username with password	GLT 
 dockerhub_key	johjaewan/***** (DHK)	Username with password	DHK 

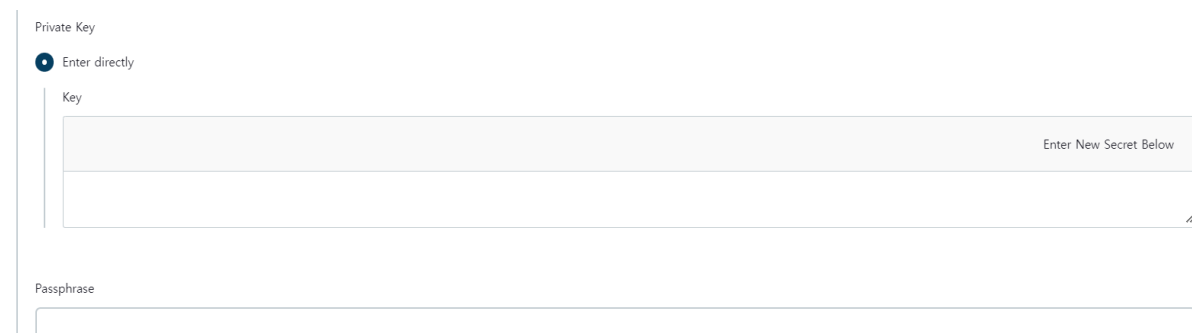
메뉴 Jenkins관리 → Manage Credentials → system → **Global credentials**

여기서 우측 상단에 Add Credentials 를 눌러서 추가한다.

-ec2_ssh_key : 서버에 접근하기 위한 권한



kind 에서 SSH Username with private key 선택후 ID(임의로 설정), Username(키를 받은 우분투 유저 이름 , 보통 Ubuntu), Description (설명)을 입력한 후



받은 서버 키를 입력해준다. ----BEGIN RSA PRIVATE KEY-----랑 ----END RSA PRIVATE KEY-----까지 포함해서 다 복사해서 넣어 준다.

-gitlab_token : 깃랩에 접근하기 위한 권한

kind 에서 Username with password 선택후 ID(임의로 설정), Username(깃랩 아이디), Description (설명)을 입력한 후 password 에 깃랩에서 받은 키를 넣는다. (만약 안된다면 깃랩 비밀번호를 넣는다.)

깃랩에서 키를 받기 위해서는 우측 상단에 자신의 계정을 누르고 Preferences → Access Tokens 에서 다 체크 후 생성하면 된다.

생성후 바로 복사를 하는거 추천 다시 못 보기 때문

(혹시 정상적으로 안된다면 kind 에 Gitlab API token 으로 한번 시도)

-dockerhub_key : 도커 허브에 접근하기 위한 권한 (이건 안씀 안 만들어도 됨)

파이프 라인 설정

새로운 item → pipeline 으로 생성 후

Pipeline 부분에

Definition을 pipeline script로 설정

backend pipeline

```
pipeline {
    agent any

    environment {
        imagename = "johjaewan/sample"
        registryCredential = 'dockerhub_key'
        dockerImage = ''
    }

    stages {

        stage('gitlab clone') {
            steps {
                git branch: 'BE', credentialsId: 'gitlab_token', url: 'https://lab.ssafy.com/s08-ai-speech-sub2/S08P22D103.git'
            }
        }

        stage('docker image'){
            steps {
                sh """
                if ! test docker; then
                curl -fsSL https://get.docker.com -o get-docker.sh
                sh get-docker.sh
                fi
                """

                sshagent(credentials: ['ec2_ssh_key']) {

                    sh '''
                    if test "`docker ps -aq --filter ancestor=backdb`; then

                    ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker stop $(docker ps -aq --filter ancestor=backdb)"
                    ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker rm -f $(docker ps -aq --filter ancestor=backdb)"
                    ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker rmi backdb"
                    fi
                    '''

                    sh "ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io 'sudo docker cp /home/ubuntu/envs/.env ubuntu_jenkins'"

                }

                sh'''
                if [ -d ./media ]; then
                rm -r media
                fi
                mkdir media
                '''

                sh '''
                docker build -t backdb ./backend
                '''

                echo 'Bulid Docker'
                // dir('backend'){
                //     script{
                //         dockerImage = docker.build imagename
                //     }
                // }
                echo 'Bulid Docker2'

            }
        }

        stage('SSH-Server-EC2'){
            steps {
                echo 'SSH'

                sshagent(credentials: ['ec2_ssh_key']) {

                    sh "ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io 'sudo docker run -i --name back -p 9999:8000 -d -v /"

                }
            }
        }

    }
}
```

front pipeline

```
pipeline {
  agent any

  tools {nodejs "node16_19"}

  environment {
    imagename = "johjaewan/sample"
    registryCredential = 'dockerhub_key'
    dockerImage = ''
  }

  stages {

    stage('gitlab clone') {
      steps {
        git branch: 'FE', credentialsId: 'gitlab_token', url: 'https://lab.ssafy.com/s08-ai-speech-sub2/S08P22D103.git'
      }
    }
    stage('build'){
      steps{
        dir('frontend/frontproject'){
          sh "npm i"
          sh "CI=false npm run build"
          sh'''
          cd src
          if [ -d ./media ]; then
            rm -r media
          fi
          mkdir media
          cd ..
          '''
        }
      }
    }

    stage('docker image'){
      steps {
        sh """
        if ! test docker; then
          curl -fsSL https://get.docker.com -o get-docker.sh
          sh get-docker.sh
          fi
        """

        sshagent(credentials: ['ec2_ssh_key']) {

          sh '''
          if test "`docker ps -aq --filter ancestor=front`"; then

            ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker stop $(docker ps -aq --filter ancestor=front)"
            ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker rm -f $(docker ps -aq --filter ancestor=front)"
            ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io "sudo docker rmi front"
            fi
            '''

          sh "ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io 'sudo docker cp /home/ubuntu/dfile/Dockerfile ubuntu_"

        }

        sh '''
        docker build -t front ./frontend/frontproject
        '''

        echo 'Bulid Docker'

        echo 'Bulid Docker2'

      }
    }

    stage('SSH-Server-EC2'){
      steps {
        echo 'SSH'

        sshagent(credentials: ['ec2_ssh_key']) {
```

```

}
}
}
}
sh "ssh -o StrictHostKeyChecking=no ubuntu@j8d103.p.ssafy.io 'sudo docker run -i --name front -p 3000:3000 -d -v
```

Web hook 설정

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://jd103.p.ssafy.io:7777/project/test> ?

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

- ☒ Approved Merge Requests (EE-only)
- ☒ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급 >

해당 부분을 체크해주고 ‘고급’을 눌러서

- ☐ 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

Clear

Secret token에 Generate 버튼을 누르면 토큰이 생성 된다 이걸 복사해서

GitLab 프로젝트에 Settings → Webhooks 에서 생성해준다.

URL 에는 젠킨스 프로젝트 주소를 넣어준다. 위의 이미지에서 체크된 항목 Build when a change ... 뒤에 있는 주소를 넣으면 된다.

Secret token 에는 아까 복사해둔 걸 넣어준다.

Push events 에는 어떤 브랜치의 push 이벤트를 듣고 싶은지 넣어준다.

그리고 add webhook 를 눌러주고 테스트를 해서 200이 나오면 정상이다.

Nginx

엔진엑스 설치와 버전 확인

```
sudo apt-get install nginx
nginx -v
```

letsencrypt 인증서 발급

```
sudo apt-get install letsencrypt

sudo systemctl stop nginx

sudo letsencrypt certonly --standalone -d www제외한 도메인 이름
```

이걸 실행했을 때 Congratulations! 이 보이면 성공한거고

시간이 뜨면서 기다려 달라는 메시지를 띄우면 그 시간 까지 기다렸다 발급받으면 된다.

`/etc/nginx/sites-available` 로 이동한 이후

`sudo vi proxy-setting` 파일을 하나 만들고

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    # 도메인 이름
    server_name j8d103.p.ssafy.io;

    location /{
        proxy_pass http://localhost:3000;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /api {
        proxy_pass http://localhost:8000/api;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    #도메인 이름
    server_name j8d103.p.ssafy.io;

    location /{
        proxy_pass http://localhost:3000;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /api {
        proxy_pass http://localhost:8000/api;
    }

    ssl_certificate /etc/letsencrypt/live/i8d208.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i8d208.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
```

80은 http, 443은 https 로 접근하는 것을 의미한다.

In 명령어를 실행

```
sudo ln -s /etc/nginx/sites-available/proxy-setting /etc/nginx/sites-enabled/proxy-setting
```

테스트 후 재시작 해주면 된다.

```
sudo nginx -t
```

```
sudo systemctl restart nginx
```

default 파일과 proxy-setting 에서 둘다 80번 포트를 써서 오류가 날 수 있는데 이럴 때는

`/etc/nginx/sites-enabled` 에서 default 에 들어가서

```
sudo vi default
```

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server;
```

이 부분에 80을 다른 안쓰는 포트로 바꿔주면 된다.