📘 Porting Manual

삼성청년 SW 아카데미 서울캠퍼스 9기 자율 프로젝트 포팅 매뉴얼

1. **프로젝트 주제 : 리밋 (RE:MEET)**
2. **프로젝트 기간** : 2023.10.09. - 2023.11.17.
3. **개발 인원** : 이승준(팀장), 김영우, 김승우, 황상미, 임병국, 이민웅
4. **담당 코치** : 김나연, 최웅렬

리밋(RE:MEET) 포팅 매뉴얼 목차

1. 프로젝트 기술 스택

A. FE

- **기술 스택**
  - React `18.2.0`
  - TypeScript `5.0.2`
  - Vite `4.4.5`
  - Node.js `18.16.1`
  - yarn `1.22.19`
  - Styled Components `6.1.0`
  - react-query `4.3.3`

- **개발 환경**
  - Visual Studio Code `1.80.1`

B. BE

- **기술 스택**
  - Java Open-JDK zulu `11.0.16`
  - SpringBoot `2.7.15`
    - Gradle `8.2.1`
    - jbcrypt `0.3`
    - jjwt `0.11.2`
    - lombok
    - JPA
  - Redis `7.2.1`
  - MySQL `8.0.26`
  - Nginx `1.18.0`
  - Flask `3.0.0`
    - ffmpeg-python `0.2.0`
    - pydub `0.25.1`
    - boto3 `1.28.77`
    - google-cloud-speech `2.22.0`
    - moviepy `1.0.3`

- **개발 환경**
  - IntelliJ `2021.2.4`
  - Visual Studio Code `1.80.1`
  - MobaXterm `23.2`
  - Postman `10.16.3`

D. etc.

- **기술 스택**
  - Jenkins `2.422`
  - Docker `24.0.6`

- **개발 환경**
  - AWS EC2 `Ubuntu 20.04.3 LTS`
  - GitLab `16.0.5`
  - MatterMost `7.8.6`

2. 프로퍼티 설정

a. Jenkinsfile

[Backend] SpringBoot Server

▼ Pipeline Script

```
pipeline {
    agent any

    tools {
        gradle 'Gradle 8.4'
        jdk 'OpenJDK 11'
    }

    environment {
        // SPRING_PROFILES_ACTIVE = 'prod'
        DOCKER_IMAGE_NAME = 'my-springboot-app'
        DOCKER_CONTAINER_NAME = 'my-springboot-container'
        GIT_BRANCH = 'develop_backend'
        GIT_REPO_URL = 'https://lab.ssafy.com/s09-final/S09P31A706.git'
        GIT_CREDENTIALS_ID = 'remeet'
    }

    stages {
        stage('Clone Repository') {
            steps {
                git branch: "${GIT_BRANCH}",
                    credentialsId: "${GIT_CREDENTIALS_ID}",
                    url: "${GIT_REPO_URL}"
                sh 'pwd'
            }
        }



        stage('Prepare application.yml') {
            steps {
                script {
                    // application.yml 파일이 있는 경로로 이동
                    dir('dev/BE') {
                        // sed를 사용하여 주석 제거
                        sh "sed -i '/^#.*ssl:/s/^#//g' src/main/resources/application.yml"
                        sh "sed -i '/^#.*key-store:/s/^#//g' src/main/resources/application.yml"
                        sh "sed -i '/^#.*key-store-password:/s/^#//g' src/main/resources/application.yml"
                        sh "sed -i '/^#.*key-alias:/s/^#//g' src/main/resources/application.yml"
                        sh "sed -i '/^#.*key-store-type:/s/^#//g' src/main/resources/application.yml"
                    }
                }
            }
        }



        stage('Build Spring Boot App') {
            steps {
                dir('dev/BE'){
                    sh 'gradle clean build -x test'
                }
            }
        }

        stage('List JAR files') {
            steps {
                dir('dev/BE/build/libs') {
                    sh 'ls *.jar'
                }
            }
        }

        stage('Prepare Docker Context and Build Image') {
            steps {
                dir('dev/BE') {
                    // app.jar 파일을 현재 디렉토리로 복사
                    sh 'cp build/libs/REMEET-0.0.1-SNAPSHOT.jar .'
                    // Docker 이미지 빌드 (현재 디렉토리가 빌드 컨텍스트)
                     sh 'docker-compose -f docker-compose.yml up -d --build'
                }
            }
```

```
        }

        stage('Cleanup Unused Docker Images') {
            steps {
                script {
                    // Cleanup dangling and <none> tagged images
                    sh 'docker image prune -f'
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|De
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|De
                )
            }
        }
    }
}
```

[Backend] Flask Server

▼ Pipeline Script

```
pipeline {
    agent any

    environment {
        // 환경 변수 설정
        DOCKER_IMAGE_CONVERSATION = 'my-flask-conversation-app'
        DOCKER_CONTAINER_CONVERSATION = 'my-flask-conversation-container'
        DOCKER_IMAGE_GOOGLESTT = 'my-flask-googlestt-app'
        DOCKER_CONTAINER_GOOGLESTT = 'my-flask-googlestt-container'
        GIT_BRANCH = 'develop_backend'
        GIT_REPO_URL = 'https://lab.ssafy.com/s09-final/S09P31A706.git'
        GIT_CREDENTIALS_ID = 'remeet'
        // 포트 번호는 각각의 서비스에 따라 구분하여 할당합니다.
        CONVERSATION_PORT = '5001'
        ENV_FILE_PATH = "${WORKSPACE}/.env" // .env 파일이 저장될 경로 설정

    }

    stages {
        stage('Prepare Environment') {
            steps {
                script {
                    // .env 파일 내용 생성
                    writeFile file: "${ENV_FILE_PATH}", text: """
                        OPENAI_API_KEY=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
                        ELEVENLABS_API_KEY=0cd38d1a7e725419c3599d6db9f58885
                        x_api_key=Y2JhNmI1ZjQ3MjEyNDJkNGFmOTdlZDRiNzYxYmJjZjgtMTY5NzAxMDQ3Mw==
                        AWS_ACCESS_KEY_ID=AKIAQO3J7YYNIYPNTM5E
                        AWS_SECRET_ACCESS_KEY=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
                    """.stripIndent()
                }
            }
        }




        stage('Clone Repository') {
            steps {
                git branch: "${GIT_BRANCH}",
                    credentialsId: "${GIT_CREDENTIALS_ID}",
                    url: "${GIT_REPO_URL}"
```

```
            }
        }


        stage('Build and Run Containers') {
            steps {
                script {

                    // conversation.py를 위한 Docker 컨테이너 빌드 및 실행
                    dir('dev/FLASK') {
                        // sh 'docker build -f Dockerfile.conversation  -t ${DOCKER_IMAGE_CONVERSATION} .'
                        // sh 'docker rm -f ${DOCKER_CONTAINER_CONVERSATION} || true'
                        // sh 'docker run -d -p 5000:5000  --env-file ${ENV_FILE_PATH} --name ${DOCKER_CONTAINER_CONVERSATION} ${D
                         sh 'docker-compose -f docker-compose.yml up -d --build'

                    }

                }
            }
        }

        stage('Cleanup Unused Docker Images') {
            steps {
                script {
                    // Cleanup dangling and <none> tagged images
                    sh 'docker image prune -f'
                }
            }
        }
    }

    post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend (
                color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|Detail
            )
        }
    }
    failure {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend (
                color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|Detail
            )
        }
    }
}

}
```

[Frontend] Node.js Server

▼ Pipeline Script

```
pipeline {
    agent any
    tools {
        nodejs 'nodejs'
    }

    stages {
        stage('gitlab clone') {
            steps {
                git branch: 'develop_frontend',
                credentialsId: 'remeet',
                url: 'https://lab.ssafy.com/s09-final/S09P31A706.git'
                sh 'pwd'
            }
        }

        stage('Check npm') {
            steps {
                sh 'echo $PATH'
                sh 'node -v'
            }
```

```
        }

        stage('Build React App') {
            steps {
                dir('dev/FE/'){

                    script {
                // Create .env file with the API base URL
                writeFile file: '.env', text: 'VITE_API_BASE_URL=https://k9a706.p.ssafy.io:8443/api/v1/'
            }
            sh 'cat .env' // To verify that .env file contains the correct content



                    sh 'yarn install'
                    sh 'yarn build'
                }
            }
        }


        stage('Prepare Docker Context and Build Image') {
        steps {
            dir('dev/FE') {
                // Docker 이미지 빌드 (Dockerfile.nginx 파일 사용)
                sh 'docker build --no-cache -t nginx -f Dockerfile.nginx .'
            }
        }
    }

        stage('Run Nginx Container') {
            steps {
                script {
                    // Stop and remove old container if it exists
                    sh 'docker rm -f my-nginx-container || true'
                    // Run a new container from the new image
                    sh 'docker run -d   -v ssl-certificates:/ssl-certificates -v /etc/nginx/nginx.conf:/etc/nginx/nginx.conf -p 80

                }
            }
        }

        stage('Cleanup Unused Docker Images') {
            steps {
                script {
                    // Cleanup dangling and <none> tagged images
                    sh 'docker image prune -f'
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|Detail
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|Detail
                )
            }
        }
    }
}
```

b. Dockerfile

[Backend] SpringBoot Server (Dockerfile)

```
# 베이스 이미지 설정
FROM openjdk:11-jre-slim
```

```
# 8000번 포트 노출
EXPOSE 8000

# 애플리케이션 파일을 컨테이너 내부로 복사
COPY REMEET-0.0.1-SNAPSHOT.jar /app/app.jar

# 작업 디렉토리 설정
WORKDIR /app

# 애플리케이션 실행
CMD ["java", "-jar", "app.jar"]
```

[Backend] Flask Server (Dockerfile.conversation)

```
# Dockerfile_conversation
FROM python:3.9

# 작업 디렉토리 설정
WORKDIR /app

# 의존성 파일 복사 및 설치
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# 나머지 소스 코드 복사
COPY . .

# Flask 서버 실행 포트 설정
EXPOSE 5001

# Flask 애플리케이션 실행
CMD ["python", "conversation.py"]
```

[Backend] Flask Server (Dockerfile.googlestt)

```
# Dockerfile_googlestt
FROM python:3.9

# 작업 디렉토리 설정
WORKDIR /app

# 의존성 파일 복사 및 설치
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# 나머지 소스 코드 복사
COPY . .

# Flask 서버 실행 포트 설정
EXPOSE 5002

# Flask 애플리케이션 실행
CMD ["python", "googlestt.py"]
```

[Frontend] Node.js Server (Dockerfile.nginx)

```
# 베이스 이미지 설정
FROM nginx:latest


# 빌드된 React 앱 파일 복사
COPY dist /usr/share/nginx/html

# 80 포트와 443 포트 열기
EXPOSE 80 443

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

c. Nginx

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;

    types_hash_max_size 2048;

server {
    listen 80;
    server_name k9a706.p.ssafy.io;
    return 301 https://$host$request_uri;



}


#HTTPS 서버 설정 (443 SSL)
server {
        listen 443 ssl;
        server_name k9a706.p.ssafy.io;

        ssl_certificate /ssl-certificates/fullchain.pem;
        ssl_certificate_key /ssl-certificates/privkey.pem;


        # SSL 설정 추가


        root /usr/share/nginx/html;


        location / {
                try_files $uri $uri/ /index.html;
}
}

 # server_tokens off;

    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off



    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ##
    # SSL Settings
    ##

    # ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
    # ssl_prefer_server_ciphers on;

    ##
    # Logging Settings
    ##

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    ##
    # Gzip Settings
    ##

    gzip on;
    gzip_disable "msie6";

    # gzip_vary on;
```

```
    # gzip_proxied any;
    # gzip_comp_level 6;
    # gzip_buffers 16 8k;
    # gzip_http_version 1.1;
    # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascri

    ##
    # Virtual Host Configs
    ##

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

d. Flask server

```
sudo docker build -t my-flask-conversation-app .

sudo docker run -d --name my-flask-conversation-container my-flask-conversation-app
```

e. .gitignore

SpringServer(dev/BE/)

```
HELP.md
.gradle
build/
!gradle/wrapper/gradle-wrapper.jar
!**/src/main/**/build/
!**/src/test/**/build/

### STS ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache
bin/
!**/src/main/**/bin/
!**/src/test/**/bin/

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr
out/
!**/src/main/**/out/
!**/src/test/**/out/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/

### VS Code ###
.vscode/
```

FrontServer(dev/FE/)

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*
```

```
node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
```

FlaskServer(dev/Flask/)

```
.env
```

f. FE property

Nginx default 세팅 방법 : 3-c 참고

```
# FE/src/constants/url.ts
export const BASEURL = 'https://k9a706.p.ssafy.io/api/v1';
```

g. BE property

`application.yml`

```
# application.yml
server:
  servlet:
    context-path: /api/v1
  port: 8443
#  ssl:
#    enabled: true
#    key-store: file:/etc/letsencrypt/keystore.p12
#    key-store-password: 1q2w3e4r1q2w3e4r
#    key-alias: yourkeyalias
#    key-store-type: PKCS12

spring:
  servlet:
    multipart:
      max-file-size: 20MB
      max-request-size: 20MB
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k9a706.p.ssafy.io:3306/mydb?serverTimezone=Asia/Seoul
    username: {$USERNAME}
    password: {$PASSWORD}
    hikari:
      maximum-pool-size: 20
  jpa:
    database: mysql
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        default_batch_fetch_size: 100
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
    open-in-view: false
    show-sql: false
  redis:
    port: 6379
```

```
    host: k9a706.p.ssafy.io
    password: 1q2w3e4r@@#

jwt:
  secretKey: rkGU45258GGhiolLO2465TFY5345kGU45258GGhiolLO2465TFY5345seeew
  access:
    expiration: 10000 # 1시간(60분) (1000L(ms -> s) * 60L(s -> m) * 60L(m -> h))
    header: Authorization

  refresh:
    expiration: 1209600000 # (1000L(ms -> s) * 60L(s -> m) * 60L(m -> h) * 24L(h -> 하루) * 14(2주))
    header: Authorization_refresh
```

h. 사용 포트

Backend

- SpringBoot Server: `8000`
- Flask Conversation Server: `5001`
- Flask Google STT Server: `5002`

Frontend

- Node.js Server (Nginx): `80` (HTTP), `443` (HTTPS)

Database

- MySQL: `3306`

기타

- Jenkins: `8080`
- Redis: `6379`

3. 도구 설치 및 설정

A. AWS EC2 설정

방화벽 설정 - ufw

- ufw 활성화 / 비활성화

```
$ sudo ufw enable
$ sudo ufw disable
```

- ufw 상태 확인

```
$ sudo ufw status
```

- ufw 상태 및 등록된 rule 확인

```
$ sudo ufw status numbered
```

- 사용할 포트 허용

```
$ sudo ufw allow [PORT]
```

- 등록한 포트 조회

```
$ sudo ufw show added
```

- 등록한 포트 삭제

  (중요) 삭제한 정책은 반드시 enable을 수행해야 적용된다.

```
$ sudo ufw delete [PORT]
```

B. MySQL 설치 및 설정

1. **AWS EC2 내에 MySQL 설치**

```
# apt(ubuntu에서 사용하는 package 관리 모듈) upgrade
$ sudo apt update
# MySQL 설치
$ sudo apt install mysql-server
# MySQL 버전 확인
$ mysql --version
```

2. **MySQL 로그인**

```
$ sudo mysql

# 아이디 [id], 비밀번호 [password]로 계정 생성
$ create user 'id'@'%' IDENTIFIED BY 'password';

# 'id' 에 권한 부여. '%'은 모든 외부 IP를 의미
# 앞 * : DB 이름, 뒤 * : 권한 내용
$ grant all privileges on *.* to 'id'@'%' with grant option;

# mysql 나가기
$ exit

# 방금 만든 admin account로 로그인하기
$ sudo mysql -u [id] -p [password]
```

3. **Database & Table 생성**

```
# database 생성
$ create database practice default CHARACTER SET UTF8;

# database 생성되었는 지 확인
$ show databases; / show schemas;

# database 접속
$ use practice;
```

```
# table 생성
CREATE TABLE menus (
    menu_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    menu_name VARCHAR(20) NOT NULL,
    menu_description TEXT NOT NULL
    );

# table에 데이터 기입
INSERT INTO menus
    (menu_name, menu_description)
     VALUES
    ("닐라닐라 바닐라", "베리베리스트로베리 친구");

INSERT INTO menus
    (menu_name, menu_description)
     VALUES
    ("복숭아 아이스티", "내가 제일좋아하는 티");

INSERT INTO menus
```

```
    (menu_name, menu_description)
     VALUES
    ("카페라떼", "Latte is horse");

# table 내용 확인
select * from menus;

# sql 나가기
exit
```

4. **Listen IP 대역폭 변경**

```
# 현재 mysql은 localhost에서만 접속 가능하므로,
# 모든 IP에서 원격접속할 수 있도록 수정할 예정

$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

```
# 127.0.0.1 ← aws의 localhost / 같은 IP인 경우에만 접근이 가능한 상황
# bind-address : local-host 수정

# 주의! 모든 IP에 대해 접근을 허용하므로, 보안 상 문제 --> 나중에 조치를 취할 것

bind-address : 0.0.0.0
```

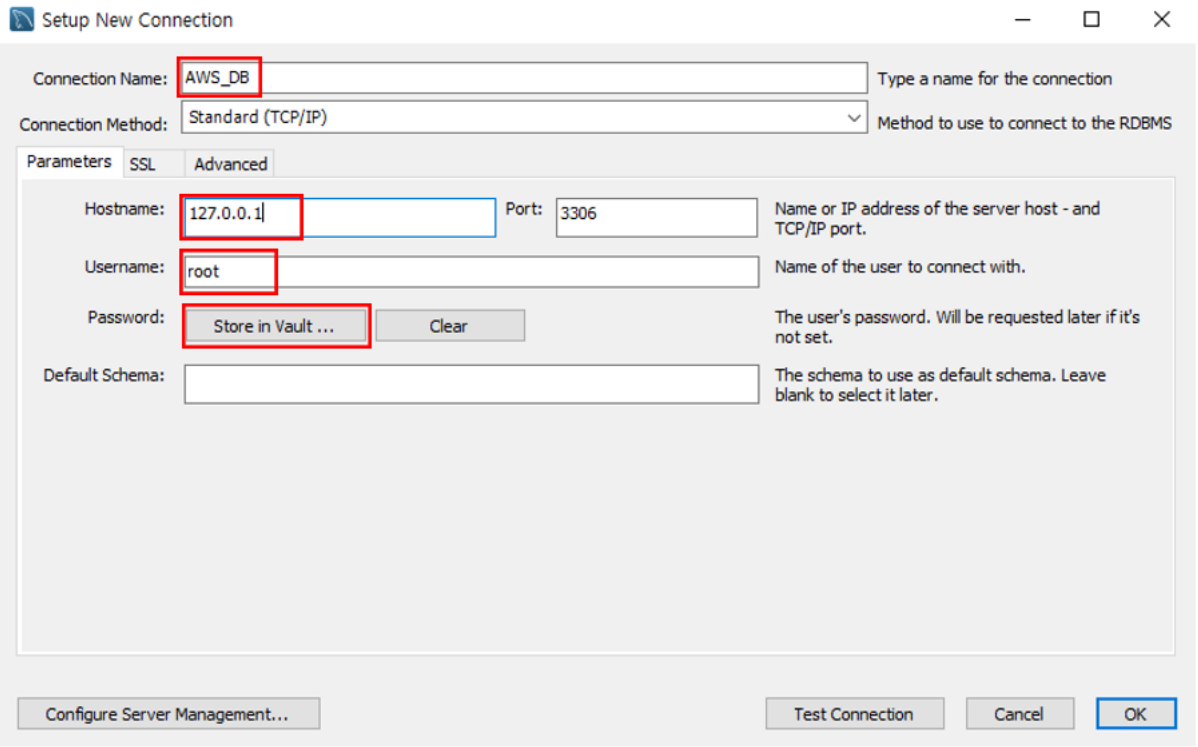
```
# If MySQL is running as a replication slave, this should be
# changed. Ref https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_tmpdir
# tmpdir                = /tmp
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
mysqlx-bind-address     = 127.0.0.1
#
# * Fine Tuning
#
key_buffer_size         = 16M
# max_allowed_packet    = 64M
# thread_stack          = 256K

# thread_cache_size       = -1
```

5. **MySQL 재시작**

```
$ sudo service mysql restart
```

6. **Workbench 연결**



- **Connection Name** : 내가 원하는 이름으로 설정
- **Hostname** : AWS EC2의 `public IPv4(or DNS)` 주소를 기입
- **Username** : 2번에서 작성한 `id`로 수정
- **Password** : `Store in Vault …`를 클릭해서, 2번에서 작성한 `password`를 기입

C. Nginx 설치 및 설정

1. **설치**

```
$ sudo apt update
$ sudo apt install nginx
```

2. **상태 확인**

```
$ systemctl status nginx
```

3. **실행 시작 / 중지**

```
$ sudo systemctl start nginx
$ sudo systemctl stop nginx
```

4. **Nginx 환경 설정**

```
$ sudo vi /etc/nginx/nginx.conf
```

- 이 후, 2-C 번 참조

5. **SSL 설정 (https) - Let's Encrypt & Certbot**

```
# Let's Encrypt 설치
$ sudo apt-get insatll letsencrypt

# Certbot 설치
$ sudo apt-get install certbot python3-certbot-nginx

# Certbot 동작
$ sudo certbot --nginx

## 추가 : 방화벽 기본 포트 설정
$ sudo ufw allow ssh
$ sudo ufw allow http
$ sudo ufw allow https
```

D. Docker 설치 및 설정

1. 우분투 시스템 패키지 업데이트

```
sudo apt-get update
```

2. 필요 패키지 설치

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
# apt가 https 저장소를 사용할 수 있게 해주는 package
```

3. Docker 공식 GPG key 추가

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# Docker 설치를 위한 gpg를 다운 받고, Docker 저장소 키를 apt에 등록
```

4. Docker 공식 apt 저장소를 추가

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
# Ubuntu 버전에 맞는 Docker를 다운로드 및 repository 리스트에 추가
# - stable : 안정화된 Docker 버전을 의미.  가장 최신 release version.
# - nightly : 차기 출시를 위해 진행중인 최신 빌드 버전
# - test : 검증 되기 이전의 가장 최신 빌드 버전
# 안정성을 위해 stable 버전 사용 권장
```

5. 시스템 패키지 업데이트

```
sudo apt-get update
```

6. Docker 설치

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
# docker-ce란? docker의 무료로 제공되는 docker engine
```

7. Docker 설치 확인

7-1 Docker 실행 상태 확인

```
sudo systemctl status docker
```

E. Docker Compose설치 및 설정

1. Docker Compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/download/v2.6.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/doc
```

2. Docker Compose 실행권한 부여

```
sudo chmod +x /usr/local/bin/docker-compose
```

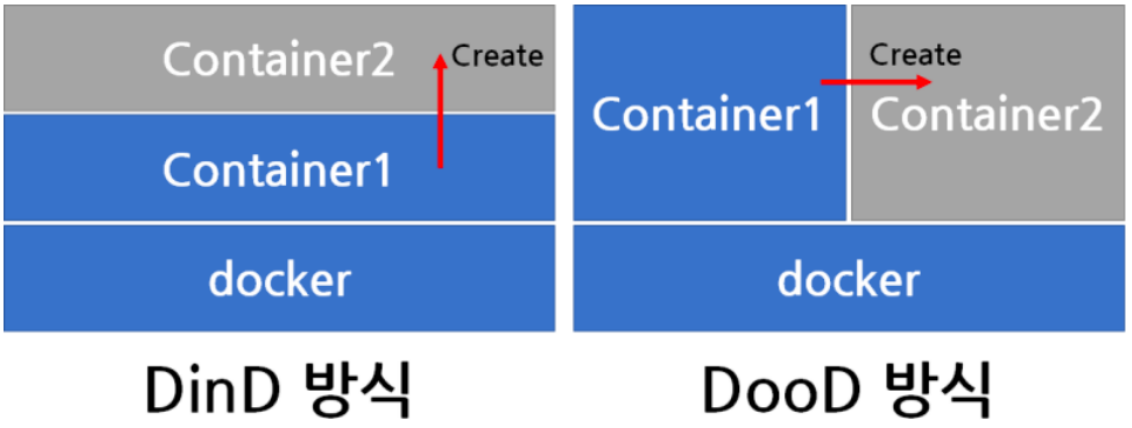
3. Docker Compose 버전 확인

```
docker-compose --version
```

F. Jenkins 설치 및 설정

▼ 중요!

- 설치하기에 앞서 DinD, DooD에 대한 개념을 알고 넘어가자.
- **DinD**는 Docker in Docker의 약어로, 도커 컨테이너 내부에 호스트와는 별개로 새로운 도커를 실행시키는 것.
- **DooD**는 Docker out of Docker의 약어로, 호스트 도커가 사용하는 소켓을 공유하여 도커 클라이언트 컨테이너에서 컨테이너를 실행시키는 것.
- 쉽게 설명해서 Jenkins Container 내에서 파이프라인을 작성하고 Nginx Web Server, SpringBoot 등을 빌드할 때, DinD는 Jenkins Container 내부에 빌드를, DooD는 호스트에서 새로운 Container를 빌드하는 것이다.



- 참고
  - https://mns010.tistory.com/25
  - https://velog.io/@inhwa1025/Docker-컨테이너-내부에서-Docker-사용
  - DooD 문제로 헤맸을 때 참고했던 블로그
  - https://velog.io/@hind_sight/Docker-Jenkins-도커와-젠킨스를-활용한-Spring-Boot-CICD

- DooD 방식으로 Jenkins 를 설치할 때 2가지 방식이 존재.
- 빌드된 이미지를 사용하거나 Dockerfile을 직접 생성하는 방법
- Dockerfile을 직접 생성해서 설치.

Dockerfile 직접 생성

```
vi Dockerfile
# 이후 i를 눌러서 수정을 활성화하고,
# 수정이 완료됐을 때, esc을 누르고 :wq를 입력하여
# 저장한다.
```

Dockerfile 생성

```
FROM jenkins/jenkins:lts
USER root

# jenkins docker 내부에 Docker 관련 파일들을 설치하는 명령어
RUN apt-get update && \
    apt-get -y install apt-transport-https \
      ca-certificates \
      curl \
      gnupg2 \
      software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
      "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
      $(lsb_release -cs) \
      stable" && \
   apt-get update && \
   apt-get -y install docker-ce

# jenkins의 docker 권한을 얻기 위해 docker 그룹에 포함시키기 위함
RUN groupadd -f docker
RUN usermod -aG docker jenkins
```

입력한 명령어

```
# Jenkins LTS, JDK 11
FROM jenkins/jenkins:lts-jdk11

USER root

# Docker 설치를 위한 필요한 패키지들을 설치
RUN apt-get clean && \
    apt-get update && \
    apt-get -y install apt-transport-https \
      ca-certificates \
      curl \
      gnupg2 \
      software-properties-common \
      lsb-release

# Docker 공식 GPG 키 추가
RUN curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg | apt-key add -

# Docker APT 리포지토리 추가
RUN add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
   $(lsb_release -cs) \
   stable"

# Docker 설치
RUN apt-get update && \
    apt-get -y install docker-ce-cli

# jenkins 유저를 docker 그룹에 추가
RUN groupadd -f docker && \
    usermod -aG docker jenkins

# jenkins 사용자로 전환하여 Jenkins 플러그인을 설치
# USER jenkins

# Jenkins에 필요한 플러그인을 설치
# Gradle 플러그인을 추가하여 Gradle 프로젝트 지원
# 아래 부분 안될시 주석처리하고 jenkins ui에서 수동설치
# RUN jenkins-plugin-cli --plugins "blueocean docker-workflow gradle"
```

Docker Image Build 명령어

```
docker build -t [이미지명] .
```

- 이후 `Docker images` 명령어를 통해 확인한다.

Docker Container 실행

```
docker run \
  -p 8080:8080 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  --name my-jenkins \
  [이미지명]
```

실행시 입력한 명령어

```
docker run --name jenkins_container --detach -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock jenkins
```

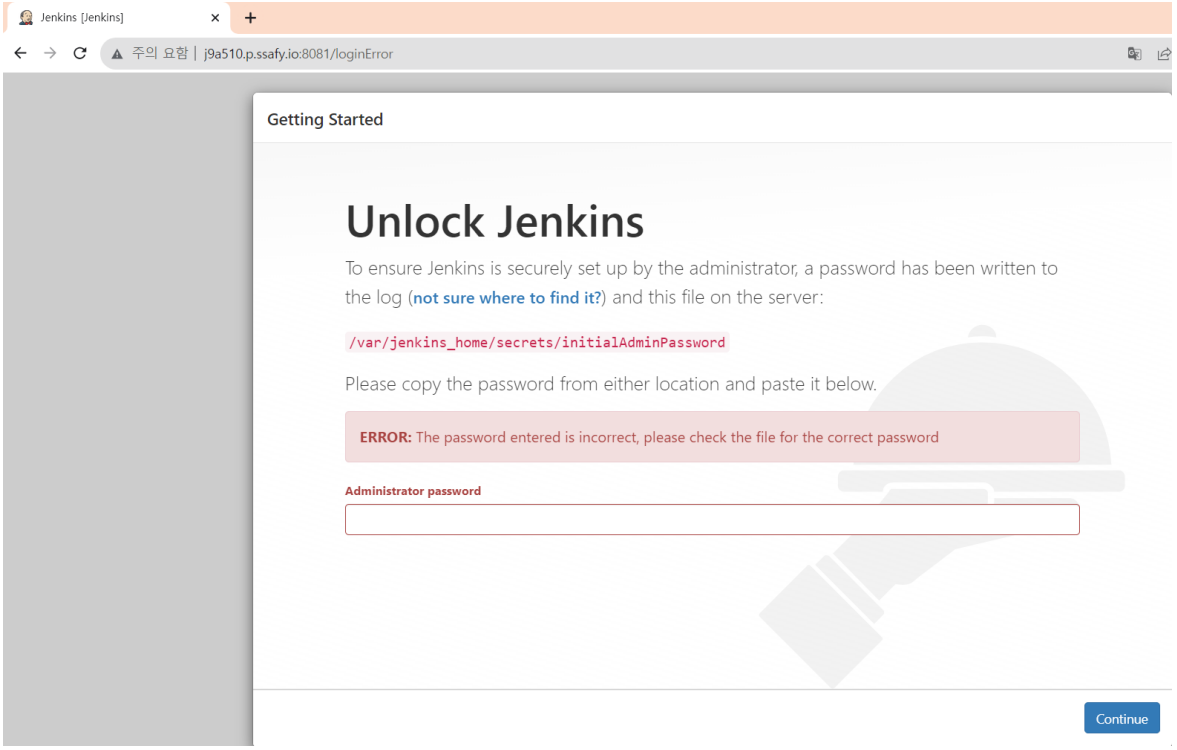
추가1. 관리자 초기 비밀번호 찾기

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

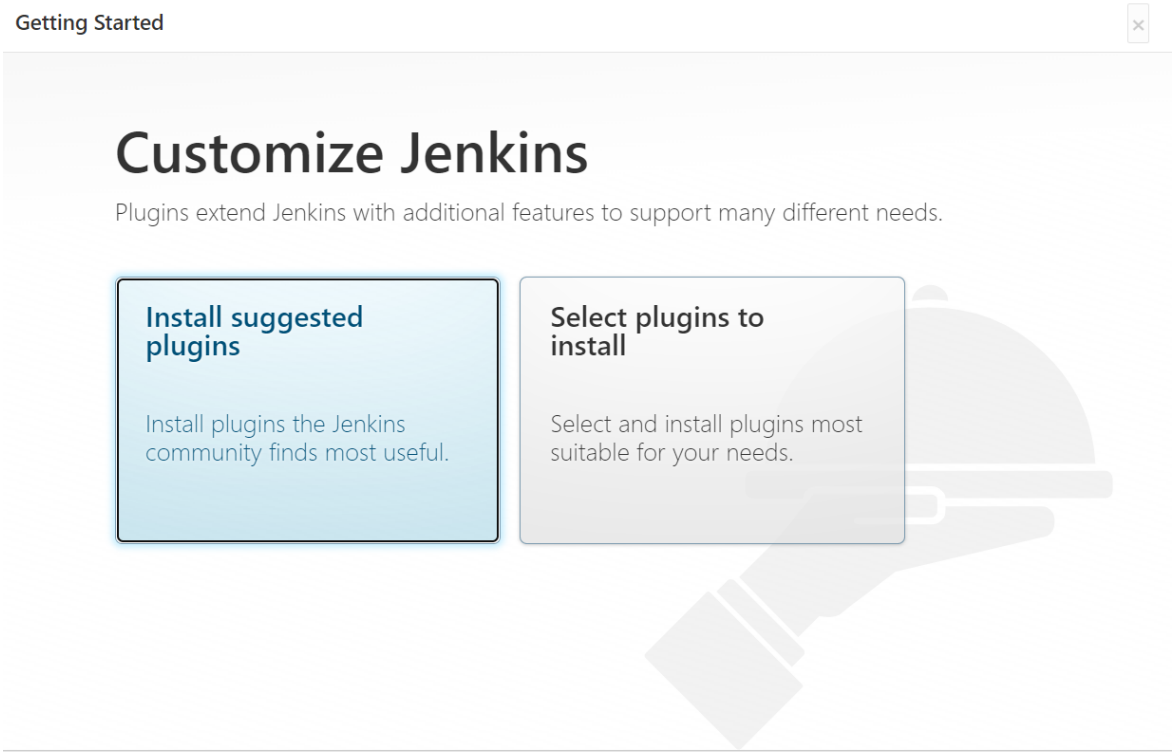
추가2. 젠킨스 도메인 페이지 접속해서 해야하는 일들

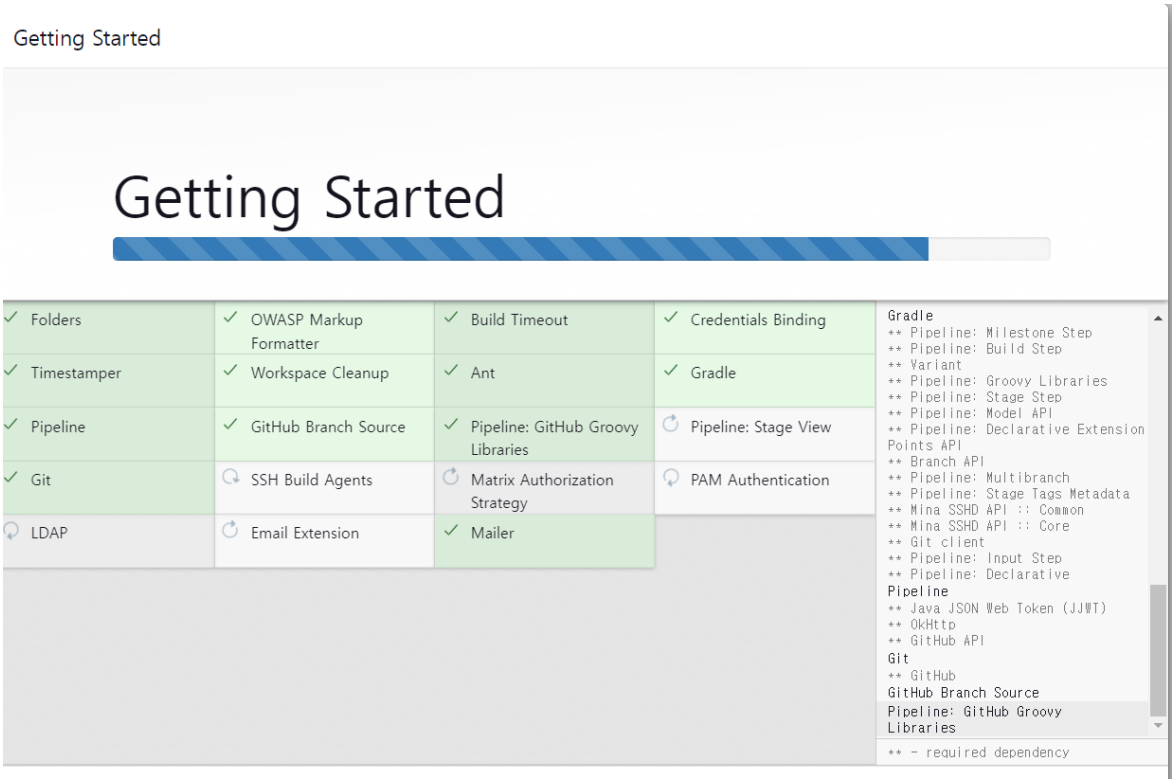
▼ 목록

1. 관리자 초기 비밀번호로 로그인

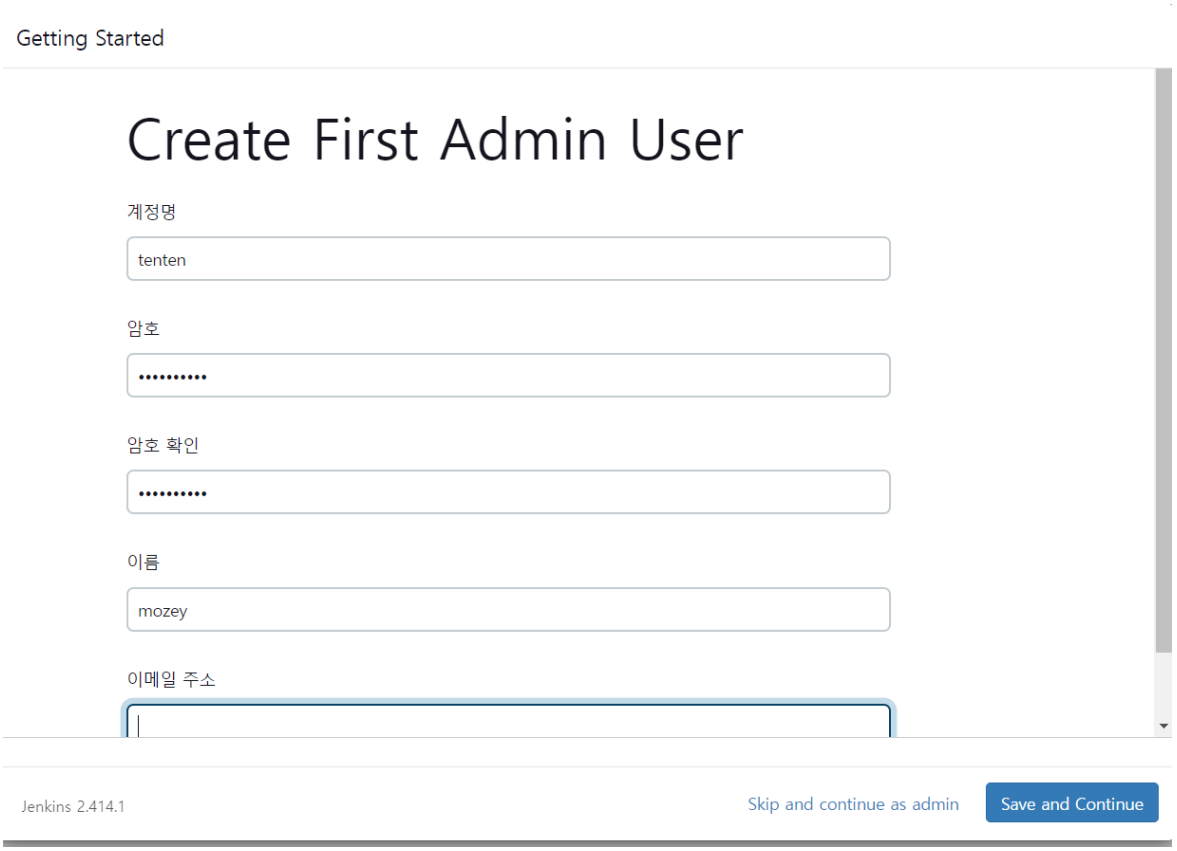


2. 플러그인 인스톨


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.
Jenkins 2.289.3


Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline: Stage View
Git
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
Email Extension
Mailer
Gradle
** Pipeline: Milestone Step
** Pipeline: Build Step
** Variant
** Pipeline: Groovy Libraries
** Pipeline: Stage Step
** Pipeline: Model API
** Pipeline: Declarative Extension Points API
** Branch API
** Pipeline: Multibranch
** Pipeline: Stage Tags Metadata
** Mina SSHD API :: Common
** Mina SSHD API :: Core
** Git client
** Pipeline: Input Step
** Pipeline: Declarative
Pipeline
** Java JSON Web Token (JJWT)
** OkHttp
** GitHub API
Git
** GitHub
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
** - required dependency

3. 계정 등록



Jenkins Container의 shell에 접속

```
docker exec -it -u root [컨테이너명] bash
```

Docker 그룹에 권한 부여

```
chown root:docker /var/run/docker.sock
```

컨테이너 재시작

```
docker restart [젠킨스 컨테이너 명]
```

4. 빌드 방법

A. BackEnd(Springboot)

1. 프로젝트에서, `dev/BE` 폴더로 이동 (Spring 루트디렉토리)
2. 다음 명령어를 통해 빌드

```
$ ./gradlew build
```

3. 명령어가 성공하면, 현재 위치한 프로젝트 내에 빌드 파일(.jar)이 위치한 `build/libs` 폴더가 생성

B. BackEnd(Flask)

** 가상환경 생성 **

```
# 가상 환경 생성
python -m venv venv

# 가상 환경 활성화
# Windows의 경우:
.\venv\Scripts\activate
# macOS 및 Linux의 경우:
source venv/bin/activate
```

1. 의존성 설치

```
pip install -r requirements.txt
```

2. 환경 변수 설정

```
# Windows
set FLASK_APP=conversation.py
# macOS 및 Linux
export FLASK_APP=conversation.py
```

3. 서버 실행

```
flask run
```

C. FrontEnd

1. 프로젝트에서, `dev/FE` 폴더로 이동 (Front 루트디렉토리)
2. 다음 명령어를 통해 패키지 다운로드

```
$ yarn install
```

3. 패키지 다운로드를 통해 node_modules 폴더가 생성됐으면, 다음 명령어를 통해 빌드

```
$ yarn build
```

5. 외부 서비스 목록

LLM(Chat Gpt)

- 대화모델 프롬프트 생성 및 응답 텍스트 생성
- `https://chat.openai.com/`

ElevenLabs(TTS)

- 음성모델 학습 및 생성, TTS 기능 활용
- `https://elevenlabs.io/`

HeyGen(Stable Diffusion)

- 대화 얼굴 모델 생성 및 대화영상 제작
- `https://app.heygen.com/home`