



1. 빌드 및 배포

1. 환경

Frontend : Vue3, Vite, UnoCSS, VueUse, Pinna, PNPM

Chrome-Extension : Vue3, Vite, UnoCSS, PNPM

Backend: Azul-Zulu version 17.0.9, Spring Boot 3.1.5, Spring Data Mongo

DB: Redis, MongoDB

Infra: Ubuntu 20.04 LTS, AWS EC2, GitLab CICD, Jenkins:jdk17, Gradle, nginx

기타: Figma, Jira, GitLab, Mattermost, Webex, Notion

2. 환경 변수 형태

1. .env.development (frontend)

```
프론트 주소
네이버 API Client ID
구글 API Client ID

REACT_APP_LOGIN_SERVER_URL=https://i9b108.p.ssafy.io
REACT_APP_NAVER_CLIENT_ID=sOLMK10cIu3pqoFbtMey
REACT_APP_GOOGLE_CLIENT_ID=953911532873-0ve3ob0gtc2eq0fdp8ui67mue02pufpr.apps.googleusercontent.com
```

2. mail

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: {user name}
    password: {user secret}
    properties:
      mail:
        smtp:
          auth: true
          timeout: 5000
          starttls:
            enable: true
```

3. MongoDB

```
spring:
  data:
    mongodb:
      host: {mongodb host}
      port: {mongodb port}
      username: {mongodb username}
      password: {mongodb password}
      authentication-database:
        database:
          uri: {mongodb uri}
```

4. Oauth

```

spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: {Google OAuth id}
            client-secret: {Google OAuth Secret}
            redirect-uri: http://sapier.co.kr/api/login/oauth2/code/google
            scope:
              - email
              - profile

          github:
            client-id: {Github OAuth Secret}
            client-secret: {Github OAuth Secret}
            redirect-uri: http://sapier.co.kr/api/login/oauth2/code/github

```

5. Redis

```

spring:
  data:
    redis:
      lettuce:
        pool:
          max-active: 5
          max-idle: 5
          min-idle: 2
      host: {Redis host}
      port: {Redis port}
      password: {Redis password}

```

6. Security

```

jwt.token.key: {jwt key}

```

7. Swagger

```

springdoc:
  version: '@project.version@'
  api-docs:
    path: /api-docs
  default-consumes-media-type: application/json
  default-produces-media-type: application/json
  swagger-ui:
    operations-sorter: alpha
    tags-sorter: alpha
    path: /swagger-ui.html
    disable-swagger-default-url: false
    display-query-params-without-oauth2: true
  paths-to-match:
    - /api/v1/**

```

8. 천원준 Dockerfile, Jenkinsfile

a. JenkinsFolder/springboot.jenkinsfile

```

pipeline{
  agent any

  environment{
    CONTAINER_NAME = "sapier-back-container"
    IMAGE_NAME = "sapier-back-image"
  }
  stages{
    stage('Checkout'){

```

```

    steps{
        checkout scm
    }
}

stage('Build'){
    steps{
        script{
            dir('backend/sapaier'){
                sh 'chmod +x gradlew'
                sh './gradlew clean build'
                sh 'ls -al ./build'
            }
        }
    }
}

stage('Docker Delete'){
    steps{
        script{
            try{
                sh 'echo "Docker Delete Start"'
                sh "docker stop ${CONTAINER_NAME}"
                sh "docker rm -f ${CONTAINER_NAME}"
            }catch(Exception e){
                echo "Docker container ${CONTAINER_NAME} does not exist. skip"
            }
            try{
                //이미지 존재 시 삭제
                sh "docker image rm ${IMAGE_NAME}"
            }catch(Exception e){
                echo "Docker image ${IMAGE_NAME} does not exist. skip"
            }
        }
    }
}

stage('Dockerizing'){
    steps{
        dir('backend/sapaier'){
            sh "echo '파일 구조 확인'"
            sh "ls"
            dir('build'){
                sh "ls"
            }
            sh "docker build -t ${IMAGE_NAME} -f Dockerfile ."
            sh "docker images"
            sh 'echo "images build 성공"'
        }
    }
}

stage('Deploy'){
    steps{
        sh "docker run --name ${CONTAINER_NAME} -d -p 8080:8080 ${IMAGE_NAME}"
        sh "docker ps"
    }
}
}
}

```

b. JenkinsFolder/vue.jenkinsfile

```

pipeline{
    agent any

    environment{
        CONTAINER_NAME = "sapier-front-container"
        IMAGE_NAME = "sapier-front-image"
    }

    stages{
        stage('Checkout'){
            steps{

```

```

        //Jenkins의 SCM 플러그인 사용하여 Git 저장소로부터 소스코드 가져옴
        checkout scm
        sh 'echo "git clone 완료"'
        sh 'echo "현재 디렉토리 경로"'
        sh 'pwd'
    }
}
stage('Docker Delete'){
    steps{
        script{
            try{
                sh 'echo "Docker Delete Start"'
                sh 'docker ps'
                sh 'docker stop ${CONTAINER_NAME}'
                sh 'docker rm -f ${CONTAINER_NAME}'
            } catch(Exception e){
                echo 'Docker container ${CONTAINER_NAME} does not exist. skip'
            }
            try{
                //이미지 존재 시 삭제
                sh 'docker image rm ${IMAGE_NAME}'
            } catch(Exception e){
                echo 'Docker image ${IMAGE_NAME} does not exist. skip'
            }
        }
    }
}
stage('Build'){
    steps{
        script{
            dir('frontend/sapier'){
                sh 'docker build -t ${IMAGE_NAME} -f Dockerfile .'
            }
        }
    }
}

stage('Deploy'){
    steps{
        // sh 'docker cp ${CONTAINER_NAME}:/app/dist /usr/share/nginx/html'
        sh 'docker run -p 3333:3333 --name ${CONTAINER_NAME} -d ${IMAGE_NAME}'
    }
}
}
}

```

c. backend/sapaier/Dockerfile

```

FROM openjdk:17-jdk

ARG JAR_FILE=./build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]

```

d. frontend/sapier/Dockerfile

```

FROM node:18-slim
WORKDIR /app
COPY . .

# pnpm을 설치하고 의존성을 설치합니다.
RUN npm install -g pnpm && pnpm install
EXPOSE 3333

ENTRYPOINT ["pnpm", "dev", "--host"]

```

10. Nginx 설정 (/etc/nginx/conf.d/default.conf)

```

server{
    listen 80;

```

```

    return 301 https://$host$request_uri;
}

server{

    listen 443 ssl;
    server_name sapier.co.kr;

    ssl_certificate /etc/letsencrypt/live/sapier.co.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/sapier.co.kr/privkey.pem;

    location / {
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_pass http://localhost:3333;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /swagger-ui{
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        client_max_body_size 50M;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_buffering off;
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

3. 배포 시 특이사항

▼ Docker Container

1. 사전 설치

```

sudo apt update
//https를 통한 패키지 저장소 접속을 위해, SSL인증서 관련 문제 해결용
sudo apt install apt-transport-https ca-certificates curl software-properties-common

```

2. 자동 설치 스크립트 활용

```

//https://get.docker.com/ 에 있는 스크립트를 다운받겠다는 의미
sudo wget -qO- https://get.docker.com/ | sh

```

3. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```

sudo systemctl start docker //Docker 서비스를 시작하는 명령
sudo systemctl enable docker //시스템 부팅 시 Docker 서비스 자동 시작 설정

```

4. Docker 그룹에 현재 계정 추가

```

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

```

```
//Docker 설치 확인
docker -v
```

- sudo 사용안하고 docker 이용 가능
- docker 그룹은 root권한과 동일하므로 꼭 필요한 계정만 포함

▼ MongoDB

1. 참고문헌

- MongoDB 설치 후 접속 : <https://poiemaweb.com/docker-mongodb>
- MongoDB 관리자 생성, DB 생성 : <https://devfunny.tistory.com/920>

2. Docker MongoDB 이미지 다운로드

```
docker pull mongo
docker images #docker 이미지 확인
```

3. Docker에 MongoDB 컨테이너 생성 및 실행

```
docker run -d --name mongodb-container -v /var/lib/docker/mongo:/data/db -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=password

docker ps -a # docker 컨테이너 목록 출력

# docker 컨테이너 중지/시작/재시작
docker stop mongodb-container
docker start mongodb-container
docker restart mongodb-container

# MongoDB 컨테이너 접속
docker exec -it mongodb-container bash
```

- -d : 컨테이너를 백그라운드에서 실행한다
- -name : 만들어서 사용할 컨테이너의 이름을 정의한다
- -v : 로컬 컴퓨터의 ~/data 디렉터리를 컨테이너가 가지는 /data/db 디렉터리에 마운트 한다
 - 로컬 docker의 mongo가 위치한 /var/lib/docker/mongo를 컨테이너가 가지는 /data/db에 마운트 시켜야 함
- -e MONGO_INITDB_ROOT_USERNAME : mongoDB의 초기 root 계정의 이름을 설정한다
- -e MONGO_INITDB_ROOT_PASSWORD : mongoDB의 초기 root계정의 비밀번호를 설정한다
- -p : 호스트와 컨테이너 간의 포트 연결
- mongo : 사용할 이미지명(위에서 설치한 mongo 이미지명)

4. MongoDB에 database 추가하고 user 권한 설정

```
mongosh -u root -p 1234 # root계정으로 로그인
# 일단은 최초 root 계정으로 sapier(프로젝트명) db 생성해주기
use sapier # 없을때 새로 생성, 있으면 해당 db 사용
db # 현재 사용중인 db 확인하기 위한 명령어
show dbs # 내가 만든 db 리스트 확인 명령어
# 리스트에서 방금 만든 db 보려면, 최소 한개의 document 필요. 그래서 하나 생성
db.book.insertOne({"name":"Sapier", "author":"wonjunchun"});
show dbs

# sapier db 생성했으니, 이제 sapier 접근하는 user 생성해주기
# user 생성은 admin db에서 해줘야 함
use admin # admin db 사용
db.createUser({user:"sapieradmin", pwd:"esfpb301sapier!", roles:[{role:"readWrite", db:"sapier"}, {role:"userAdmin", db:"sapier"}]})
```

▼ Redis

1. Redis 이미지 받기

```
docker pull redis:alpine
```

2. 도커 네트워크 생성

```
docker network create redis-network
# 도커 네트워크 상세정보 확인
docker inspect redis-network
```

3. local-redis 라는 이름으로 로컬 - docker 간 6379 포트 개방, 컨테이너 진입

```
docker run --name local-redis -p 6379:6379 --network redis-network -v /redis_temp:/data -d redis:alpine redis-server --appendonly y
# 도커 컨테이너 확인
docker ps -a
# 컨테이너 진입
# 실행 중인 redis 컨테이너에 대해 docker redis-cli 로 직접 진입
docker run -it --network redis-network --rm redis:alpine redis-cli -h local-redis

# bash로도 진입 가능하다.
docker run -it --network redis-network --rm redis:alpine bash
redis-cli

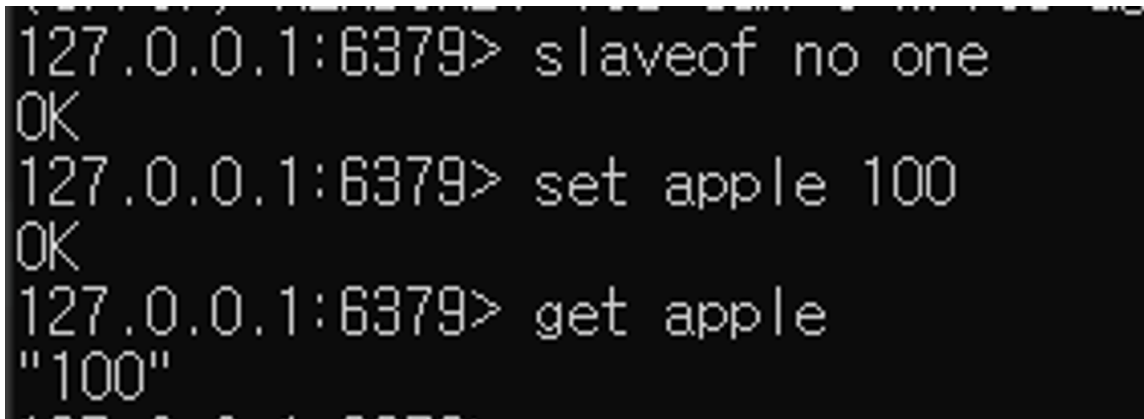
# bash가 안되면 /bin/bash로, 이마저도 안되면 /bin/sh로 대체
```

4. (컨테이너 진입 후)권한 추가

```
# slaveof no one : 현재 슬레이브(복제)인 자신을 마스터로 만듭니다.
127.0.0.1:6379> slaveof no one
```

5. 테스트

- OK가 뜨면 성공



```
127.0.0.1:6379> slaveof no one
OK
127.0.0.1:6379> set apple 100
OK
127.0.0.1:6379> get apple
"100"
```

6. Redis 비밀번호 설정

- <https://peterica.tistory.com/421>

▼ Jenkins

Dockerfile로 Jenkins images 받기(Docker out of Docker, DooD 방식)

- 젠킨스 컨테이너 권한 설정 등 이슈로 root권한 이용할 때 실행

```
docker exec -it -u root jenkinscid /bin/bash
```

- Jenkins 컨테이너 안에서 직접적으로 Docker를 실행하지 않고, 호스트 시스템의 Docker 활용하여 컨테이너 관리 작업 수행
- Dockerfile 작성

```
# 폴더 생성
mkdir config && cd config

vi Dockerfile
```

- Dockerfile

```
FROM jenkins/jenkins:jdk17

#도커를 실행하기 위한 root 계정으로 전환
USER root

#Jenkins 컨테이너 내에서 Docker를 사용할 수 있도록 설정하는 부분
COPY docker_install.sh /docker_install.sh
RUN chmod +x /docker_install.sh
RUN /docker_install.sh

#설치 후 도커그룹의 jenkins 계정 생성 후 해당 계정으로 변경
RUN groupadd -f docker
RUN usermod -aG docker jenkins
USER jenkins
```

- Docker, Jenkins 설정 shell 파일(config 폴더에 vi docker_install.sh 명령을 이용해 파일 만들어서 넣어야 함)

```
#!/bin/sh

# 필요한 패키지 설치
apt-get update && \
apt-get -y install apt-transport-https \
ca-certificates \
curl \
gnupg2 \
zip \
unzip \
software-properties-common

# Docker 관련 설정
# Docker GPG 키 추가
curl -fsSL https://download.docker.com/linux/${. /etc/os-release; echo "$ID"}/gpg > /tmp/dkey
apt-key add /tmp/dkey

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

# 패키지 정보 업데이트
apt-get update

# 필요한 패키지 설치 (이미 Docker가 설치된 상태이므로 설치 과정은 건너뜁니다.)
#apt-get -y install docker-ce

# Jenkins 사용자가 Docker 그룹에 추가되었다고 가정하여 설정
usermod -aG docker jenkins

# Jenkins 컨테이너 내에서 Docker 실행을 위한 디렉토리 및 권한 설정
mkdir /var/jenkinsDir/
chown 1000 /var/jenkinsDir/
```

- Docker 이미지 생성


```
# 현재 디렉토리에 있는 Dockerfile을 사용하여 젠킨스 이미지 빌드
docker build -t jenkins/myjenkins .
```

- Docker 볼륨 폴더 권한 설정

```
# 디렉토리는 Jenkins 컨테이너에서 사용할 데이터 및 설정 저장 위한 목적으로 사용됨
mkdir /var/jenkinsDir/
# Jenkins 컨테이너가 /var/jenkinsDir/ 디렉토리에 쓰기 및 읽기 권한 갖게 됨
sudo chown 1000 /var/jenkinsDir/
```

- Jenkins 컨테이너 생성

```
docker run -d -l 9090:8080 --name=jenkinscid \
-e TZ=Asia/Seoul \
-v /var/jenkinsDir:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/myjenkins
```

Jenkins 초기 세팅 및 테스트(호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신하기 위함)

- 젠킨스 접속 전, `/var/run/docker.sock` 에 대한 권한 설정 필요
 - 초기, 소유자와 그룹 모두 권한이 root였기 때문에 그룹을 root에서 docker로 변경

```
# 컨테이너 접속(root로 접속)
docker exec -it -u root [컨테이너ID] /bin/bash

# 그룹을 바꾸기 위한 명령어
chown root:docker /var/run/docker.sock

# 셸 빠져나옴
exit
```

```
# 빠져나온 후, 컨테이너 재실행
docker restart [컨테이너ID]
# jenkins 패스워드 확인(initialAdminPassword 출력)
docker logs [jenkins 컨테이너 ID]
```

•

```
*****
*****
*****

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

b99f67a2cf054174a73017e4498ce87d

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

*****
*****
*****
```

- k9b301.p.ssafy.io:9090 접속 후 initialAdminPassword 입력하여 다음단계 진행
- 정상 입력했다면 플러그인 설치 단계 진입, Install suggested plugins 선택

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

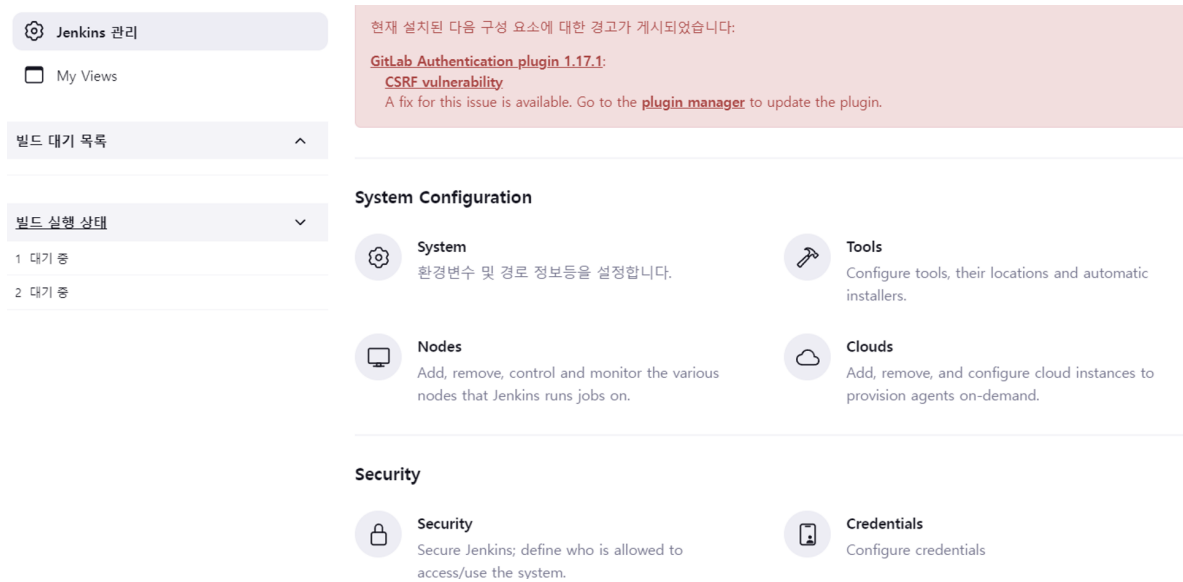
Select and install plugins most suitable for your needs.

✓ Folders	OWASP Markup Formatter	Build Timeout	Credentials Binding	** SSH server Folders ** Trilead API
Timestampers	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		

- 설치 완료 후, Admin 계정 생성창 나오고, 본인이 사용할 정보 입력
- jenkins url : k9b301.p.ssafy.io:9090
- jenkins admin ID : sapieradmin
- jenkins admin PW : esfp*****!
- 이후, Gitlab 및 Docker 플러그인 설치

CI/CD 초기 세팅(우선, Plugins에서 GitLab과 WebHook 설치)

1. Jenkins 관리 → Credentials



2. Add Credentials
3. GitLab Connection
4. Pipeline 구성 → Build when a change is pushed to Gitlab 체크
5. WebHook 등록 위한 Secret Token 생성(이걸로 자동 감지하도록 만들기)

WebHook 설정

- Jenkins 트리거 체크

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k9b301.p.ssafy.io:9090/project/sapier-backend> ?

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 ?

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급 ▼

☐ Generic Webhook Trigger ?

- develop 브랜치에 push 될때만 감지

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssafy.com/s09-final/S09P31B301

Credentials ?

chunjh1103@naver.com/*****

+ Add

고급

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/develop

- Jenkins 파이프라인 스크립트 위치 지정

Repository browser ?

(자동) ▼

Additional Behaviours

Add ▼

Script Path ?

JenkinsFolder/springboot.jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

▼ Backend

- 위에 올린 JenkinsFolder/springboot.jenkinsfile , backend/sapaier/Dockerfile 기준으로 설명
- 컨테이너명, 이미지명 설정

```
environment{
    CONTAINER_NAME = "sapier-back-container"
    IMAGE_NAME = "sapier-back-image"
}
```

- 우선, Checkout 단계

```
stage('Checkout'){
    steps{
        //Jenkins의 SCM 플러그인 사용하여 Git 저장소로부터 소스코드 가져옴
        checkout scm
    }
}
```

- Build 단계
 - gradlew에 실행권한 추가
 - backend/sapaier 경로(backend root경로)에서 클린 빌드 실행

```
stage('Build'){
    steps{
        script{
            dir('backend/sapaier'){
                sh 'chmod +x gradlew'
                sh './gradlew clean build'
                sh 'ls -al ./build'
            }
        }
    }
}
```

- Delete 단계
 - 현재 실행중인 컨테이너를 중지 및 제거
 - 이전에 만든 이미지 존재 시 삭제

```

stage('Docker Delete'){
  steps{
    script{
      try{
        sh 'echo "Docker Delete Start"'
        sh "docker stop ${CONTAINER_NAME}"
        sh "docker rm -f ${CONTAINER_NAME}"
      }catch(Exception e){
        echo "Docker container ${CONTAINER_NAME} does not exist. skip"
      }
      try{
        //이미지 존재 시 삭제
        sh "docker image rm ${IMAGE_NAME}"
      }catch(Exception e){
        echo "Docker image ${IMAGE_NAME} does not exist. skip"
      }
    }
  }
}

```

- Dockerizing 단계
 - backend/sapaier 경로에 있는 Dockerfile을 이용해 이미지 빌드함

```

stage('Dockerizing'){
  steps{
    dir('backend/sapaier'){
      sh "echo '파일 구조 확인'"
      sh "ls"
      dir('build'){
        sh "ls"
      }
      sh "docker build -t ${IMAGE_NAME} -f Dockerfile ."
      sh "docker images"
      sh 'echo "images build 성공"'
    }
  }
}

```

- 참고로, Dockerfile는 다음과 같음

```

# Dockerfile
FROM openjdk:17-jdk

ARG JAR_FILE=./build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]

```

- Deploy 단계
 - Dockerizing 단계에서 새로 빌드한 이미지 이용해 컨테이너 생성

```

stage('Deploy'){
  steps{
    sh "docker run --name ${CONTAINER_NAME} -d -p 8080:8080 ${IMAGE_NAME}"
    sh "docker ps"
  }
}

```

▼ Frontend

- 위에 올린 JenkinsFolder/vue.jenkinsfile , frontend/sapier/Dockerfile 기준으로 설명
- 컨테이너명, 이미지명 설정

```
environment{
    CONTAINER_NAME = "sapier-front-container"
    IMAGE_NAME = "sapier-front-image"
}
```

- Checkout 단계

```
stage('Checkout'){
    steps{
        //Jenkins의 SCM 플러그인 사용하여 Git 저장소로부터 소스코드 가져옴
        checkout scm
        sh 'echo "git clone 완료"'
        sh 'echo "현재 디렉토리 경로"'
        sh 'pwd'
    }
}
```

- Docker Delete 단계

- 기존 컨테이너와 이미지 제거

```
stage('Docker Delete'){
    steps{
        script{
            try{
                sh 'echo "Docker Delete Start"'
                sh 'docker ps'
                sh 'docker stop ${CONTAINER_NAME}'
                sh 'docker rm -f ${CONTAINER_NAME}'
            } catch(Exception e){
                echo 'Docker container ${CONTAINER_NAME} does not exist. skip'
            }
            try{
                //이미지 존재 시 삭제
                sh 'docker image rm ${IMAGE_NAME}'
            } catch(Exception e){
                echo 'Docker image ${IMAGE_NAME} does not exist. skip'
            }
        }
    }
}
```

- Build 단계(Dockerfile 기준으로 이미지 빌드함)

```
stage('Build'){
    steps{
        script{
            dir('frontend/sapier'){
                sh 'docker build -t ${IMAGE_NAME} -f Dockerfile .'
            }
        }
    }
}
```

- Dockerfile는 다음과 같음

```
FROM node:18-slim
WORKDIR /app
COPY . .

# pnpm을 설치하고 의존성을 설치합니다.
RUN npm install -g pnpm && pnpm install
EXPOSE 3333

ENTRYPOINT ["pnpm", "dev", "--host"]
```


- Deploy 단계
 - 빌드한 이미지 기준으로 컨테이너 생성 후 실행

```
stage('Deploy'){
  steps{
    // sh 'docker cp ${CONTAINER_NAME}:/app/dist /usr/share/nginx/html'
    sh 'docker run -p 3333:3333 --name ${CONTAINER_NAME} -d ${IMAGE_NAME}'
  }
}
```

4. DB 접속 정보 및 ERD에 활용되는 주요 계정 및 프로퍼티 정의

1. Mongodb

- a. host : `k9b301.p.ssafy.io`
- b. port : `27017`
- c. username : `sapieradmin`
- d. password : `esfpb301sapier!`
- e. database : `sapier`

2. Redis

- a. host : `k9b301.p.ssafy.io`
- b. port : `6379`
- c. password : `esfpb301sapier!`